CMS/CAIMS Books in Mathematics

Canadian
Mathematical Society
Société mathématique
du Canada

David E. Stewart

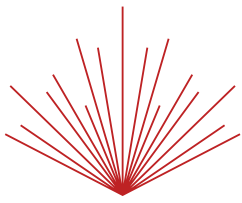
# Numerical Analysis: A Graduate Course

CAIMS
SCMAI

# CMS/CAIMS Books in Mathematics

Volume 4

CMS/CAIMS Books in Mathematics is a collection of monographs and graduate-level textbooks published in cooperation jointly with the Canadian Mathematical Society- Societé mathématique du Canada and the Canadian Applied and Industrial Mathematics Society-Societé Canadienne de Mathématiques Appliquées et Industrielles. This series offers authors the joint advantage of publishing with two major mathematical societies and with a leading academic publishing company. The series is edited by Karl Dilcher, Frithjof Lutscher, Nilima Nigam, and Keith Taylor. The series publishes high-impact works across the breadth of mathematics and its applications. Books in this series will appeal to all mathematicians, students and established researchers. The series replaces the CMS Books in Mathematics series that successfully published over 45 volumes in 20 years.

David E. Stewart

# Numerical Analysis: A Graduate Course

Springer

David E. Stewart
Department of Mathematics
University of Iowa
Iowa, IA, USA

ISSN 2730-650X ISSN 2730-6518 (electronic)
CMS/CAIMS Books in Mathematics
ISBN 978-3-031-08120-0 ISBN 978-3-031-08121-7 (eBook)
https://doi.org/10.1007/978-3-031-08121-7

Mathematics Subject Classification: 65-01



This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

# Preface

Numerical analysis has advanced greatly since it began as a way of creating methods to approximate answers to mathematical questions. This book aims to bring students closer to the frontier regarding the numerical methods that are used. But this book is not only about newer, as well as "classical", numerical methods. Rather the aim is to also explain how and why they work, or fail to work. This means that there is a significant amount of theory to be understood. Simple analyses can result in methods that usually work, but can then fail in certain circumstances, sometimes catastrophically. The causes of success of a numerical algorithm and its failure are both important. Without understanding the underlying theory, the reasons for a method's success and failure remain mysterious, and we do not have a means to determine how to fix the problem(s).

In this way, numerical analysis is a dialectic[1] between practice and theory; the practice being computation and programming, and the theory being mathematical analysis based on our model of computation. While we do indeed prove theorems in numerical analysis, the assumptions made in these theorems may not hold in many situations. Also, the conclusions may involve statements of the form "as $n$ goes to infinity" (or "as $h$ goes to zero") while in actual computations $n$ might not be especially large (or $h$ especially small). Numerical analysis will sometimes ignore errors that we know exist (like roundoff error in the analysis of a method for solving differential equations). This is usually based on an understanding that some sources of errors are insignificant in a particular situation. Of course, there will be situations where roundoff error should be considered in the solution of differential equations, but only if the step size becomes unusually small. In that case, new analyses, and even new methods, may be necessary.

[1] *Dialectic* is a dialog between a claim (a *thesis*) and counter-claims (an *antithesis*) hopefully leading to a new understanding (a *synthesis*) that incorporates both the original claim and the counter-claims. The synthesis is expected to give further understanding, but will itself eventually meet counter-claims.

Inspiration for this book has been found in the books of Atkinson and Han [11], Atkinson [13], Sauer [228], and Stoer and Bulirsch [241]. However, we wish to include current issues and interests that were not addressed in these books.

We aim to present numerical methods and their analysis in the context of modern applications and models. For example, the standard asymptotic error analysis of differential equations gives no advantage to implicit methods, which have a much larger computational cost. But for "stiff" problems there is a clear, and often decisive, advantage to implicit methods. While "stiffness" can be hard to quantify, it is also common in applications. We also wish to emphasize multivariate problems alongside single-variable problems: multivariate problems are crucial for partial differential equations, optimization, and integration over high-dimensional spaces. We deal with issues regarding randomness, including pseudo-random number generators, stochastic differential equations, and randomized algorithms. Stochastic differential equations meet a need for incorporating randomness into differential equations. High-dimensional integration is needed for studying questions and models in data science and simulation.

To summarize, I believe numerical analysis must be understood and taught in the context of applications, not simply as a discipline devoted solely to its own internal issues. Rather, these internal issues arise from understanding the common ground between analysis and applications. This is where the future of the discipline lies.

I would like to thank the many people who have been supportive of this effort, or contributed to it in some way. I would like to thank (in alphabetical order) Jeongho Ahn, Kendall Atkinson, Bruce Ayati, Ibrahim Emirahmetoglu, Koung-Hee Leem, Paul Muhly, Ricardo Rosado-Ortiz, and Xueyu Zhu. My wife, Suely Oliveira, has a special thanks for both encouraging this project and having the patience for me to see it through. Finally, I would like to thank the staff at Springer for their interest and support for this book, most especially Donna Chernyk.

## *How to Use This Book*:

Numerical analysis is a combination of theory and practice. The theory is a mixture of calculus and analysis with some algorithm analysis thrown in. Practice is computation and programming. The algorithms in the book are shown as pseudo-code. Working code for MATLAB and/or Julia can be found at https://github.com/destewart2022/NumerAnal-Gradbook. The exercises are intended to develop both, and students need practice at both.

Like most intellectual disciplines, numerical analysis is more a spiral than a straight line. There is no linear ordering of the topics that makes complete sense. Thus teaching from this book should not be a matter of starting at one cover and ending at the other. In any case, there is probably too much material and an instructor must of necessity choose what they wish to emphasize. Matrix

computations are foundational, but even just focusing on this could easily take a semester or more. Differential equations arise in many, many applications, but the technical issues in partial differential equations can be daunting. The treatment here aims to be accessible without "dumbing down" the material. Students in data science may want to focus on optimization, high-dimensional approximation, and high-dimensional integration. Randomness finds its way into many applications, whether we wish it or not. So, here is a plan that you might consider when you first teach from this book:

- Chapter 1: A little on computing machinery to get started, floating point arithmetic, norms for vectors and matrices, and at least one-variable Taylor series with remainder.
- Chapter 2: LU factorization for linear systems, linear least squares via the normal equations and the QR factorization as a black box. Eigenvalues can wait until later.
- Chapter 3: Bisection, fixed-point, Newton, and secant methods are the foundation; guarded multivariate Newton and one-variable hybrid methods give good examples of how to modify algorithms for better reliability.
- Chapter 4: Polynomial interpolation is so central that you need to cover this well, including the error formula and the Runge phenomenon; the Weierstrass and Jackson theorems (without proof) give a sense of rate of convergence. Cubic splines give useful alternatives to plain polynomials. Lebesgue numbers may appear abstract, but give a good sense of the reliability of interpolation schemes. Radial basis functions give an entry into high-dimensional approximation.
- Chapter 5: Simple ideas can go a long way, but "integrate the interpolant" is a central idea; multivariate integration is also valuable here if you want to use it for partial differential equations.
- Chapter 6: Basic methods for solving ordinary differential equations are still very useful, although the revolution brought about but John Butcher's approach to Runge–Kutta methods is worth a look—if you have time. Partial differential equations need some more set-up time, but are worthwhile for more advanced students, or a second time around. The scale of the problems for partial differential equations means that you should point your students back to Chapter 2 on how to solve large linear systems.
- Chapter 7: Randomness is important, and some statistical computation using SVDs may be a doorway to these issues. Random algorithms are also very important, but often involve advanced ideas.
- Chapter 8: Optimization has become more important in a number of areas, and including it is an option to consider. If your students want to do machine learning, some outline of the algorithms is available here.

A second course could be focused on specific issues. A machine learning focus could include iterative methods and SVDs for matrix computations in Chapter 2, radial basis functions from Chapter 4, high-dimensional integration from Chapter 5, methods for large-scale optimization from Chapter 8, rounded out with some

analysis of random algorithms from Chapter 7. A simulation-based course would focus on approximation and interpolation in two or three dimensions from Chapter 4, multi-dimensional integration from Chapter 5, much of the material from Chapter 6 on differential equations, both ordinary and partial. Uncertainty quantification could be served by starting with Chapter 7, progressing to iterative methods for large linear systems in Chapter 2, radial basis functions in Chapter 4, perhaps some partial differential equations in Chapter 6 and optimization in Chapter 8.

## *Notes on Notation*:

Scalars are usually denoted by lower case italic letters (such as $x$, $y$, $z$, $\alpha$, $\beta$) while vectors are usually denoted by lower case bold letters (such as $\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{z}$, $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$). Matrices are usually denoted by upper case italic letters ($X$, $Y$, $A$, $B$). Entries of a vector $\boldsymbol{x}$ are scalars, and so denoted $x_i$; entries of a matrix $A$ are denoted $a_{ij}$. Matrices and vectors usually have indexes that are integers $i = 1, 2, \ldots, m$, $j = 1, 2, \ldots, n$ for an $m \times n$ matrix. Sometimes it is convenient to have index sets that are different, so a matrix $A = [a_{ij} | i \in R, j \in C]$ can have row index set $R$ and column index set $C$, and $\boldsymbol{x} = [x_i | i \in C]$ has index set $C$. This means $A\boldsymbol{x} = [\sum_{j \in C} a_{ij} x_j | i \in R]$ is the matrix–vector product. Just as $A^T$ is used to denote the transpose of $A$, $A^{-T}$ is the inverse of $A^T$, which is also the transpose of $A^{-1}$.

Functions are usually introduced by naming them. For example, the squaring function can be introduced as $q(x) = x^2$. Functions of several variables can be introduced as $f(x, y) = x^2 + y^2$ or using vectors as $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{x}$. Anonymous functions can be introduced as $\boldsymbol{x} \mapsto \boldsymbol{x}^T\boldsymbol{x}$.

Pseudo-code uses "$\leftarrow$" to assign a value to a variable (such as $x \leftarrow y$ assigns the value of $y$ to $x$) while "=" tests for equality (where $x = y$ returns *true* if $x$ and $y$ have the same value).

In some occasions, ":=" is used to define a quantity of function where using "=" might be ambiguous. The sets $\mathbb{R}$, $\mathbb{C}$, and $\mathbb{Z}$ are understood to be the set of real numbers, the set of complex numbers, and the set of integers, respectively.

Iowa, USA David E. Stewart

# Contents

# Chapter 1
# Basics of Numerical Computation


There was a time when computers were people [110]. But since the late 1950s with the arrival and development of electronic digital computers, people who did the calculations were being replaced by machines. The history of computing in NASA revealed in *Hidden Figures* [233] gives an example of this. The machines have advanced enormously since that time. Gordon Moore, one of the founders of Intel, came up with his famous "law" in 1965 that the number of transistors on a single "chip" of silicon doubled every year [179]. Since then the time for doubling has stretched out from 1 year to over 2 years, but the exponential growth of the capabilities of computer hardware has not yet stopped.

Numerical computations have been an essential part of the work done by computers since the earliest days of electronic computers. An understanding of how they work is essential if you want to get the best performance out of your computer. And the scale of numerical computations can be truly enormous. In this chapter, some aspects of how computers function will be explained, along with how that affects numerical methods and numerical software.

## 1.1 How Computers Work

The heart of any computer is the *Central Processing Unit* (CPU). Since the 1980s they are usually on a single piece of silicon—on a single "chip". Other essential parts of a computer are its memory, which almost always includes fast-access writable memory (usually referred to as RAM for *Random Access Memory*), and permanent storage (such as a hard disk drive or solid-state drive), and means of communicating with the outside world. Communicating with the outside world can be through keyboard, mouse, Wi-Fi, and display for your laptop, a direct Internet connection, or a data link to a large-scale storage system. Perhaps you have other sensors, like a camera,



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_1

a microphone, as well as sensory output devices, such as a headphone set. However it communicates with the outside world, it is essential to do so.

But here we focus on the "beating heart" of any computer: the CPU. This is where the computations are done. How a CPU is organized and performs its computations is the subject of computer architecture. There are many books on this subject, such as [24, 192].

### 1.1.1 The Central Processing Unit

Modern computers are sometimes referred to as *electronic digital stored program computers*. *Electronic* refers to the physical nature of the data in the computer. *Digital* means that only discrete values of these physical values are important, rather than an *analog* computer which uses continuous values of voltage, for example, to represent the computed quantities. *Stored Program* means that the task (the *program*) is stored in memory, rather than being physically fixed.

The fact of a computer being a *stored program* computer means that the CPU has to retrieve the operations to perform as well as the data to apply the operations to. This also means that the CPU has to have a *control unit* for decoding the instructions in the program, as well as an *arithmetic and logic unit* (ALU) which does the actual computations. In addition, the CPU has its own fast-access memory. Every CPU has, at a minimum, a number of *registers* which are fixed small units of memory that are easily accessed to computations.

### 1.1.2 Code and Data

The instructions are in *machine code*, which is the basic "language" that the CPU understands and executes. The CPU reads an instruction from memory, and then sets about executing the instruction by sending signals to the arithmetic and logic unit, the memory system, or some communication system. Executing an instruction will typically involve reading some data item from a register, sending it to the appropriate input of the arithmetic and logic unit, performing the appropriate operations, and sending the result back to a certain register to await the next operation, or being written to memory. Some operations simply read a specified memory location and stores the result in a specified register. Other operations do the reverse: saves the contents of a specified register in a specified memory location.

The machine code used by a CPU is an important part of its design. Intel has been producing CPUs for longer than any other current producer (as of time of writing). Intel's commercial strategy has dictated that new CPUs in their line have essentially the same machine code as older CPUs, bolstered by a few new instructions to provide some additional features. This is an example of *backwards compatibility*, where old computer code can still be executed without change. As a result, while Intel CPUs

have many thousands times as many transistors as their CPUs from the 1980s, they run essentially the same machine code. On the other hand, there was a revolution in 1990s in CPU design which emphasized the simplicity of their architecture. These CPUs were called RISC for reduced instruction set computers. The current processors in this family are the ARM (for advanced RISC machine) CPUs [84]. ARM CPUs are currently found in cell phones and tablets and other devices such as Raspberry Pi systems. Their chief advantage for these applications is low power consumption. Other instruction sets are available, such as the public domain RISC V (pronounced "risk-5") instruction set [199]. There are a number of hardware implementations of the RISC V instruction set, although few are ready-to-use computers.

#### 1.1.2.1 Read–Execute Cycle

The read–execute cycle is the cycle of reading an instruction, decoding the instruction(s), and executing the instructions. This cycle is controlled by a clock. Ideally, a basic instruction can be executed in one clock cycle. However, the chase after ever shorter clock cycles has resulted in instructions executing over multiple clock cycles while the CPU can still complete close to one instruction each clock cycle. To achieve this, the CPU uses a *pipelined architecture*: computations are spread over multiple units. Each unit carries out a basic step in one clock cycle, then passes its results to the next unit in the pipeline.

The vast numbers of transistors in modern CPUs means that there are opportunities to exploit parallelism across the different components of the CPU. This might take a basic form, such as where an integer processing unit is used to control a `for` loop, while a floating point processing unit is simultaneously used to carry out the main computations in the body of the loop. Multiple computational units can also be harnessed in parallel, forming a *vector architecture*. These possibilities provide opportunities for greater performance, but require more complex control units in the CPU.

#### 1.1.2.2 Memory Hierarchies

One of the bottlenecks in modern CPUs is reading and writing data from or to memory. More transistors on a chip means smaller transistors that are closer together. This means that switching times are shorter, and signal transmission times within the CPU are shorter. However, the time needed to send data to RAM or read data from RAM has not become much shorter. As a result, the time needed just to store or read data has become much larger than the time for a floating point operation.

To deal with this, computer architecture has developed memory hierarchies, similar to that shown in Figure 1.1.1. At the top of the hierarchy is the small, fast memory of registers. Typically, the contents of a register are available in a single clock cycle. The next level(s) consists of *cache memory*. This memory in level 1 cache can be read from or written to a register in one to a few clock cycles depending on the specific

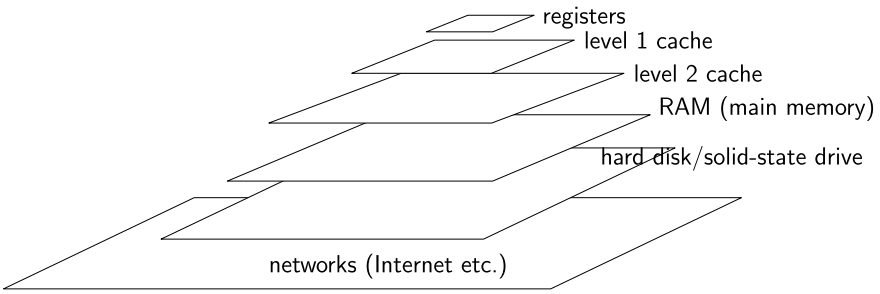



**Fig. 1.1.1** Memory hierarchy

architecture. Rather than require the programmer to remember if a piece of data is currently in cache memory, the hardware keeps track of this. If a requested data item is not in level 1 cache, the system then looks in level 2 cache. If the requested item is in level 2 cache it will be transferred to level 1 cache, and can then be transferred to a register for computations on it. Cache memory is organized into *cache lines*. Each cache line is a single block of consecutive memory locations. The size of this block can depend on the level of the cache. Level 1 cache typically (at time of writing) has a cache line size of 64 bytes. Some architectures have a level 2 cache line size of 128 bytes.

If a data item is not in level 2 cache memory, then the system goes to level 3 cache (if present), or to main memory (RAM). Since RAM is not on the chip itself, the request goes to another piece of silicon on the same printed circuit board. Instead of taking dozens of clock cycles to obtain a data item from level 2 cache, obtaining a data item from RAM typically takes hundreds of clock cycles. Data transferred from RAM is typically transferred in much larger blocks than from cache memory.

Almost all modern computers implement *virtual memory*, which means that not all data in use by the computer is stored in RAM. Some memory blocks not in RAM can be stored in more permanent memory such as the hard disk drive or solid-state drive of the computer. In this way, the computer can work with larger data sets directly without having explicit instructions to save data in this way. The time to access a data item on a hard disk is typically measured in milliseconds and thus millions of clock cycles in current computers.

Beyond the permanent storage of a computer is the memory that is accessible over networks, such as the Internet. This gives the computer access to much larger memory systems, but the time to access them is again much larger—measured in seconds or billions of clock cycles, rather than milliseconds for permanent storage.

This creates a hierarchy of memory systems called a *memory hierarchy*. At each level of this hierarchy, there has been a way of remembering where in main memory, or the *virtual address space*, each data item belongs. This is done using a *tag* for each block of memory and a *translation look-aside buffer* (TLB) containing the tags and other data for each block of memory at a given level. There also needs to be a bit for each block of memory indicating whether it needs to be written back to the next

lower level of memory or not. If a block needs to be written back to the next level, it is called *dirty.*

Since each level of the memory hierarchy contains a fixed amount of memory, when a new block is read in, there must be a way of deciding which block of memory must be removed. If that block is "*dirty*" then it must be written back to the next lower level of memory first. Various algorithms are used to decide which block of memory is to be removed. The most common of these is the *least recently used* (LRU) heuristic. Implementing this correctly in a deep memory hierarchy requires co-ordination between the different levels.

As we go down the memory hierarchy, the amount of memory available increases enormously, while the access times also increase enormously. The *rate* of data transfer decreases significantly as we go down the memory hierarchy as well. This means that to obtain best performance, data should be re-used at the top levels of the memory hierarchy as much as possible.

#### 1.1.2.3 Thrashing

*Thrashing* is what happens when you are trying to fit too much into a level of the memory hierarchy. This can happen at any level of the memory hierarchy and was first noticed in dealing with virtual memory. Virtual memory makes it appear to a programmer that there is more memory available to a program than can fit in main memory by using permanent storage, like a hard disk. If accessing data that is outside of main memory is done sporadically, then there is little cost to this. But if the program is regularly scanning the entire data set, then data transfers from permanent storage to main memory are done with every step of the program. The program slows down to the speed with which data can be transferred from permanent storage. This can slow down programs by a factor of 1,000 or more.

Thrashing can also happen between cache memory and main memory. If the program is trying to scan a data set that does not fit in cache memory, then the computer will find itself loading data from main memory into cache memory with every step of the program. The program will slow down to the speed with which data can be transferred from main memory, slowing the program by a factor of 10 to 100.

The way to reduce thrashing is to re-use data when loaded as much as possible, and to use data that is nearby in memory as much as possible.

#### 1.1.2.4 Code Versus Data Caches

Many modern CPU architectures include separate level 1 caches for data and code. Because of the different memory access patterns for code and for data, this is often a good idea. Code usually should not change during execution, so there is often no need for "*dirty*" bit in a cache for code. Modern CPU architectures also often have a specialized kind of cache for code called a *micro-operation cache* or *$\mu$op cache*. The idea is that the front end of the control unit decodes machine code instructions into

a stream of *micro-operations* (or $\mu ops$) that can be more directly executed by the hardware. The back end of the control unit reads this stream of micro-instructions and can parallelize or re-order these micro-instructions to better use the hardware.

Short loops can often be converted into a short sequence of micro-operations that can fit into the $\mu$op cache. This avoids the need for further memory access to get instructions for the duration of the loop.

#### 1.1.2.5 Multi-core CPUs

As the number of transistors in a CPU has grown, there are more opportunities for parallel computation. Multiple functional units in the arithmetic-logic unit were an early sign of this, but in the past decade, it has become more explicit with CPUs now having multiple cores. A *core* is essentially a mini-CPU with its own control unit, arithmetic-logic unit(s), registers, and cache. However, multiple cores share the same "chip" as well as higher level cache, connections to main memory, and to the outside world.

When a CPU has multiple functional units, the programmer does not have to be aware of them in order to use them: the CPU hardware identifies opportunities for this form of parallelism and uses them. Multiple cores, however, require some kind of explicit parallel computing. Programming with *threads* is the most common way of exploiting multi-core CPUs, although this is not the only way to do so. There are other reasons for programming with threads—for example, having one thread in a program doing computations, while another thread handles user interactions.

More information about parallel computing can be found in Section 1.1.6.

### *1.1.3 On Being Correct*

As a practical matter, it is important for your algorithms and implementations to be correct. Ideally, we would prove that our algorithms are correct before we implement them. And once they are implemented, prove that our implementation is correct. Indeed, *proof of correctness* techniques are valuable tools in the arsenal of software engineers in their fight against bugs. However, it is infuriatingly common to look at some code we have just written and not see some flaw that another person would see in a moment. Or a simple test would reveal.

It is therefore imperative to *test your code*. Build tests alongside your code, or even before you code. Test *as* you code. Try to avoid writing a large piece of software and *then* test it, because the largest part of the art of debugging is finding the error. Once you find the error, it is usually easy to understand why it is wrong. Most times the fix is also quite clear. Sometimes the fix is not clear, in which case there may be a deeper misunderstanding about your algorithm that needs to be resolved first. But if you have a large piece of software, you have a much larger area to search if the results are incorrect.

If you are modifying an already tested and accepted piece of code, keep the old code for comparison. Revision control systems (such as Git, and older systems such as the Unix RCS system) are useful for doing this. As instances of bugs arise, add an extra test to your list of tests.

### *1.1.4 On Being Efficient*

While computing hardware has become more efficient, the software—the algorithms and data structures—have also become more efficient. Of course, for many computing tasks, such as sorting a list of data items, there are some hard limits on how efficient the algorithm can be.

But a numerical analyst should understand the efficiency of algorithms and how that relates to computing practice.

#### 1.1.4.1 Big "Oh" Notation

The computational resources needed by an algorithm are often described using "big-Oh" asymptotic notation. The time needed (or *time complexity*) to process an input with $n$ data items might be a function $T(n)$, but it is often very difficult to obtain a precise formula for $T(n)$, which in any case can vary with the CPU, the clock speed, the operating system, the programming language, the particular compiler or interpreter used, or even various options used by the compiler or interpreter. So it is often better to simply give an asymptotic estimate for the time taken of the form "$T(n) = \mathcal{O}(g(n))$ as $n \to \infty$" which means

$$\text{there are constants } C, n_0 > 0 \text{ where } n \geq n_0 \text{ implies } T(n) \leq C\, g(n).$$

The constants $C$ and $n_0$ are called "hidden constants" as they are not mentioned in the "$\mathcal{O}$" statement "$T(n) = \mathcal{O}(g(n))$". These hidden constants are often dependent on the hardware and software used as well as the way the method is implemented. Interpreted languages (like Basic, Python, or Ruby) are typically slower than compiled languages (like C/C++, Fortran, or Go), which changes the value of the constant $C$. Doubling the clock speed halves the value of $C$. Compiler or interpreter optimization as well as the details of how an algorithm is implemented can strongly affect the value of $C$. The cache size can affect the value of $n_0$ as well as $C$.

Because of all these issues, it is better to tell the reader something that does *not* depend on implementation details, like "*mergesort takes* $\mathcal{O}(n \log n)$ *time to sort n items*". This is true whether mergesort is implemented in Basic on a 1990-vintage Mac, or a C++ version on the latest supercomputer, even if the times taken by the two implementations are very different. We know roughly how the time taken will change as the value of $n$ becomes large.

**Table 1.1.1** Asymptotic resource usage for some sorting algorithms for $n$ items

| Algorithm | Time | Memory | Extra memory |
|---|---|---|---|
| bubblesort | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| insertion sort | $\mathcal{O}(n^2)$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| shellsort | $\mathcal{O}(n\sqrt{n})$ | $\mathcal{O}(n)$ | $\mathcal{O}(1)$ |
| with gap sequence $2^{\lfloor 2^{-k-1}n \rfloor} + 1$ | | | |
| quicksort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(\log n)$ |
| mergesort | $\mathcal{O}(n \log n)$ | $\mathcal{O}(n)$ | $\mathcal{O}(n)$ |

Variations of this notation look at what happens as a parameter becomes small. For example, the step size $h$ used for approximately solving differential equations should be small. We say "$f(h) = \mathcal{O}(g(h))$ as $h \downarrow 0$" means that there are $C$ and $h_0 > 0$ where

$$|f(h)| \leq C\, g(h) \qquad \text{provided } 0 < h < h_0.$$

Lower bounds are designated using "big-$\Omega$" notation: "$f(n) = \Omega(g(n))$ as $n \to \infty$" means there are $C$ and $n_0$ where

$$f(n) \geq C\, g(n) \qquad \text{provided } n \geq n_0.$$

Finally "big-$\Theta$" notation combines these: "$f(n) = \Theta(g(n))$ as $n \to \infty$" means that both $f(n) = \mathcal{O}(g(n))$ and $g(n) = \mathcal{O}(f(n))$ as $n \to \infty$. Analogous definitions hold as the input parameter becomes small. For example,

$$e^x = 1 + x + \frac{1}{2}x^2 + \Theta(x^3) \qquad \text{as } x \to 0$$

from the Taylor series for the exponential function.

### 1.1.4.2 Using Efficiency Measures

We use the "big-Oh" notation for measuring the performance of algorithms. We can use this for measuring and reporting the amount of time needed by an algorithm, and the amount of memory needed. Table 1.1.1 shows the "big-Oh" measures of the time and memory needed for some well-known sorting algorithms (see [59]).

The time needed for quicksort and mergesort is asymptotically equivalent. Furthermore, what the table does not show is that this is guaranteed for mergesort, but for quicksort it is only for the average value over randomized choices during the algorithm. Why then is quicksort so much more popular than mergesort?[1] Because

[1] Yes, quicksort is more popular. For example, the standard C library includes a quicksort function, but no mergesort function.

**Algorithm 1** Recursive factorial function

```
1  function factorial(n)
2    if n = 0:       return 1
3    else if n > 0:  return n · factorial(n − 1)
4    else:           return 0
5    end if
6  end function
```

mergesort requires substantial ($\Theta(n)$) extra memory, which must be allocated, and later deallocated. The quicksort algorithm does require $\mathcal{O}(\log n)$ extra memory, but this is in the quicksort recursion, and so does not require explicit allocation. This makes quicksort a more practical and attractive algorithm. With quicksort, you can also push the size of the input to near the maximum amount of memory available without creating memory overflow.

Using efficient algorithms is the start to making great and efficient programs. Use standard implementations first, such as available in the standard library for your programming language.

#### 1.1.4.3 Optimizing Code

*Premature optimization is the root of all evil.*

Donald Knuth

For programmers with some experience, it is tempting to make each piece of code efficient. This can be a mistake. Each additional "optimization" can

- make the code more confusing,
- introduce hidden dependencies (or worse, outright bugs), and
- require careful documentation.

It is better to write code at the level of generality appropriate for the algorithm being implemented. If you need the code to be faster in a particular situation, *profile* the code first to see where the code is actually spending its time. Focus attention on that part of the code. Keep your original code. Then you can check the results and performance of your new code against the old code to ensure that they do the same thing (or are, at least, consistent).

### *1.1.5 Recursive Algorithms and Induction*

A *recursive function* is a function that, either directly or indirectly, calls itself. A standard example is the factorial function where $n! = n \cdot (n-1) \cdot (n-2) \cdots 3 \cdot 2 \cdot 1$ for $n = 1, 2, \ldots$ and $0! = 1$ in Algorithm 1.

**Algorithm 2** Recursive summation

```
1   function sum_recursive(a, i, j)
2     if i > j:       return 0
3     else if i = j: return a_i
4     else
5       m ← ⌊(i + j)/2⌋ // (i + j)/2 rounded down
6       return sum_recursive(a, i, m) + sum_recursive(a, m + 1, j)
7     end if
8   end function
```

#### 1.1.5.1 Principles of Induction

The principle of mathematical induction is complementary to recursive functions in computing. Suppose that $P(x)$ is a condition on $x$, $x$ a non-negative integer, satisfies the properties that $P(0)$ is true, and $P(k)$ true implies $P(k+1)$ true as well. The *principle of induction* states that any such condition is true for *all* non-negative integers: $P(x)$ is true for $x = 0, 1, 2, 3, \ldots$. The *principle of complete induction* applies to any condition $P(x)$ that is true for $x = 0$ and $P(j)$ is true for all non-negative integers $j$ between zero and $k$ (inclusive) implies $P(k+1)$ is also true. Again, if $P(x)$ is a condition on $x$ having these properties, then $P(x)$ is true for $x = 0, 1, 2, 3, \ldots$. The principle of complete induction can be proved by applying the principle of induction to the condition $Q(x)$ which states that $P(y)$ is true for all integers $0 \leq y \leq x$.

How do we know that the *factorial* function does indeed compute the factorial of $n$? For $n = 0$ we get the correct value from the first `if`. If we compute the correct value for $n = k$ with $k$ a non-negative integer, then for $n = k + 1$ the computed value is $(k+1) \cdot \mathit{factorial}(k)$ which is, by the induction hypothesis $(k+1) \cdot (k!) = (k+1)!$ as we wanted. Thus, by the principle of mathematical induction, it computes the correct value for $n = 0, 1, 2, 3, \ldots$.

For another example, consider the recursive algorithm for computing a sum $\sum_{k=i}^{j} a_k$ in Algorithm 2.

To prove that this correctly computes the sum, we let $n = (j - i + 1)$ which is the number of terms of the sum for $n \geq 0$. Let $P(n)$ be the condition that the sum of $n$ terms is computed correctly by *sum_recursive*(). This is clearly true for $n \leq 0$ as then the first `return` is taken. If $n = 1$ then $P(1)$ is true as the sum is just $a_i$ as returned by the second `return`. Now suppose that $P(k)$ is true for all $0 \leq k \leq n$. We want to show that $P(n+1)$ is also true. Now *sum_recursive*$(a, i, m)$ is computed correctly as the number of terms in the sum is $m - i + 1 \leq n$; also *sum_recursive*$(a, m+1, j)$ is computed correctly as the number of terms in the sum is $j - m + 1 \leq n$. Then adding these two values gives $\sum_{k=i}^{m} a_k + \sum_{k=m+1}^{j} a_k = \sum_{k=i}^{j} a_k$ as we wanted. Then by the principle of complete induction, $P(n)$ is true for $n = 0, 1, 2, \ldots$ and so no matter the number of terms in the sum, *sum_recursive*$(a, i, j)$ does indeed correctly compute $\sum_{k=i}^{j} a_k$.

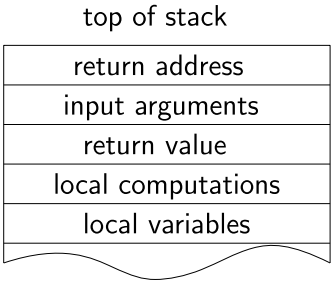



**Fig. 1.1.2** Stack frame structure

#### 1.1.5.2 Implementation of Recursion

Early computer systems did not make allowance for recursion, but that had changed by the 1970s. Some early computer systems did use recursion, such as Lisp in 1958 by John McCarthy, where recursion was not only permitted, it was central to the whole design. In 1960, the programming language Algol was developed which also permitted recursion. By the early 1980s, recursion was a basic part of all computer systems.

The key to implementing recursion is to have a *stack* which is a data structure where data items can be *pushed* onto the top of the stack, or *pulled* (or *popped*) from the top of the stack. Stacks can be used to create a general mechanism for calling a function that makes no distinction between whether the call is to another function or is recursive.

Figure 1.1.2 shows the structure of a *stack frame*.

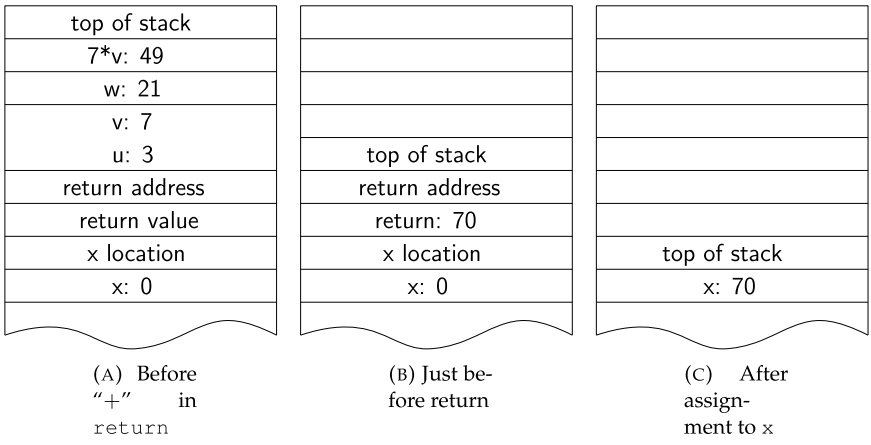
As an example consider the code

```
function g(u, v)
  w = u * v
  return w + 7 * v
end function
  ...
x = g(3, 7)
// return address points near here
  ...
```

Just before the "+" operation on the line with the return, the top of the stack would look like Figure 1.1.3a; just before executing the return statement is shown in Figure 1.1.3b; after assignment of the return value to $x$ is shown in Figure 1.1.3c.

### *1.1.6 Working in Groups: Parallel Computing*

Explicit *parallel computing* is beyond the scope of this book, but suitable references can be found in [247]. Computing with large numbers of threads is also used for




(A) Before "+" in `return`

(B) Just before return

(C) After assignment to `x`

**Fig. 1.1.3** Stack frames

programming *Graphical Processing Units* (GPUs) [87] to do large-scale numerical computations [204, Chap. 11].

Parallel computing can happen at a number of different levels. Parallel computing can happen at the level of individual cores through SIMD instructions that apply the same operation to multiple data items simultaneously, or multiple computational units working simultaneously on different data items. It can happen through having multiple "cores", each of which is a CPU in its own right, with each core working on a different computation. Parallel computing can happen with multiple "chips", each with multiple cores, working on different computations. Or multiple parallel computers can coordinate computations over the Internet.

Parallel computation at the lowest level, using hardware units simultaneously as controlled by the control unit of each core, does not require explicitly parallel instructions from the programmer. However, there are ways of writing code that makes it possible (or makes it difficult) for the compiler and the hardware to identify this parallelism. Consider in Algorithm 3 which shows two ways of computing the inner product $\boldsymbol{x}^T \boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$. The second variant uses *loop unrolling* to make it clear that the summations over $s_1, s_2, s_3$, and $s_4$ in lines 3–8 can be done independently and in parallel.

Parallelism at the level of using separate cores is often called *fine-grained parallelism*. This is often best achieved using either a *thread*-based method such as PThreads or OpenMP in C or C++ [49, 55, 175]. For matrix operations, PLAPACK (1997) also exploits thread-level parallelism [250]. Other languages, such as Python and Julia, have methods of creating and running threads in parallel—whether or not they make use of additional hardware, or use *time sharing* to share the same hardware. Threads are often used for purposes other than improving overall computational speed. For example, one thread in a program may be performing computations, while

**Algorithm 3** Inner product and loop unrolling

```
1  function innerprod(x, y)
2    s ← 0
3    for i = 1, 2, ..., n
4      s ← s + x_i y_i
5    end for
6    return s
7  end function
```

```
1  function innerprod2(x, y)
2    s_1, s_2, s_3, s_4 ← 0
3    for j = 0, 1, 2, ..., ⌊n/4⌋ − 1
4      s_1 ← s_1 + x_{4j+1} y_{4j+1}
5      s_2 ← s_2 + x_{4j+2} y_{4j+2}
6      s_3 ← s_3 + x_{4j+3} y_{4j+3}
7      s_4 ← s_4 + x_{4j+4} y_{4j+4}
8    end for
9    s ← s_1 + s_2 + s_3 + s_4
10   for i = 4⌊n/4⌋ + 1, ..., n
11     s ← s + x_i y_i
12   end for
13   return s
14 end function
```

another thread is handling user interaction, so as to avoid situations where a program appears "dead" while focusing solely on computation.

Multiple cores on a single CPU often have separate level-1 caches while sharing common level-2 and higher level caches. Thus multiple cores ultimately share the same memory. Threads reflect this: while separate threads have separate system stacks, the data and code accessed belong to the same memory system. This is called *shared-memory parallelism*. Thus, there can be conflicts between threads due to separate threads trying to simultaneously write to the same memory location, or one thread might be reading from a location in memory while another thread is writing to that same location. Thus, it is easy to corrupt data in thread-based systems. To prevent this, programmers use a combination of software and hardware facilities to implement *locks* and *semaphores* for controlling access to common memory areas and common variables. Some operations have to be made *atomic*; an atomic operation is one that cannot be interrupted—it cannot be split into parts. Closing and opening locks typically need to be atomic operations. It is common in shared-memory parallelism to lock access to common variable, perform an operation with that variable, and then open (or release) the lock so that other threads can then access that variable. OpenMP takes care of most of these details. The trade-off in using OpenMP is that it is somewhat restrictive in terms of the types of parallelism that can be achieved.

There are generally two to eight cores on most CPUs at time of writing. These numbers will undoubtedly increase over time. On the other hand, Graphical Processing Units (GPUs) typically have hundreds or even thousands of cores on a single unit.

As an example of how shared-memory parallelism can be exploited, consider the recursive summation function in Algorithm 2. This can be turned into an efficient shared-memory parallel function for summation. The parallel sections are marked by `parallel ... end parallel`; each statement in the parallel block of code is meant to run on a separate thread in parallel.

This approach to parallel summation is called *parallel reduction*, and is applicable to many other operations besides addition. Parallel reduction can be applied

**Algorithm 4** Parallel summation

```
1   function sum_parallel(a, i, j)
2     if i > j:        return 0
3     else if i = j: return a_i
4     else
5       m ← ⌊(i + j)/2⌋
6       parallel
7         s_1 ← sum_parallel(a, i, m)
8         s_2 ← sum_parallel(a, m + 1, j)
9       end parallel
10      return s_1 + s_2
11    end if
12  end function
```

to any operation that is *associative*: $(a * b) * c = a * (b * c)$. Parallel reduction can compute $a_1 * a_2 * \cdots * a_n$ in $\lceil \log_2 n \rceil$ parallel "$*$" operations assuming an unlimited number of parallel threads.

Using multiple CPUs means that memory is not shared; this is *distributed memory parallelism*. Data is passed between CPUs by *passing messages* over a communications network. This might be a dedicated network or even the Internet. The best known system for implementing distributed memory parallelism is the *Message Passing Interface (MPI)* [113]. MPI was designed to use with Fortran and C/C++, but most languages (including Java, Julia, Python, and R) have libraries with MPI bindings. Julia comes with modules for distributed memory parallelism.

In distributed memory parallelism, there is a cost of moving data that does not occur in shared-memory parallelism. However, distributed processing removes memory access congestion that often occurs in shared memory parallelism. Separate level-1 cache for cores relieves memory access congestion to some extent for shared-memory parallelism, but to obtain very high speed-ups, distributed memory parallelism is necessary.

To obtain best performance, shared and distributed parallelism need to be combined. MPI-3 has features to do this; otherwise, OpenMP and MPI should be combined.

### *1.1.7 BLAS and LAPACK*

Linear algebra tends to dominate much of numerical analysis. Large problems usually involve large matrices. So doing matrix computations fast has been a focus of numerical analysts for many years. Basic Linear Algebra Subprograms (BLAS) was first published in 1979 [157], extended to BLAS-2 [81] in 1988 and BLAS-3 [79] in 1990. While BLAS concerns itself with operations on vectors, BLAS-2 concerns matrix–vector operations, and BLAS-3 concerns matrix–matrix operations. The level

of the BLAS (indicated by the "-2" or "-3") indicates the depth of nested loops needed for a naive implementation.

Thus, BLAS (or BLAS-1) deals with vector addition, scalar multiplication of vectors, and dot products and lengths of vectors; BLAS-2 deals with matrix–vector products, matrix–transpose–vector products, solving triangular linear systems (see Section 2.1); BLAS-3 deals with matrix–matrix products including matrix–transpose–matrix products, and solving systems of the form $TX = Y$ for $X$ where $T$ is a triangular matrix and $Y$ is a matrix right-hand side.

The LINPACK numerical package [82] developed in the 1970s and early 1980s, targeting mainly supercomputers, used the BLAS-1 library. The follow-up package LAPACK [5] (1992) used all levels of BLAS, with particular focus on using BLAS-3 with block algorithms wherever possible.

To understand the performance benefit of using blocked algorithms, consider three operations from the different levels of BLAS: dot products from BLAS-1, matrix–vector products from BLAS-2, and matrix–matrix products from BLAS-3. Modern processes can perform floating point operations far faster than data can be transferred from main memory. So it is important to re-use data brought in from main memory as much as possible. Table 1.1.2 shows the number of data items to be transferred, the number of floating point operations, and the ratio of the two for $n$-dimensional vectors and $n \times n$ matrices.

As can be seen in Table 1.1.2, the highest ratio of flops to data transfer occurs with BLAS-3 operations. While it can be tempting to think that we can make the value of $n$ large for even better performance with BLAS-3 operations, when the data needed to do the operations overflows the CPU cache, the benefit is lost (see *thrashing* in Section 1.1.2.3). LAPACK makes full use of these operations by using block operations on $b \times b$ matrices, where $b$ is chosen so that the blocks fit in cache memory.

BLAS implementations are available for various computer architectures as well as a reference implementation available from `netlib` [80]. The reference implementation should only be used to check correctness of a BLAS implementation. Vendor-provided BLAS tends to provide optimal performance for their architecture, such as Intel's Mathematics Kernel Library (MKL) [132] or nVIDIA's cuBLAS for running on their GPUs. ATLAS (automatically tuned linear algebra software) [259] is open-source software that generates partly optimized BLAS implementations in C or Fortran by timing various operations to obtain a near-optimal selection of strategies for improving performance. While not as highly optimized as vendor-provided implementations, for new architectures ATLAS quickly provides a BLAS implementation that is hard to beat.

**Exercises.**

(1) Write code for multiplying a pair of $n \times n$ matrices ($C \leftarrow AB$):

```
for i = 1, 2, ..., n
  for j = 1, 2, ..., n
    c_ij ← 0
```

**Table 1.1.2** BLAS level, flops, and data transfer

| BLAS level | Operation | Flops | Data transfer | Ratio |
|---|---|---|---|---|
| BLAS-1 | dot product | $2n$ | $2n$ | 1 |
| BLAS-2 | matrix–vector product | $2n^2$ | $n^2 + 2n$ | $\approx 2$ |
| BLAS-3 | matrix–matrix product | $2n^3$ | $2n^2$ | $n$ |

```
      for k = 1, 2, ..., n
        c_ij ← c_ij + a_ik b_kj
      end for
    end for
  end for
```

Do this in your favorite interpreted language (MATLAB, Python, Ruby, …). Use your code for multiplying a pair of $100 \times 100$ matrices. If your language has a built-in operation for performing matrix multiplication, use this to compute the matrix product of your test matrices. How much faster is the built-in operation?

(2) The matrix operation $C \leftarrow C + AB$ on $n \times n$ matrices can be written as three nested loops:

```
for i = 1, 2, ..., n
  for j = 1, 2, ..., n
    for k = 1, 2, ..., n
      c_ij ← c_ij + a_ik b_kj
    end for
  end for
end for
```

The order of the loops can be changed; in fact, any order of "`for` $i$", "`for` $j$" and "`for` $k$" can be used giving a total of $3! = 6$ possible orderings. In a *compiled* language (such as Fortran, C/C++, Java, or Julia), time the six possible orderings. Which one is faster. Can you explain why?

(3) A rule of thumb for high-performance computing is that when a data item is read in, we should use it as much as possible, rather than re-reading that data item many times and doing a little computing on it each time. In matrix multiplication, this can be achieved by subdividing each matrix into $b \times b$ blocks. For $n \times n$ matrices $A$ and $B$, we can let $m = n/b$ and write

$$AB = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} & \cdots & B_{1m} \\ B_{21} & B_{22} & \cdots & B_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ B_{m1} & B_{m2} & \cdots & B_{mm} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & \cdots & C_{1m} \\ C_{21} & C_{22} & \cdots & C_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ C_{m1} & C_{m2} & \cdots & C_{mm} \end{bmatrix} = C.$$

Then for $i, j = 1, 2, \ldots, n$, $C_{ij} \leftarrow \sum_{k=1}^{m} A_{ik} B_{kj}$. As long as we can keep all of $A_{ik}$, $B_{kj}$, and $C_{ij}$ in cache memory at once, the update $C_{ij} \leftarrow C_{ij} + A_{ik} B_{kj}$ can be done without any memory transfers once $A_{ik}$ and $B_{kj}$ have been loaded into memory. Write out a blocked version of the matrix multiplication algorithm and count the number of memory transfers with the blocked and original matrix multiplication algorithms.

(4) Implement the original and unrolled inner product algorithms in Algorithm 3 as a function in your favorite programming language. Time the function for taking the product of two vectors of $10^6$ entries. Use the simpler version to check the correctness of the unrolled version. Note that there may be differences due to roundoff error. Put the function call inside a loop to perform the inner product $10^3$ times. Is there any difference in the times of the two algorithms? Note that interpreted languages, such as Python and MATLAB, may see little or no difference in timings. This is probably due to the fact that the time savings for the unrolled version is negligible compared to the overhead of interpretation. Also, interpreted languages will typically "box" and "unbox" the raw floating point value, converting it to and from a data structure that contains information about the type of the object as well as the object itself.

(5) The following pseudo-code is designed to provoke a "stack overflow" error:

```
function overflow(n)
  if n = 2^k for some k
    print(n)
  end if
  overflow(n + 1)
end function
```

Implement in your favorite programming language. How does it behave on your computer? How large a value of $n$ is printed out before overflow occurs?

(6) Dynamic memory allocation is memory allocation that occurs at run-time. It is an essential in all modern computational systems. Pick a programming language. How does this language allocate memory or objects? Do you need to explicitly de-allocate memory or objects in your programming language? How is this done?

(7) There are different ways of automatically de-allocating unusable objects (*garbage collection*) in programming systems. In this exercise, we look at two of them. Describe reference counted garbage collection and "mark and sweep" garbage collection. What are their strengths and weaknesses?

(8) *Memory leaks* are a kind of bug that can be hard to find and remove. These arise when memory is allocated for objects that are never de-allocated, and so eventually take up all available memory. Even if a system has garbage collection, this can still occur. Explain how this might happen.

(9) Memory allocation and de-allocation can lead to fragmentation of the memory allocation system over time, so that there may be a great deal of memory available, but no large object can be allocated because the available memory is fragmented

into small pieces. Read about and describe the SmallTalk double indirection scheme that allows SmallTalk to de-fragment the memory allocation system.

## 1.2 Programming Languages

Algorithms are often described using pseudo-code, but to make them work we need to translate these high-level descriptions into a programming language that can be executed. MATLAB$^{TM}$ is a language that was developed specifically for numerical computation. It is a commercial system, although there are publicly available work-a-like systems such as GNU Octave. Python is a general-purpose programming language. The NumPy and SciPy extensions to Python make Python into a suitable platform for numerical computation. Julia is also a general-purpose programming language, but was developed with numerical computation in mind. C, C++, and Java are general-purpose programming languages. C was developed to write the original Unix operating system. C++ was developed as an object-oriented version of C. Java was developed to be a write-once-run-anywhere language for web applications running on computers, tablets, and smartphones.

### *1.2.1 MATLAB$^{TM}$*

Here is an example of MATLAB code:

```
n = 10;
A = zeros(n,n);
for i = 1:n
  for j = 1:n
    A(i,j) = 1/(i+j-1);
  end
end
rhs = 1 ./ (1:n);
sol = A \ rhs'
```

This solves a $10 \times 10$ linear system $A\boldsymbol{x} = \boldsymbol{b}$ where $a_{ij} = 1/(i + j - 1)$ and $b_i = 1/i$. The exact solution is $\boldsymbol{x} = \boldsymbol{e}_1$.

MATLAB is an interpreted language, but matrix operations are done in C. Originally, MATLAB was an interface to LINPACK and EISPACK, which were written in Fortran in the 1970s. Gradually all the code was translated into C.

Originally, MATLAB's only data structure was the matrix, and even strings were represented in terms of matrices. However, as MATLAB developed and its users became more sophisticated, more general heterogeneous data structures were incorporated into MATLAB. The basic mechanism for doing this is the "`struct`":

```
obj = struct('a',37,'b','a string','c',[1 2; 3 4]);
obj.c
```

returns a $2 \times 2$ matrix. Compilers are available for MATLAB, but their use is uncommon. Three and higher dimensional arrays are now available in MATLAB.

As with most "scripting" languages, the type of a variable is determined by its value. There is no way of declaring that a particular variable to have a particular type. The efficiency of MATLAB comes mainly from the fact that underlying matrix operations have been implemented in LAPACK and BLAS since 2000. Trying to re-create matrix operations by means of `for` loops is generally very inefficient. Optimizing MATLAB code then usually involves "vectorizing" operations so that multiple operations can be expressed concisely.

When passing arrays, MATLAB passes them by reference for efficiency. However, MATLAB's programming model is that of pass-by-value. To reconcile these two aspects, MATLAB uses a copy-on-modify rule: if a passed array is modified, a copy is created and the copy is modified. In this way, the entries of the passed array are not changed and the programmer does not have to be concerned about function arguments being modified outside the function.

### *1.2.2 Julia*

Here is the corresponding Julia code:

```
using LinearAlgebra
n = 10;
A = [1/(i+j-1) for i = 1:n, j = 1:n];
rhs = [1/i for i = 1:n];
sol = A \ rhs
```

Julia is a just-in-time-compiled language, meaning that there is no separate compiler. As you enter or load code, it is compiled. This means that some large packages take some time to load as this involves compilation. Large packages can be designated to be pre-compiled.

Julia has a sophisticated type system and uses these types to determine which function of the correct name to use. This enables Julia users to override the arithmetic operations (+, -, *, /) for user-defined types. For example, we could create a specialized three-dimensional vector type:

```
struct Vec3
  x,y,z :: Float64 # double precision
end
```

and the usual vector operations, as well as the cross product

```
function +(a::Vec3, b::Vec3)
  return Vec3(a.x+b.x,a.y+b.y,a.z+b.z)
end
  ...
function cross(a::Vec3, b::Vec3)
  return Vec3(a.y*b.z-a.z*b.y,
              a.z*b.x-a.x*b.z,
              a.x*b.y-a.y*b.x)
end
```

When there is a conflict between two different versions of a function whose input types match the function call, Julia will pick the more specific one where this can be decided. In this way, you can write one version for a specific case that is highly optimized for that case, and a more general version that is slower.

Julia's numerical matrix computations use BLAS and LAPACK (see Section 1.1.7) wherever possible for efficiency.

Unlike MATLAB, since Julia is compiled as needed, `for` and `while` loops are reasonably efficient. Julia passes arrays by reference, but Julia has a convention that if a function modifies its arguments, the name must have an exclamation point ("!") at the end. In this way, programmers are alerted to the possibility that a passed array or other object may be modified.

### *1.2.3 Python*

Here is the corresponding Python code using NumPy for handling matrices and linear algebra:

```
import numpy as np
n = 10
rhs = np.array([1/(i+1) for i in range(n)])
A = np.array([[1/(i+j+1) for i in range(n)] for j in range(n)])
sol = np.linalg.solve(A,rhs)
print(sol)
```

Python is usually used as an interpreted language, although there are compilers for Python. NumPy performs numerical operations by calling C routines for the NumPy array operations.

### *1.2.4 C/C++ and Java*

C, C++, and Java are compiled languages, with much of the syntax of these languages the same. Java was designed to be run on any platform, and so the output of a Java compiler is not machine code, but an intermediate language (Java bytecode), which

is interpreted by a Java Virtual Machine (JVM). Java bytecode can be compiled, and some JVMs do compile parts of a running Java program to machine code for greater performance. There are a number of packages for numerical computations in each of the languages with some free and some commercial. Free libraries include the GNU Scientific Library for C, while Boost has numerical components for C++, and Colt is a general numerical library for Java. Commercial libraries such as ISML and NAG have C bindings although are mostly Fortran underneath. However, there is no standard "matrix" data structure for these languages. Bindings to Fortran routines for BLAS are available in most libraries.

### *1.2.5 Fortran*

Fortran is arguably the oldest generally available compiled programming language, with the original Fortran I becoming available on certain IBM 704 computers in 1957 [14]. Fortran has undergone a number of transformations, but is still in use, and there is much code that is written in Fortran. When it was developed, Fortran had to show that using a compiler could result in executable code that was about as efficient as writing assembly language or machine code. So a focus of Fortran was efficiency. Consequently, many aspects of modern programming languages were not present. For example, recursion was initially not permitted in Fortran. Other modern programming language features were slowly added: the Fortran 66 standard, approved in 1966, is widely regarded as the starting point; Fortran 77, approved in 1978, allowed `if ... end if` blocks and `do ... end do` loops instead of requiring numeric line labels to designate the end of an if statement or a loop; Fortran 90, approved 1991, allowed free format rather than column oriented input, recursive functions, modules for combining related functions and variables, pointers (also known as references), dynamic memory allocation, interfaces, array slicing, and operator overloading. In short, Fortran 90 caught up with many of the requirements of a modern programming language. Further extensions were made in subsequent standards: Fortran 95 incorporated some minor extensions, such as `forall` for efficient vectorization of array operations; Fortran 2003 with object-oriented programming support, procedure (that is, function) pointers, and enhanced module support allowing separation of interfaces and implementation; Fortran 2008 provides some additional support for parallel computing; finally, Fortran 2018 provides some enhancements regarding interoperability with C, and improvements in the parallel computing features.

```
real, allocatable, dimension(:,:) :: a, b, c
! missing initialization code to
! allocate a, b, c and initialize a and b
do j=1,n
  do i=1,n
    tmp = 0.0
    do k=1,n
```

```
        tmp = tmp + a(i,k) * b(k,j)
      enddo
      c(i,j) = tmp
    enddo
  enddo
```

Standard components of modern scientific computing, such as BLAS and LAPACK, are written in Fortran. In fact, LAPACK was written in Fortran 77 until 2008, when it was translated into Fortran 90. While Fortran is regarded as "old fashioned" (even back in 1968), it is still an important language, especially for scientific and numerical computing. Modern versions of Fortran provide many of the features expected of modern programming languages, although many of these are "retrofitted", rather than part of the initial design.

**Exercises.**

(1) Pick your favorite programming language. Is it interpreted? Is it compiled? What is (or are) the data type(s) for floating point numbers? Is there a built-in data type for vectors or matrices? Can the vectors (or matrices) be added? Can they be multiplied?
(2) What facilities are included in your favorite programming language for parallel programming? Are there libraries that provide these facilities? Is it possible to perform both shared memory and distributed memory computing in your language? With these libraries?
(3) Is your favorite programming language garbage collected? That is, is there automatic de-allocation of unusable objects? What are the advantages and disadvantages of garbage collection?
(4) The following function applies the function $f$ to each item of a vector of items. Implement it in your favorite programming language.

```
function map(f, x)
  y ← new array of same size as x
  for i an index of x
    y_i ← f(x_i)
  end for
end function
```

Now implement it in your least favorite, or a previously unknown, programming language.
(5) Macros are pieces of code that *transform other pieces of code* before compilation or execution. Does your favorite programming language have macros? What kinds of transformations can the macros in that language perform? If your favorite programming language does not have macros, or you discover another language that does have macros (such as Lisp or Julia), describe the macro facilities of that language.

(6) The C programming language was developed for implementing the Unix operating system. This led to many design decisions by the creators of the language, such as pointer arithmetic, and lack of array bounds. Explain why these decisions may have been necessary for their purpose at that time, and if you think those decisions help or hinder the creation of mathematical software now.
(7) Some languages are stack languages, such as Forth, Joy, and PostScript. Arguments for a "function" in a stack language do not have names, but the $k$th input is the $k$th item from the top of the stack. Download one of these languages and implement the "map" function of Exercise 4 in that language.

## 1.3 Floating Point Arithmetic

Real numbers cannot be stored in a computer, because storing a real number, like

$$\pi = 3.1415926\ldots,$$

would take an infinite amount of memory. So computers and calculators store just a finite number of digits or bits after the decimal point. Computer scientists and mathematicians have worked since the dawn of electronic computers to find efficient ways to accurately represent, or approximate, real numbers. The common idea is to use *floating point arithmetic*. A floating point number consists of three parts: a sign ($\pm$), a mantissa or significand (where most of the digits or bits of interest are), and an exponent. Floating point numbers have a *base*, which is typically 2 or 10 but was sometimes 16 or something different. For floating point numbers in base $b$ we represent a number $x$ as

$$x = \pm(d_0.d_1d_2\cdots d_m)_b \times b^e,$$

where $e$ is the exponent and $(d_0.d_1d_2\ldots d_m)_b = d_0 + d_1/b + d_2/b^2 + \cdots + d_m/b^m$ is the mantissa.

Before 1985, different manufacturers of computers and computer hardware used different formats for storing floating point numbers, making it difficult to use programs written for one computer system on another. It also made it difficult to reason about how software operating on floating point numbers should behave. In 1985, the Institute for Electronic and Electrical Engineering standard IEEE 754 for floating point arithmetic was adopted and has become the main standard used for floating point arithmetic.

### *1.3.1 The IEEE Standards*

In 1985, the IEEE Standard for Floating-Point Arithmetic (IEEE 754) was published by the Institute for Electrical and Electronic Engineers (IEEE). It represented the culmination of collaboration between industry, academics, and engineers [128]. Since

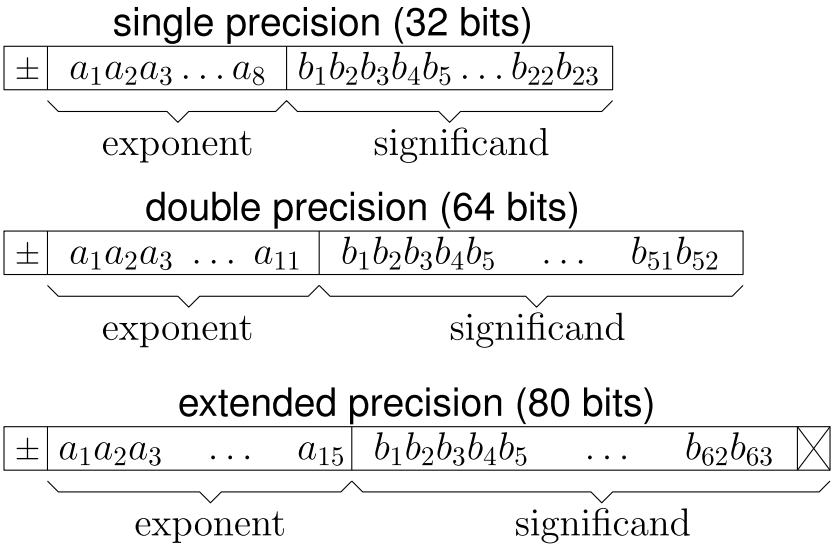



**Fig. 1.3.1** IEEE 754: Floating point standards

then, most floating point computations have been carried out through the IEEE standards. Before the IEEE 754 standards, there were many different and incompatible floating point systems. Trying to write software to work under the different formats and systems was challenging. The IEEE standards completely changed this, but not by aiming for the lowest common denominator. The IEEE standards set a new bar for how floating point systems should work, with support for gradual underflow, correctly rounded arithmetic, rounding modes, and with values for "infinity" and "Not-a-Number". More extensive discussions on floating point arithmetic can be found in [102, 194].

The IEEE standards were actually three different standards for single precision, double precision, and extended precision floating point arithmetic. The three standards use binary representation of numbers (base 2), but use different numbers of bits for the different components as well as different numbers of bits for the entire number. The important components for a floating point number are: the *sign bit*, the *exponent*, and the *significand*. The sizes of these components are shown in Figure 5.4.2. In what follows, we let $M$ be the number of bits in the exponent field, and $N$ the number of bits in the significand (Figure 1.3.1).

The leading bit is the sign bit which is zero for "+" and one for "−". The exponent field does not directly represent the exponent. Instead, the exponent is $e = (a_1a_2\ldots a_M)_2 - E_0$ where $E_0$ is chosen to give a balance between positive and negative exponents: $E_0 = 2^{M-1} - 1$. This gives values of $e$ ranging from $-E_0 = 1 - 2^{M-1}$ to $(2^M - 1) - E_0 = +2^{M-1} = E_0 + 1$. Provided $-E_0 < e < E_0 + 1$, the number represented is

$$\pm(1.b_1b_2\ldots b_N)_2 \times 2^e.$$

This is a *normalized* floating point number. *Unnormalized* floating point numbers have a lead digit (or bit) that is zero. An advantage of working in binary is that a normalized floating point number must start with a "1". Unnormalized floating point numbers can be normalized by shifting the bits in $b_1 b_2 \ldots b_N$ left and adjusting the exponent down to compensate, until the leading bit is not zero (and must therefore be 1). The only exception to being able to normalize an unnormalized floating point number is if $b_1 b_2 \ldots b_N = 00 \ldots 0$, that is, the number is exactly zero, or the exponent field hits its minimum value.

If $e = -E_0$, in which case $(a_1 a_2 \ldots a_M)_2 = (00 \ldots 0)_2 = 0$, the number is a *denormalized* floating point number

$$\pm(0.b_1 b_2 \ldots b_N)_2 \times 2^{-E_0+1} = \pm(b_1.b_2 \ldots b_N)_2 \times 2^{-E_0}.$$

If $e = +E_0 - 1$, in which case $(a_1 a_2 \ldots a_M)_2 = (11 \ldots 1)_2 = 2^M - 1$, the quantity represented is either infinity (denoted "Inf") or a NaN ("not-a-number"). Infinity can be signed, depending on the sign bit, so both $\pm$Inf are possible distinct values. The quantity represented is then $\pm$Inf if $b_1 b_2 \ldots b_N = 00 \ldots 0$; otherwise, the quantity represented is a NaN, which is best understood as "undefined". Any arithmetic operation or function evaluation with a NaN results in NaN. Any comparison of a NaN with any floating point number gives the value "false". That is, if $x$ is a NaN then even the equality test "$x = x$" will return false. This gives a quick way of identifying if a quantity is a NaN.

There are many ways of creating "Inf" and "NaN". For creating "Inf" we can use division by zero (such as $1.0/0.0$), very large values (such as $\exp(\mathit{BigNumber})$ or $\mathit{VeryBigNumber} \times \mathit{AnExtremelyBigNumber}$), or certain functions applied to $\pm$Inf ($\exp(\text{Inf})$, $\text{Inf}^2$, $\log(\text{Inf})$, or $\sqrt{\text{Inf}}$). NaN can be created by zero on zero ($0.0/0.0$), certain function values ($\sin(\text{Inf})$ or $\cot(0)$), or operations on "infinity" ($\text{Inf} - \text{Inf}$, $\text{Inf}/\text{Inf}$, or $1^{\text{Inf}}$).

Any computation that gives a floating point result that is too small in magnitude to be a normalized, is called *underflow*. If the result is a denormalized number that is not actually zero, then it is called *gradual underflow*. Any computation that gives a floating point result that is too large to be normalized (so that the result is $\pm$Inf or NaN), we say that there has been *overflow*. There is nothing gradual about overflow.

NaNs are to be avoided wherever possible. Once created, they tend to propagate. Anything it touches becomes NaN. Even $0 \times \text{NaN}$ is NaN. In the time before IEEE arithmetic, what happened instead of generating this symbolically undefined quantity is that a program in that situation would crash: the program would terminate, hopefully generating a useful error message about what went wrong and where it happened. Now, a program will happily continue past the point at which a NaN is generated, probably using that value multiple times resulting in many NaNs in your results. You can expect to then have a printout of results consisting mostly, if not entirely, of NaNs. You will look at this useless output and ask "What went wrong?" and "Where did it go wrong?" The NaNs will not tell you, unless you put tests in your code to declare an emergency as soon as a NaN is found.

NaNs also have special properties regarding comparisons: any comparison with a NaN is *false*. So, for example, $\text{NaN}^2 \geq 0$ is always false, as is $\text{NaN} \leq \text{Inf}$. Most importantly $\text{NaN} = \text{NaN}$ is also always false. Thus, a variable $x$ can be tested to be a NaN by checking if $x = x$; if $x$ is not a NaN then the result is *true*, but if $x$ is a NaN, then the result is *false*. In this way, NaNs can be identified.

Apart from debugging purposes, testing explicitly for NaNs in programs is often not needed, except perhaps in code for high-reliability applications. It is even better to design your code to avoid NaNs. This may not always be possible if your inputs include user-defined functions. While it might be hoped that your clients would write functions that never generate a NaN, it can happen, so you should be prepared to deal with this eventuality.

### *1.3.2 Correctly Rounded Arithmetic*

The most important property of IEEE arithmetic is that it is *correctly rounded*. That is, provided $x$ and $y$ are floating point numbers and $\circ = +, -, \times, /$, the computed value of $x \circ y$ is *the nearest floating point number to the exact value* of $x \circ y$. This means that there are some very useful models of floating point arithmetic that apply to IEEE arithmetic, but not necessarily to other implementations of floating point arithmetic. To develop these models we need a notation that distinguishes between the value of an expression using real numbers, with infinite precision, and the value obtained by using a given system of floating point arithmetic. For a given expression *expr*, we use *expr* to represent the exact value, and $fl(expr)$ to represent the value of *expr* as computed using floating point arithmetic.

IEEE arithmetic achieves correctly rounded arithmetic for the usual floating point operations by using *guard digits*; these are extra digits (bits, actually) to achieve some additional accuracy during the computation, *but before rounding*, so that the final rounded result is correct. IEEE arithmetic, in fact, has several rounding modes, of which *round-to-nearest* is the default option just described. Other rounding options include *round-up*, *round-down*, and *round-toward-zero*.

Using the default round-to-nearest mode enables us to give a formal model for IEEE arithmetic that is not a complete model in that it does not completely specify the result of a given floating point operation. Many attempts have been made to create an algebra for floating point operations. However, the resulting algebra must be a non-associative: that is, there are $a$, $b$, and $c$ where $(a + b) + c \neq a + (b + c)$ where "+" is taken to be floating point addition. To see this, suppose that in a given floating point system that is correctly rounded, there must be a positive floating point number $\delta > 0$ where the computed value $fl(1 + \delta) = 1$; choose $\delta > 0$ to smaller than half the distance from one to the next largest floating point number. Then

$$fl(fl(\delta + 1) + (-1)) = fl(1 + (-1)) = 0 \qquad \text{while}$$
$$fl(\delta + fl(1 + (-1))) = fl(\delta + 0) = \delta \neq 0.$$

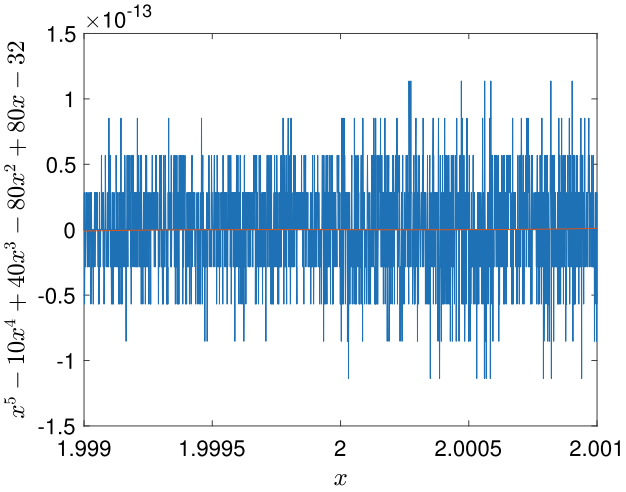



**Fig. 1.3.2** Plot of $g(x) = x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 32$

As another example, to illustrate the difficulty in giving an exact model, consider the function $g(x) = (x-2)^5 = x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 32$. A plot of computed values of the expanded expression at steps of $10^{-6}$ computed using MATLAB$^{TM}$ using IEEE double precision is shown in Figure 1.3.2. Attempting to predict exactly what rounding errors will be incurred is extremely difficult and makes analysis difficult.

#### 1.3.2.1 A Formal Model of Floating Point Arithmetic

Instead of trying to model the exact behavior, we just try to determine bounds for the results of a floating point operation. If $z$ is a positive real number in the range of normalized floating point numbers, from $2^{-E_0+1}$ to $2^{+E_0}$, then the closest floating point number $fl(z)$ to $z$ satisfies

$$|z - fl(z)| \leq \mathbf{u}\,|z|$$

for a positive number $\mathbf{u}$ called *unit roundoff*. Unit roundoff can be determined to be $2^{-N}$ where $N$ is the number of bits in the significand. Values of unit roundoff for the three IEEE standards, along with the approximate ranges of the three standards, are shown in Table 1.3.1.

Note that *absolute error* in a computed quantity $\widehat{z}$ with exact value $z$ is just the difference $|z - \widehat{z}|$, the *relative error* is $|z - \widehat{z}|\,/\,|z|$. Then if $z$ is in the range of normalized floating numbers, then the absolute error in $fl(z)$ is $\leq \mathbf{u}\,|z|$, while the relative error is $\leq \mathbf{u}$.

**Table 1.3.1** Parameters of floating point arithmetic

| | | denormalized | normalized | |
|---|---|---|---|---|
| | unit roundoff ($\mathbf{u}$) | smallest number ($\mathbf{f}_{\min}$) | smallest | largest |
| single | $2^{-23} \approx 1.2 \times 10^{-7}$ | $2^{-148} \approx 2.8 \times 10^{-45}$ | $2^{-126} \approx 1.2 \times 10^{38}$ | $\approx 2^{128} \approx 3.4 \times 10^{38}$ |
| double | $2^{-52} \approx 2.2 \times 10^{-16}$ | $2^{-1072} \approx 2.0 \times 10^{-323}$ | $2^{-1022} \approx 2.2 \times 10^{-308}$ | $\approx 2^{1024} \approx 1.8 \times 10^{308}$ |
| extended | $2^{-63} \approx 1.1 \times 10^{-19}$ | $2^{-16\,447} \approx 9.1 \times 10^{4\,952}$ | $2^{-16\,382} \approx 3.4 \times 10^{-4\,932}$ | $\approx 2^{32\,768} \approx 1.4 \times 10^{9\,864}$ |

This leads to the following model of floating point arithmetic: for floating point numbers $x$ and $y$ and operations $* = +,\ -,\ \times,\ /$,

$$fl(x * y) = (x * y)(1 + \epsilon) \qquad \text{for some } |\epsilon| \le \mathbf{u}, \tag{1.3.1}$$

*provided there is no underflow (including gradual underflow) or overflow*. Again, $\mathbf{u}$ is unit roundoff. To expand this to allow for underflow, including gradual underflow, we need an additional parameter $\mathbf{f}_{\min}$, the smallest positive floating point number:

$$fl(x * y) = (x * y)(1 + \epsilon) + \eta \qquad \text{for some } |\epsilon| \le \mathbf{u},\ |\eta| \le \mathbf{f}_{\min}. \tag{1.3.2}$$

More general functions can be correctly rounded, although this is much harder to achieve. One reason is that to determine the correct rounding of a function value $f(x)$ may require knowing the value of $f(x)$ to much higher precision than the floating point arithmetic provides if $f(x)$ is close to the midpoint between two adjacent floating point numbers. Providing a formal model for function evaluation is more complex, and the error behavior of a function can depend very much on how the function is implemented. We do expect that built-in functions (such as exp, log, square roots, sin, cos, tan) are well implemented and have an error behavior as good as can be expected. A suitable model for a well-implemented function is

$$fl(f(x)) = f((1 + \epsilon_1)x)\,(1 + \epsilon_2) \qquad \text{for some } |\epsilon_1|, |\epsilon_2| \le \mathbf{u} \tag{1.3.3}$$

*provided there is no underflow (including gradual underflow) or overflow*. Incorporating underflow requires the $\mathbf{f}_{\min}$ parameter:

$$fl(f(x)) = f((1 + \epsilon_1)x + \eta_1)\,(1 + \epsilon_2) + \eta_2 \qquad \text{for some } |\epsilon_1|, |\epsilon_2| \le \mathbf{u},\ |\eta_1|, |\eta_2| \le \mathbf{f}_{\min}. \tag{1.3.4}$$

A reason that we cannot use $fl(f(x)) = f(x)(1 + \epsilon)$ with $|\epsilon| \le \mathbf{u}$ alone is that if $f(x) \approx 0$ and $x \approx 1$ there can still be an error of size $\mathbf{u}$ from preliminary operations on the input $x$. Consider, for example, $f(x) = \sin(\pi x)$ implemented as `sin(pi*x)` in your favorite programming language. (In Java, use `sin(Math.pi*x)`.) Using

the built-in `sin` function and the built-in value of $\pi$, this should be considered "well implemented". Yet, $f(1) = \sin \pi = 0$ while its computed value in MATLAB is $fl(f(1)) \approx 1.22 \times 10^{-16}$ which is not $f(1)\,(1+\epsilon)$ for *any* $\epsilon$. Of course, the numerical value of $\pi$ used is $fl(\pi) \neq \pi$ and the difference $|fl(\pi) - \pi| \leq \mathbf{u}\,\pi \approx 6.91 \times 10^{-16}$. This explains the error in $f(1)$. We can, however, represent $fl(f(1)) = f(1(1+\epsilon_1))\,(1+\epsilon_2)$ for some $|\epsilon_1|, |\epsilon_2| \leq \mathbf{u}$. In this case, $\epsilon_1$ applied to the input is more important than $\epsilon_2$ applied to the output.

In the next section (Section 1.4), we will apply this formal model to understand how things sometimes go wrong.

For some computations, floating point arithmetic is exact: adding zero, and multiplying by a power of two (provided neither overflow nor gradual underflow occur). Multiplying by a power of two generally only results in a change in the exponent. Addition, subtraction, and multiplication of modest sized integers are also exact: as long as the number of non-zero bits of the inputs and the output in binary is less than the number of bits of the significand, the results are computed exactly.

### *1.3.3 Future of Floating Point Arithmetic*

While IEEE floating point arithmetic is the mainstay of modern numerical computation, there are a number of alternatives going in different directions:

- Greater accuracy: variable-size arithmetic, quad precision, software defined precision, and "double double" arithmetic.
- Greater speed: often lower accuracy but with fewer bits and therefore faster to move the data.
- Greater reliability: interval arithmetic, which gives *guaranteed* bounds on the results.

There are always new proposals coming forward, so there is little hope of any description remaining comprehensive. However, there are some common threads to the alternatives to floating point arithmetic. We will look at the different themes in turn.

#### 1.3.3.1 Greater Accuracy

There are many ways of gaining greater accuracy. Fortran has long had its own `quad precision` type where `double precision` was not sufficient. C and C++ have a `long double` type that is often, but not always, implemented by the IEEE extended precision standard. The IEEE754–2008 revision of the IEEE standards allows for a 128-bit quad precision arithmetic. However, this has not been implemented in commonly available hardware, so software implementation must be used. This makes the quad precision arithmetic very slow in comparison with double or extended precision. There are also "double double" and "quad-double" packages that leverage the hardware advantage of using a pair (or four) of IEEE

double precision numbers to create a virtual quad precision system with 106 bits of mantissa (105 bits of significand) [124].

There are pure software systems for arbitrary precision arithmetic, including the GNU Scientific Library [107], as well as "BigFloat" or "BigNum" systems in Java, Julia, Python, and other languages. As with other software implementations of floating point arithmetic, these are relatively slow. Used strategically, though, they can be very useful.

There are more adventurous proposals that aim for efficient and hardware-implementable floating point systems that try to make better use of the bits that are available. The IEEE standards have a fixed exponent size. This means that quantities with values closer to one have the same number of significand bits as much larger or much smaller quantities. Since most computation uses quantities that are closer to one, it is natural to try to allow the number of exponent bits to vary so that these quantities have smaller exponent fields and larger significand fields. Variable exponent size is a challenge for hardware implementation and harder to analyze as there is no longer a single "unit roundoff" to estimate errors. Various "flexible exponent" or "flexible range" forms of floating point arithmetic have been proposed, such as Clenshaw and Olver's *level-index arithmetic* (see [52, 53]) which uses an iterated logarithm scheme to represent numbers over a very wide range. A more recent proposal is Gustafson's *unum* idea [115], which is a simpler "flexible exponent" idea than level-index arithmetic. For most high-performance computing needs, there has to be a fixed number of bits, so that individual entries in an array can be accessed at will. Whatever the number of bits, there is a finite range; overflow and underflow are still possible. The higher complexity of these systems mean that the basic model of "correctly rounded" floating point arithmetic has to be replaced by a more sophisticated, but less understandable, model of floating point error. In short: each system has its limits.

#### 1.3.3.2 Greater Speed

There are developments in the opposite direction: lower precision for greater speed. Applications in signal processing, graphics, and machine learning have amplified the desire for greater speed. These applications typically have lower accuracy requirements than most scientific computation tasks. Uncompressed graphic and audio data streams typically have 1 byte per audio sample and 1 byte per color stream per pixel. Machine learning applications often use low-precision computations and stochastic algorithms where high precision is not useful.

This has led to the use of half precision (16 bit) floating point numbers. These can be transferred roughly twice as fast as single precision (32 bit) and four times as fast as double precision (64 bit) floating point numbers. SIMD (single instruction multiple data) units for higher throughput can process four half precision numbers for each double precision number processed.

A further alternative to half precision floating point arithmetic is to use fixed-point arithmetic. Fixed-point arithmetic is certainly not a new idea; floating point

arithmetic was first developed for computing to overcome the limitations of fixed-point arithmetic in the 1950s. Beyond this, digital signal processing has long used fixed point arithmetic, usually implemented in hardware to achieve sufficient speed.

Half precision floating point arithmetic can be analyzed in the same way as higher precision floating point systems. Fixed-point arithmetic can be analyzed in similar ways, often with $\mathbf{u} = 0$ and $\mathbf{f}_{\min}$ set to the smallest positive fixed-point number. Appropriate methods of analysis of fixed-point arithmetic depend on the system.

#### 1.3.3.3 Greater Reliability

The best way to greater reliability for floating point arithmetic is *interval arithmetic* [140, 180]. Instead of having a single floating point value, we use intervals representing a range of possible values. Then we can apply arithmetic operations to intervals

$$
\begin{aligned}
[a,b] + [c,d] &= \{\, x + y \mid x \in [a,b],\ y \in [c,d] \,\} = [a + c,\ b + d],\\
[a,b] - [c,d] &= \{\, x - y \mid x \in [a,b],\ y \in [c,d] \,\} = [a - d,\ b - c],\\
[a,b] \times [c,d] &= \{\, x \cdot y \mid x \in [a,b],\ y \in [c,d] \,\}\\
&= [\min(ac, ad, bc, bd),\ \max(ac, ad, bc, bd)],\\
[a,b] \,/\, [c,d] &= [a,b] \cdot \frac{1}{[c,d]} \qquad \text{with}\\
\frac{1}{[c,d]} &= \begin{cases} [\frac{1}{d}, \frac{1}{c}], & \text{if } 0 \notin [c,d],\\ [\frac{1}{d}, \infty), & \text{if } c = 0,\\ (-\infty, \frac{1}{c}], & \text{if } d = 0,\\ (-\infty, \frac{1}{c}] \cup [\frac{1}{d}, \infty), & \text{otherwise.} \end{cases}
\end{aligned}
$$

We can guarantee inclusion if we use the round-down rounding mode for the lower bound and round-up for the upper bound, and $\pm$Inf where appropriate. This means that we can guarantee the true value lies in the interval computed.

The computed interval may be much wider than necessary, in which case the result of standard floating point computation should be used. This is the basic idea of the language triplex Algol 60 [7].

Functions can be applied to intervals: for increasing functions such as exp, log, $\sqrt{\cdot}$, we have $f([a,b]) = [f(a),\ f(b)]$. Again, the rounding mode should be set appropriately according to whether the upper or lower bound is being computed. Functions that are a mixture of increasing and decreasing such as $x \mapsto x^2$ and trigonometric functions can also be computed accurately for intervals. Interval computations can give results that are well beyond the actual bounds of function values. Consider, for example, computing $\cosh x = \frac{1}{2}(e^x + e^{-x})$ with the interval $[-2,\ +2]$:

$$\cosh([-2,\ +2]) \subset \frac{1}{2}(\exp([-2, +2]) + \exp(-[-2, +2]))$$
$$= \frac{1}{2}([e^{-2},\ e^{+2}] + [e^{-2},\ e^{+2}]) = [e^{-2},\ e^{+2}] \quad \text{whereas}$$
$$\cosh([-2,\ +2]) = [1, \cosh(2)] = [1,\ \frac{1}{2}(e^{+2} + e^{-2})].$$

Interval arithmetic can be a very powerful tool. However, it is best used selectively.

**Exercises.**

(1) Compute machine epsilon for your computer and programming language:

```
x ← 1
while 1 + x > 1
   x ← x/2
end while
x ← 2x
```

What is machine epsilon for your system?

(2) Estimate the range of your floating point system:

```
x ← 1;  y ← 2x
while y − y = 0  // fails if y = Inf
   x ← y;  y ← 2x
end while
print x
```

Report the value of $x$ returned. Check that the computed value of $2x$ is Inf. Is $1.99 \times x$ computed to be Inf as well?

(3) The Taylor series for $e^x = 1 + x + x^2/2! + x^3/3! + \cdots + x^n/n! + \cdots$. We approximate $e^x \approx 1 + x + x^2/2! + x^3/3! + \cdots + x^n/n!$ for some fixed $n$:

```
y ← 1;  term ← x
for k = 1, 2, ..., n + 1
   y ← y + term;  term ← term × x/(k + 1)
end for
```

If $n = 20$ the remainder term of $e^x$ for $|x| \leq 1$ is less than unit roundoff for double precision. Using this approximation for $x = \pm 1$ compute an estimate for $e^{+1} \times e^{-1}$. If this was done in exact arithmetic you would get exactly one. What is the difference between the computed value of $e^{+1} \times e^{-1}$ and one with this method?

(4) Modify the code in the previous question to sum the terms *in reverse order*, that is, from highest order terms to lowest order terms. Repeat the test of finding the computed value of $e^{+1} \times e^{-1} - 1$ using this method. Which has smaller error?

(5) In interval arithmetic, a number $u$ is represented by an interval $[\underline{u}, \overline{u}]$ that contains the exact value of the number. A vector of two intervals is a rectangle with sides

parallel to the $x$- and $y$-axes. Show that the operation applied to a vector of rectangles $[u, v]^T$ of the same width

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} +1 & -1 \\ +1 & +1 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} u - v \\ u + v \end{bmatrix}$$

results in a rectangle $[x, y]^T$ that has width $\sqrt{2}$ times the width of $[u, v]^T$.

(6) Compare three ways of summing a large number of roughly equal positive quantities: $\sum_{i=1}^{N} a_i$: (1) sum from the beginning; (2) sum from the end; (3) use the recursive algorithm Algorithm 2. Give bounds on the error in each of these summation algorithms of the form $\sum_{k=1}^{N} a_k \, \eta_k$ where $\eta_k$ are numbers independent of the $a_k$. Which algorithm minimizes your bound on $\max_k |\eta_k|$. [**Hint:** Note that $(1 + \alpha/(1 - \alpha))(1 + \beta/(1 - \beta)) \leq 1 + (\alpha + \beta)/(1 - (\alpha + \beta))$.]

(7) Overflow and underflow can occur for surprisingly modest inputs to the exponential function. What are the smallest values of $x$ where $e^x$ results in either overflow or underflow in IEEE single precision, IEEE double precision, and IEEE extended precision?

(8) Interval arithmetic can be used to do more than simply rigorously control roundoff error. The interval version of Newton's method for solving $f(x) = 0$ is

$$x_{n+1} = x_n \cap \left( x_n - f(x_n)/f'(x_n) \right). \tag{1.3.5}$$

Note that division of intervals $[a, b]/[c, d]$ is a pair of intervals if $c < 0 < d$. Implement this method, noting that at each iteration you will need to keep a *union of intervals* rather than a single interval.

(9) The *Table Maker's dilemma* is the problem that determining correctly rounded values of a transcendental function may require evaluation of that function to far higher precision. This issue is discussed in [159], where quad precision is used to resolve these issues, at least for double precision tables. Read this article and explain how these issues are resolved in [159].

## 1.4 When Things Go Wrong

### *1.4.1 Underflow and Overflow*

Overflow is an all-or-nothing phenomenon. Underflow can be gradual or immediate. In diagnosing these problems, we look for results that are Inf, NaN, or zero.

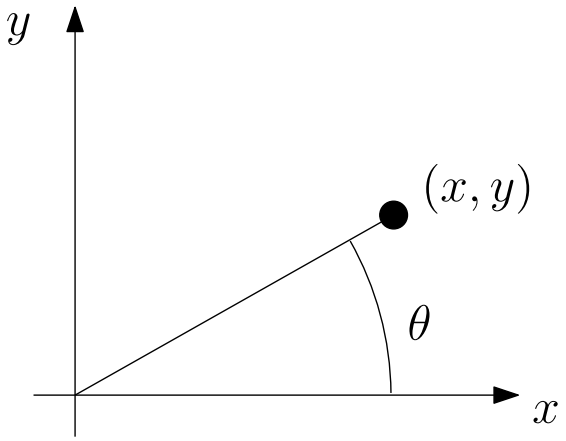



**Fig. 1.4.1** Computation of $\cos\theta$ and $\sin\theta$ given $(x, y)$ coordinates

#### 1.4.1.1 A Trigonometric Computation

Consider, for example, computing $\cos\theta$ and $\sin\theta$ from $(x, y)$ coordinates of a point, as illustrated in Figure 1.4.1. The formulas we use are

$$\cos\theta = \frac{x}{\sqrt{x^2+y^2}}, \tag{1.4.1}$$

$$\sin\theta = \frac{y}{\sqrt{x^2+y^2}}. \tag{1.4.2}$$

In MATLAB, using double precision for $x = 10^{200}$ and $y = 10^{200}$ the computed values of $\cos\theta$ and $\sin\theta$ from (1.4.1, 1.4.2) are both zero. Why? For $\cos\theta = x/\sqrt{x^2+y^2}$, we can get the computed value of $\sqrt{x^2+y^2} = \text{Inf}$. Note that if we use $x = 10^{100}$ and $y = 10^{100}$ then $\sqrt{x^2+y^2}$ is finite and $x/\sqrt{x^2+y^2}$ is computed to differ from $1/\sqrt{2}$ by about $1.1 \times 10^{-16}$. So there is a threshold in the size of $x$ and $y$ where this bad behavior occurs. This indicates overflow.

We can see overflow already in the computation of $x^2$ which evaluates to $fl(x^2) = \text{Inf}$. To see why, $x = 10^{200}$ so $x^2 = (10^{200})^2 = 10^{400}$. However, the largest number representable by double precision arithmetic is $\approx 1.8 \times 10^{308}$. Clearly $10^{400}$ is too big, so the computation overflows and gives the value Inf. Similarly $fl(y^2) = \text{Inf}$. This gives $fl(x^2+y^2) = \text{Inf} + \text{Inf} = \text{Inf}$. Furthermore, $\sqrt{\text{Inf}} = \text{Inf}$ so $fl(\sqrt{x^2+y^2}) = \text{Inf}$. So the final computation is $fl(x/\sqrt{x^2+y^2}) = fl(10^{200}/\text{Inf}) = 0$ as $10^{200}$, while large, is insignificant next to Inf. The same computations and results are obtained for computing $\sin\theta$.

If we try the computation with $x = 10^{-200}$ and $y = 10^{-200}$ then we get the reverse issue: both $\cos\theta = x/\sqrt{x^2+y^2}$ and $\sin\theta = y/\sqrt{x^2+y^2}$ are evaluated to Inf. Again, using $x = 10^{-100}$ and $y = 10^{-100}$ the computed result differs from the correct value $1/\sqrt{2}$ by about $1.1 \times 10^{-16}$. The problem with $x = 10^{-200}$ and

$y = 10^{-200}$ is underflow. We can see this in computing $x^2$ for $x = 10^{-200}$: this gives zero. To see why we see that for double precision the smallest positive floating point number is $\approx 2.0 \times 10^{-323}$ while $x^2 = (10^{-200})^2 = 10^{-400}$ is much smaller. So the computed value must be zero. Then $fl(x^2 + y^2) = fl(x^2) + fl(y^2) = 0 + 0 = 0$. Then $fl(\sqrt{x^2 + y^2}) = fl(\sqrt{0}) = 0$. Finally, the computed value $fl(x/\sqrt{x^2 + y^2}) = fl(10^{-200}/0)$ which evaluates to Inf.

#### 1.4.1.2 Finding a Remedy

Now that we have diagnosed the problem, how do we find a way to do the computation without overflow or underflow?

The essential issue here is that if $x$ is sufficiently large but finite, $x^2$ can overflow. Or if $x$ is sufficiently small but non-zero, then $x^2$ can underflow to zero. The values of $x/\sqrt{x^2 + y^2}$ should be invariant under scaling. That is, if $s > 0$ then

$$\frac{x}{\sqrt{x^2 + y^2}} = \frac{(x/s)}{\sqrt{(x/s)^2 + (y/s)^2}}.$$

By choosing $s$ appropriately we can prevent overflow from happening. We focus on avoiding overflow first, as overflow is more likely to be fatal to the computation. We still need to avoid the denominator underflowing as well. We could scale $(x, y)$ by $|x|$:

$$\frac{(x/|x|)}{\sqrt{(x/|x|)^2 + (y/|x|)^2}} = \frac{\text{sign}\, x}{\sqrt{1 + (y/x)^2}}.$$

This will avoid overflow if $|y/x|$ is not large. So this appears to be appropriate if $|y| \leq |x|$. If $|y| \geq |x|$, then we should be scaling by $|y|$:

$$\frac{(x/|y|)}{\sqrt{(x/|y|)^2 + (y/|y|)^2}} = \frac{(x/|y|)}{\sqrt{(x/y)^2 + 1}}.$$

With these formulas, it is still possible that $(y/x)^2$ underflows if $|y| \leq |x|$, or that $(x/y)^2$ underflows if $|y| \geq |x|$. However, this does not cause problems: for $|y| \leq |x|$, if $(y/x)^2$ underflows, then $|y/x| \leq \sqrt{\mathbf{f}_{\min}} \ll \mathbf{u}$ (at least for the IEEE standards). The computed value of the denominator $fl(\sqrt{1 + (y/x)^2}) = 1$, so that the result is sign $x$, which is the correctly rounded exact value. For $|y| \geq |x|$, if $(x/y)^2$ underflows, then $|x/y| \leq \sqrt{\mathbf{f}_{\min}} \ll \mathbf{u}$, so $fl(\sqrt{(x/y)^2 + 1}) = 1$ and the computed result is $fl(x/|y|)$. The absolute error in this value is no more than $\mathbf{u}$, but it is even better than that. If $fl(x/|y|)$ itself does not underflow, the relative error is no more than $\mathbf{u}$: $x/y = \cot\theta = \cos\theta/\sin\theta$ where $\sin\theta = \text{sign}\, y/\sqrt{1 + (x/y)^2}$ differs from sign $y$ by much less than $\mathbf{u}$. So $\cos\theta = (x/y)\ \sin\theta \approx (x/y)\,\text{sign}\, y$ has a relative error of much less than $\mathbf{u}$, and therefore $fl(x/|y|)$ has a relative error of no more than $2\mathbf{u}$.

**Algorithm 5** Computing $\cos\theta$ and $\sin\theta$ from $(x, y)$

```
1   function cos_sin_xy(x, y)
2     if x = 0 & y = 0: return fail
3     if |x| ≥ |y|
4       r ← y/|x|;  d ← 1/√(1 + r^2)
5       c ← (sign x) d;  s ← (y/|x|) d
6     else
7       r ← x/|y|;  d ← 1/√(1 + r^2)
8       c ← (x/|y|) d;  s ← (sign y) d
9     end if
10    return (c, s)
11  end function
```

Code for implementing the algorithm for computing both $\cos\theta$ and $\sin\theta$ given $(x, y)$ is shown in Algorithm 5. Note that the only case where the algorithm fails is if $(x, y) = (0, 0)$, where the computation is impossible.

This kind of computation is common enough that there is a function `hypot(x,y)` that is available in most libraries for computing $\sqrt{x^2 + y^2}$ without overflow or underflow because of the squaring.

## *1.4.2 Subtracting Nearly Equal Quantities*

### 1.4.2.1 Computing $(1 - \cos x)/x^2$ for $x \approx 0$

Consider the expression $(1 - \cos x)/x^2$. We can use l'Hospital's rule to show that $\lim_{x\to 0}(1 - \cos x)/x^2 = 1/2$. When we compute values numerically (in MATLAB), however, we get the results in Table 1.4.1.

Clearly something goes very wrong for $x \le 10^{-8}$. But even for $x = 10^{-5}$, $10^{-6}$, and $10^{-7}$, there is something unusual happening. Using the Taylor series for $\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \cdots$ we get

$$\frac{1 - \cos x}{x^2} = \frac{1}{2} - \frac{1}{24}x^2 + \frac{1}{720}x^4 - \cdots \approx \frac{1}{2} - \frac{1}{24}x^2.$$

So the value computed for $(1 - \cos x)/x^2$ should be a little below $\frac{1}{2}$. Instead, for $x = 10^{-5}$ and $10^{-6}$ the computed value is a little above $\frac{1}{2}$, while for $x = 10^{-7}$ the computed value is again below $\frac{1}{2}$, but the distance from $\frac{1}{2}$ has only increased for these values of $x$. Why?

We want to understand not just the catastrophic loss of accuracy for $x \le 10^{-8}$, but also the gradual loss of accuracy for $x$ in the range $10^{-5}$ to $10^{-7}$. Assuming no underflow or overflow we will use the formal model (1.3.1) noting that the $\epsilon$ is different for each computation: assuming that $x$ is a normalized floating point number,

**Table 1.4.1** Computed values of $(1-\cos x)/x^2$

| $x$ | $\frac{1-\cos x}{x^2}$ | $x$ | $\frac{1-\cos x}{x^2}$ |
|---|---|---|---|
| $10^{-1}$ | 0.499583472197429 | $10^{-6}$ | 0.500044450291171 |
| $10^{-2}$ | 0.499995833347366 | $10^{-7}$ | 0.499600361081320 |
| $10^{-3}$ | 0.499999958325503 | $10^{-8}$ | 0 |
| $10^{-4}$ | 0.499999996961265 | $10^{-9}$ | 0 |
| $10^{-5}$ | 0.500000041370186 | $10^{-10}$ | 0 |

$$\begin{aligned} fl\left(\frac{1-\cos x}{x^2}\right) &= \frac{fl(1-\cos x)}{fl(x^2)}(1+\epsilon_1) \\ &= \frac{(fl(1)-fl(\cos x))(1+\epsilon_2)}{fl(x)^2(1+\epsilon_3)}(1+\epsilon_1) \\ &= \frac{(1-fl(\cos x))}{x^2}\frac{1+\epsilon_2}{1+\epsilon_3}(1+\epsilon_1), \end{aligned}$$

where $|\epsilon_1|, |\epsilon_2|, |\epsilon_3| \le \mathbf{u}$. Applying the function model to $\cos x$ we have $fl(\cos x) = \cos((1+\epsilon_5)x)(1+\epsilon_6)$. For $x \approx 0$, $\cos x \approx 1 - \frac{1}{2}x^2$ so $fl(\cos x) \approx \cos x + \epsilon_6 + x^2(\epsilon_5 - \frac{1}{2}\epsilon_6)$. Thus, for $x \approx 0$,

$$\begin{aligned} fl\left(\frac{1-\cos x}{x^2}\right) &\approx \frac{1-\cos x - \epsilon_6 + x^2(\epsilon_5 - \frac{1}{2}\epsilon_6)}{x^2}\frac{1+\epsilon_2}{1+\epsilon_3}(1+\epsilon_1) \\ &= \left[\frac{1-\cos x}{x^2} - \frac{\epsilon_6}{x^2} + \epsilon_5 - \frac{1}{2}\epsilon_6\right]\frac{1+\epsilon_2}{1+\epsilon_3}(1+\epsilon_1). \\ &\approx \left[\frac{1}{2} - \frac{1}{24}x^2 - \frac{\epsilon_6}{x^2} + \epsilon_5 - \frac{1}{2}\epsilon_6\right]\frac{1+\epsilon_2}{1+\epsilon_3}(1+\epsilon_1). \end{aligned}$$

The expression

$$\frac{1+\epsilon_2}{1+\epsilon_3}(1+\epsilon_1) \quad \text{differs from 1 by } \le 3\mathbf{u} + \mathcal{O}(\mathbf{u}^2).$$

So the main part of the error for $x$ small is $\epsilon_6/x^2$, which is bounded by $\mathbf{u}/x^2$. If $x = 10^{-8}$ then for double precision, $\mathbf{u}/x^2 \approx 2.2 \times 10^{-16}/(10^{-8})^2 = 2.2$ which is larger than $\frac{1}{2}$, so we do not expect any accuracy for $x = 10^{-8}$ using double precision. Which is what Table 1.4.1 shows. This also allows us to get estimates of the size of the roundoff error in $fl((1-\cos x)/x^2)$: for $x = 10^{-6}$, $\mathbf{u}/x^2 \approx 2.2 \times 10^{-4}$ while the actual roundoff error for $x = 10^{-6}$ is $\approx 4.4 \times 10^{-5}$. The difference between $fl((1-\cos x)/x^2)$ and $\frac{1}{2}$ can be broken down into the roundoff error in computing $(1-\cos x)/x^2$ and the difference $(1-\cos x)/x^2 - \frac{1}{2}$ (called the truncation error). These errors are shown in Figure 1.4.2.

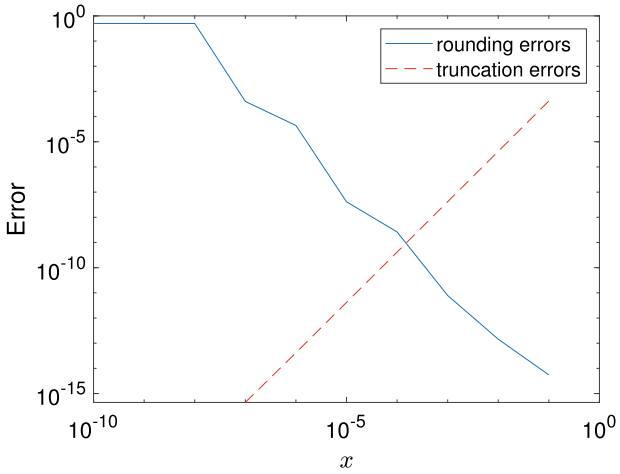



**Fig. 1.4.2** Roundoff and truncation errors for $(1-\cos x)/x^2 \to \frac{1}{2}$ as $x \to 0$

### 1.4.2.2 Subtracting Nearly Equal Quantities

The fundamental issue is that in computing $1-\cos x$ for $x \approx 0$ we are subtracting nearly equal quantities. This results in potentially large *relative* errors. A simple decimal example illustrates this idea:

$$\begin{array}{r} 3.285253249 \\ -3.285252015 \\ \hline 0.000001234 \end{array}$$

The result has about four digits of accuracy while the quantities subtracted each have 10 digits of accuracy. The reason for the loss of the digits of accuracy is the subtraction of nearly equal quantities. This can be seen in the relative error of a difference of two expressions $e_1 - e_2$:

$$\frac{|e_1 - e_2 - fl(e_1 - e_2)|}{|e_1 - e_2|}.$$

Now $fl(e_1 - e_2) = (fl(e_1) - fl(e_2))(1+\epsilon_1)$ with $|\epsilon_1| \le \mathbf{u}$. If $e_1$ and $e_2$ are computed accurately, then $fl(e_1) = e_1(1+\eta_1)$ with $|\eta_1| = \mathcal{O}(\mathbf{u})$ and $fl(e_2) = e_2(1+\eta_2)$ with $|\eta_2| = \mathcal{O}(\mathbf{u})$. Then

$$\begin{aligned} |e_1 - e_2 - fl(e_1 - e_2)| &= |e_1 - e_2 - (e_1(1+\eta_1) - e_2(1+\eta_2))(1+\epsilon_1)| \\ &\le |e_1\eta_1 - e_2\eta_2| + |\epsilon_1|\,|e_1(1+\eta_1) - e_2(1+\eta_2)| \\ &\le |e_1\eta_1 - e_2\eta_2| + |\epsilon_1|\,[|e_1 - e_2| + |e_1\eta_1 - e_2\eta_2|] \\ &= (1+|\epsilon_1|)\,|e_1\eta_1 - e_2\eta_2| + |\epsilon_1|\,|e_1 - e_2|\,. \end{aligned}$$

So

$$\begin{aligned}\frac{|e_1 - e_2 - fl(e_1 - e_2)|}{|e_1 - e_2|} &\leq (1 + |\epsilon_1|)\frac{|e_1\eta_1 - e_2\eta_2|}{|e_1 - e_2|} + |\epsilon_1| \\ &\leq (1 + \mathbf{u})\frac{|e_1| + |e_2|}{|e_1 - e_2|}\max(|\eta_1|, |\eta_2|) + \mathbf{u} \\ &= \frac{|e_1| + |e_2|}{|e_1 - e_2|}\mathcal{O}(\mathbf{u}).\end{aligned}$$

If $(|e_1| + |e_2|)/|e_1 - e_2|$ is large, then there can be a large reduction in the relative accuracy, due to the subtraction of nearly equal quantities. Indeed, if $(|e_1| + |e_2|)/|e_1 - e_2|$ is of the order of $1/\mathbf{u}$, then the resulting error can be 100% or more of the true value $e_1 - e_2$, in which case the loss of precision is regarded as catastrophic: there are no correct digits in the result.

It is therefore desirable to avoid subtracting nearly equal quantities. In some cases, this cannot be avoided. But we try to avoid it where we can.

#### 1.4.2.3 Remedying the Loss of Accuracy

If the problem of computing $(1 - \cos x)/x^2$ for $x \approx 0$ is subtracting $\cos x$ from one, then we should reformulate the expression to avoid this subtracting. If we can perform the subtracting *symbolically* instead of numerically, we can improve the accuracy. For example,

$$\frac{1 - \cos x}{x^2} = \frac{1 - \cos x}{x^2}\frac{1 + \cos x}{1 + \cos x} = \frac{1 - \cos^2 x}{x^2(1 + \cos x)}.$$

Now using $1 - \cos^2 x = \sin^2 x$ we get

$$\frac{1 - \cos x}{x^2} = \frac{\sin^2 x}{x^2(1 + \cos x)} = \left(\frac{\sin x}{x}\right)^2 \frac{1}{1 + \cos x}.$$

No longer do we have a subtraction of nearly equal quantities for $x \approx 0$. This new formula probably would not work well for $x \approx \pm\pi$ as there $1 + \cos x \approx 0$. But for $|x| \leq \pi/2$, for example, this new formula should work well.

There are many other examples of how we can use similar techniques to give new expressions that are equivalent to the original expression in exact arithmetic, but avoids subtraction of nearly equal quantities and give greater accuracy. For example, for $x \approx 0$

$$\begin{aligned}\frac{\sqrt{1+x}-(1+\frac{1}{2}x)}{x^2} &= \frac{\sqrt{1+x}-(1+\frac{1}{2}x)}{x^2}\,\frac{\sqrt{1+x}+(1+\frac{1}{2}x)}{\sqrt{1+x}+(1+\frac{1}{2}x)} \\ &= \frac{(1+x)-(1+\frac{1}{2}x)^2}{x^2(\sqrt{1+x}+(1+\frac{1}{2}x))} = \frac{-\frac{1}{4}x^2}{x^2(\sqrt{1+x}+(1+\frac{1}{2}x))} \\ &= \frac{-1}{4(\sqrt{1+x}+(1+\frac{1}{2}x))}.\end{aligned}$$

Some special cases have support from built-in functions, such as $\texttt{log1p}(x) = \ln(1+x)$ and $\texttt{expm1}(x) = e^x - 1$ for $x \approx 0$. If we use $\texttt{log}(1+x)$ for computing $\ln(1+x)$, the subtraction that occurs is implicit as for the input $z = 1+x$ then

$$\ln(z) = \ln(1+x) = x - \frac{1}{2}x^2 + \frac{1}{3}x^3 - \cdots = (z-1) - \frac{1}{2}(z-1)^2 + \frac{1}{3}(z-1)^3 - \cdots$$

As a result, internally to the call $\log(1+x)$ the log function has to compute $fl((1+x)-1)$, which can result in poor relative accuracy if $x$ is small. Instead, `log1p()` provides a way of computing $\log(1+x)$ accurately for small $x$.

On the other hand, `expm1()` represents an explicit subtraction which is a subtraction of nearly equal quantities if $x$ is small. Both `log1p` and `expm1` are available in the standard libraries of most languages.

### *1.4.3 Numerical Instability*

Loss of accuracy does not necessarily happen all at once. Sometimes it can happen through a process that amplifies errors at each stage until the error overwhelms that computation.

Consider the integral $I_n := \int_0^1 x^n\, e^{-x}\, dx$. This is easy to compute analytically for $n = 0 : \int_0^1 e^{-x}\, dx = 1 - e^{-1}$. We can create a recursive algorithm for computing these values:

$$\begin{aligned}I_{n+1} &= \int_0^1 x^{n+1}\, e^{-x}\, dx = -\int_0^1 x^{n+1}\frac{d}{dx}(e^{-x})\, dx \\ &= -x^{n+1}e^{-x}\Big|_{x=0}^{x=1} + \int_0^1 \frac{d}{dx}(x^{n+1})\, e^{-x}\, dx \\ &= -e^{-1} + (n+1)I_n. \qquad (1.4.3)\end{aligned}$$

With this recurrence we can compute $I_1,\ I_2,\ \ldots$. However, problems become apparent using this scheme by the time we get to $I_{20}$, as can be seen in Table 1.4.2 which shows the computed values for double precision.

The results are evidently wrong for $n \geq 17$ as $I_n > 0$ for all $n$. Furthermore, $I_n \sim 1/(e\,(n+1))$ as $n \to \infty$. The error grows before it becomes evident: $I_5 - \widehat{I}_5 \approx 4.1 \times 10^{-15}$, $I_{10} - \widehat{I}_{10} \approx 1.2 \times 10^{-10}$, and $I_{15} - \widehat{I}_{15} \approx 4.4 \times 10^{-5}$. To see why, consider

$$\begin{aligned} \widehat{I}_{n+1} &= \mathit{fl}((n+1)\widehat{I}_n - e^{-1}) \\ &= ((n+1)\widehat{I}_n(1+\epsilon_{n,1}) - e^{-1})(1+\epsilon_{n,2}) \qquad \text{while} \\ I_{n+1} &= (n+1)\,I_n - e^{-1}. \end{aligned}$$

So

$$I_{n+1} - \widehat{I}_{n+1} = (n+1)(I_n - \widehat{I}_n) + \left[\epsilon_{n,1}(n+1)\widehat{I}_n - \epsilon_{n,2}\,e^{-1}\right].$$

Assuming the quantity in $[\cdots]$ is $\mathcal{O}(\mathbf{u})$, which is reasonable before $\widehat{I}_n$ "blows up", the error in the results grows according to

$$\begin{aligned} I_{n+1} - \widehat{I}_{n+1} &= (n+1)(I_n - \widehat{I}_n) + \mathcal{O}(\mathbf{u}), \qquad \text{so} \\ I_n - \widehat{I}_n &= \mathcal{O}(n!\,\mathbf{u}). \end{aligned}$$

Because $n!$ grows so fast with $n$, it does not take an enormous number of steps before the error in $\widehat{I}_n$ is larger than $I_n$.

The iteration $I_{n+1} = (n+1)\,I_n - e^{-1}$ is an unstable iteration, as small errors are amplified with each step. We can turn this around to our advantage, by reversing the direction of the iteration:

$$I_n = \frac{1}{n+1}\left[I_{n+1} + e^{-1}\right]. \tag{1.4.4}$$

The recurrence (1.4.4) is actually very stable. It is so stable, in fact, that we can start with $n = n_0$ for some $n_0$ such as $n_0 = 40$ with a rough approximation for $\widetilde{I}_{n_0}$ and work backward to obtain more accurate approximations than we get from the forward recurrence. Errors in the computed values of $\widetilde{I}_n$ for selected values of $n$ are shown in Table 1.4.3.

The results in Table 1.4.3 indicate full double precision accuracy for these results.

### *1.4.4 Adding Many Numbers*

The apparently trivial problem of adding many numbers can reveal surprising depth in numerical computation. If we aim for maximal accuracy, such as implementing a special mathematical function for many people to use, or doing some other high-precision computation, we might want to find how to add numbers with the least error.

**Table 1.4.2** Computed values of $I_n = \int_0^1 x^n\, e^{-x}\, dx$

| $n$ | $\widehat{I}_n$ | $n$ | $\widehat{I}_n$ | $n$ | $\widehat{I}_n$ |
|---|---|---|---|---|---|
| 0 | 0.6321205588285577 | 8 | 0.04043407756729511 | 16 | 0.009553035697593693 |
| 1 | 0.2642411176571153 | 9 | 0.03646133450150879 | 17 | −0.19592479861475587 |
| 2 | 0.1606027941427883 | 10 | 0.03319523834515437 | 18 | $-4.090450614851804 \times 10^{0}$ |
| 3 | 0.1139289412569227 | 11 | 0.03046341897041005 | 19 | $-8.217689173820752 \times 10^{1}$ |
| 4 | 0.0878363238562483 | 12 | 0.02814500544388832 | 20 | $-1.726082605943529 \times 10^{3}$ |
| 5 | 0.0713021781097991 | 13 | 0.02615063504299409 | 21 | $-3.797418521019881 \times 10^{4}$ |
| 6 | 0.0599336274873523 | 14 | 0.02438008447346896 | 22 | $-8.734066277140138 \times 10^{5}$ |
| 7 | 0.0516559512400239 | 15 | 0.02220191040406094 | 23 | $-2.096175943301578 \times 10^{7}$ |

**Table 1.4.3** Errors in computed values of $I_n$ using reverse recurrence (1.4.4)

| $n$ | $\widetilde{I}_n - I_n$ |
|---|---|
| 0 | $-1.24 \times 10^{-17}$ |
| 5 | $+5.40 \times 10^{-19}$ |
| 10 | $-1.55 \times 10^{-18}$ |
| 15 | $+1.48 \times 10^{-18}$ |
| 20 | $+8.57 \times 10^{-20}$ |

**Algorithm 6** Naive algorithm for adding numbers in an array

```
1   function sum(a)
2     s ← 0
3     for i = 1, 2, ..., length(a)
4       s ← s + a_i
5     end for
6     return s
7   end function
```

The standard, or naive, algorithm for adding an array of numbers is shown in Algorithm 6.

For the error analysis for Algorithm 6, we use the notation $\widehat{s}_i$ to be the computed value of $s$ after adding $a_i$ on line 4. Let $\widehat{s}_0 = 0$. Then $\widehat{s}_{i+1} = fl(\widehat{s}_i + a_i) = (\widehat{s}_i + a_i)(1 + \epsilon_i)$ with $|\epsilon_i| \le \mathbf{u}$. If $n = length(\boldsymbol{a})$, the returned value is

$$\widehat{s}_n = \sum_{i=1}^{n} a_i \prod_{j=i}^{n} (1 + \epsilon_j), \qquad \text{which leads to the bound}$$

$$\left| \widehat{s}_n - \sum_{i=1}^{n} a_i \right| \le \sum_{i=1}^{n} |a_i|\, (n - i + 1)\mathbf{u} + \mathcal{O}(n\mathbf{u})^2.$$

This means that for maximum accuracy numbers should be added *from smallest to largest*.

But our analysis of the naive Algorithm 6 indicates that the more additions applied to a sum with a given term $a_i$, the larger the bound on the roundoff error. We can think of this in terms of the depth of term in the sum:

$$((\cdots(((a_1 + a_2) + a_3) + a_4)\cdots + a_{n-2}) + a_{n-1}) + a_n.$$

If we reduce the depth of the terms, we have the possibility of reducing the error in the sum. One way of reducing the maximum depth is to split sums in the middle: $\sum_{\ell=i}^{j} a_\ell = \sum_{\ell=i}^{m} a_\ell + \sum_{\ell=m+1}^{j} a_\ell$ with $m = \lfloor (i+j)/2 \rfloor$, for example. The maximum depth is then $\lceil \log_2 n \rceil$ and we can get bounds on the rounding error of $\approx (\log_2 n)\mathbf{u} \max_i |a_i|$. Pseudo-code for this can be found in Algorithm 2.

Finally, the pseudo-random character of roundoff error should remind us that sometimes a statistical analysis can be beneficial for understanding the behavior of roundoff error for long sums of terms of similar magnitude, as can occur solving ordinary differential equations.

**Exercises.**

(1) Carry out the following computations: initially set $x$ to 2; then repeat $x \leftarrow \sqrt{x}$ 20 times, and then repeat $x \leftarrow x^2$ 20 times. If everything was done in exact arithmetic you would get exactly 2 for the final value of $x$. What do you get? Can you explain why? [**Hint:** Note that $(a+\epsilon)^2 = a^2 + 2a\epsilon + \epsilon^2$. If $a$ is the exact value, ignoring the $\epsilon^2$ term, the error $\epsilon$ is amplified by a factor of $2a$.]

(2) The most common statistics computed from a data set $x_1, x_2, \ldots, x_N$ are the mean $\overline{x} = (1/N)\sum_{i=1}^{N} x_i$ and variance $s^2 = (N-1)^{-1}\sum_{i=1}^{N}(x_i - \overline{x})^2$. There is a second, equivalent, formula for the variance $s^2 = (N-1)^{-1}[\left(\sum_{i=1}^{N} x_i^2\right) - N\overline{x}^2]$. However, they are not equivalent numerically. Create a set of $N = 100$ values $x_i = 10^9 + v_i$ where $v_i$ are randomly generated numbers uniformly over the interval $[-1, +1]$. Compute the variance $s^2$ by these two formulas. Which is more accurate? Explain why.

(3) Compute the values of $((1+x)^{1/3} - 1)/x$ for $x = 10^{-k}$ for $k = 1, 2, \ldots, 15$. Compute the limit of the expression as $x \to 0$ using l'Hospital's rule. Re-write the formula $((1+x)^{1/3} - 1)/x$ to give an equivalent expression (assuming exact arithmetic) that is more accurate for small $x$.

(4) The function $f(x) = e^x - 1$, if implemented directly, is not accurate for $x \approx 0$ because of the subtraction of nearly equal quantities. Show that evaluating the expression in floating point

$$\frac{e^x - 1}{(1+x) - 1}x$$

has a relative error of no more than unit roundoff $\mathbf{u}$ for $0 < x < \sqrt{\mathbf{u}}$.

(5) The hyperbolic tangent function $\tanh(x) = (e^{+x} - e^{-x})/(e^{+x} + e^{-x})$ implemented like this gives NaN if $x$ is large, for example, if $x = \pm 1000$ when using IEEE double precision. Write a new implementation that gives accurate values but avoids generating NaNs.

(6) The expression $(\tan x - \sin x)/x^3$ gives inaccurate values for $x \approx 0$ due to subtraction of nearly equal quantities. Re-write this in an equivalent way (in exact arithmetic) that will give accurate values for $x \approx 0$.

(7) The function $g(u) = \log(1 + e^u)$ is a smooth approximation to $\max(0, u)$. However, it suffers from overflow if $x$ is positive and large enough. Implement this function so that overflow does not occur for any reasonable value of $u$. Reasonable values must include $u = \pm 10^3$.

(8) The update formula (1.4.3) for computing $\int_0^1 x^n e^{-x}\, dx$ amplifies errors, while the reversed iteration (1.4.4) $I_n = \frac{1}{n+1}\left[I_{n+1} + e^{-1}\right]$ reduces the error. Implement a method to compute $I_n$ for any given $n$ using the reversed iteration and a starting value $\widehat{I}_m \leftarrow e^{-1}/m$ for some $m > n$. To test your method, compare the computed value $\widehat{I}_n$ with $\int_0^1 x^n e^{-x}\, dx = \sum_{k=0}^{\infty}(-1)^k/((n+k+1)k!)$.

(9) If $a \approx b$ are large positive numbers, which gives the smaller roundoff error in general: $a^2 - b^2$ or $(a - b)(a + b)$? Give bounds on the roundoff error for these two expressions based on the formal model (1.3.1) assuming no overflow or underflow. Which expression is more likely to produce overflow?

(10) Write a recursive routine to compute the cardinality $n(S) = |S|$, the mean $\mu(S) = \sum_{i \in S} x_i$, and sum $ss(S) = \sum_{i \in S}(x_i - \mu(S))^2$ for a set of indexes $S = \{\, i \mid k \le i \le \ell \,\}$. The core of the routine is how it computes $(n(S),\ \mu(S),\ ss(S))$ from $(n(S_1),\ \mu(S_1),\ ss(S_1))$ and $(n(S_2),\ \mu(S_2),\ ss(S_2))$ where $S = S_1 \cup S_2$ and $S_1 \cap S_2 = \emptyset$. Do this in a way to avoid subtraction of large, nearly equal, quantities.

## 1.5 Measuring: Norms

We need to measure the size of an error, whether the error is a scalar, a vector, or a matrix. So we need to measure the "size" of a vector or matrix. There are many ways of doing this. Some measures of size are more suitable to certain applications (such as geometry or the worst-case error). But all measures have basic properties that must hold in order to use them to prove error bounds.

### *1.5.1 What Is a Norm?*

A norm is a real-valued function $\|\cdot\|$ on a vector space that we use to measure the size of vectors in that space. Norms have the following properties:

- $\|\boldsymbol{x}\| \ge 0$ for all vectors $\boldsymbol{x}$ and $\|\boldsymbol{x}\| = 0$ implies $\boldsymbol{x} = \boldsymbol{0}$;
- $\|s\boldsymbol{x}\| = |s|\,\|\boldsymbol{x}\|$ for any vector $\boldsymbol{x}$ and scalar $s$;
- $\|\boldsymbol{x} + \boldsymbol{y}\| \le \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$ for vectors $\boldsymbol{x}$ and $\boldsymbol{y}$.

The last inequality is called the *triangle inequality*. Norms of matrices have those properties:

- $\|A\| \geq 0$ for all matrices $A$ and $\|A\| = 0$ implies $A = 0$;
- $\|sA\| = |s| \, \|A\|$ for all matrices $A$ and scalar $s$;
- $\|A + B\| \leq \|A\| + \|B\|$ for all matrices $A$ and $B$,

along with these additional properties provided the matrix and vector norms are compatible:

- $\|A\boldsymbol{x}\| \leq \|A\| \, \|\boldsymbol{x}\|$
- $\|AB\| \leq \|A\| \, \|B\|$.

**Example 1.1** Norm examples.

Vector norms in common use include the following:

- $\|\boldsymbol{x}\|_2 = \left[\sum_i |x_i|^2\right]^{1/2} = \sqrt{\boldsymbol{x}^T\boldsymbol{x}}$ for real $\boldsymbol{x}$;
- $\|\boldsymbol{x}\|_\infty = \max_i |x_i|$;
- $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$.

These norms are part of family of norms: $\|\boldsymbol{x}\|_p = \left[\sum_i |x_i|^p\right]^{1/p}$ for $p \geq 1$. Note that $\|\boldsymbol{x}\|_\infty = \lim_{p\to\infty} \|\boldsymbol{x}\|_p$.

Matrix norms can be *induced* by vector norms:

$$\|A\| = \sup_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|A\boldsymbol{x}\|}{\|\boldsymbol{x}\|}. \tag{1.5.1}$$

Note that the norms on the right are both *vector* norms.

Induced matrix norms are compatible with the vector norms used to define them. Formulas are known for the induced matrix norms of the 1-, 2-, and $\infty$-norms:

- $\|A\|_2 = \sqrt{\lambda_{max}(A^T A)}$ where $\lambda_{max}(B)$ is the maximum eigenvalue of $B$;
- $\|A\|_\infty = \max_i \sum_j \left|a_{ij}\right|$;
- $\|A\|_1 = \max_j \sum_i \left|a_{ij}\right|$.

New norms can be created from old norms. For example, if $D$ is an invertible matrix, then we can define the scaled norms:

$$\|\boldsymbol{x}\|_{D,p} = \|D\boldsymbol{x}\|_p .$$

The corresponding induced matrix norm is

$$\begin{aligned} \|A\|_{D,p} &= \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|A\boldsymbol{x}\|_{D,p}}{\|\boldsymbol{x}\|_{D,p}} = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|DA\boldsymbol{x}\|_p}{\|D\boldsymbol{x}\|_p} = \max_{\boldsymbol{z}\neq\boldsymbol{0}} \frac{\left\|DAD^{-1}\boldsymbol{z}\right\|_p}{\|\boldsymbol{z}\|_p} \qquad (\boldsymbol{z} = D\boldsymbol{x}) \\ &= \left\|DAD^{-1}\right\|_p . \end{aligned}$$

A special matrix norm that is easy to compute is the *Frobenius norm*:

$$\|A\|_F = \left[ \sum_{i,j} \left|a_{ij}\right|^2 \right]^{1/2} .$$

The Frobenius norm is not an induced norm, but it is compatible with the vector 2-norm:

$$\|A\boldsymbol{x}\|_2 \le \|A\|_F \, \|\boldsymbol{x}\|_2 .$$

A fact that is important for theory is that *any* two norms $\|\cdot\|_{(1)}$ and $\|\cdot\|_{(2)}$ on a finite-dimensional vector space are *equivalent* in the sense that there is a constant $c > 0$ where

$$\frac{1}{c} \|\boldsymbol{x}\|_{(1)} \le \|\boldsymbol{x}\|_{(2)} \le c \, \|\boldsymbol{x}\|_{(1)} \qquad \text{for all } \boldsymbol{x}. \tag{1.5.2}$$

### *1.5.2 Norms of Functions*

How big is a function? It depends on the norm you use to measure it. The usual norm properties must hold: for any functions $f, g : D \to \mathbb{R}$

- $\|f\| \ge 0$, and $\|f\| = 0$ implies $f(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in D$;
- $\|\alpha \, f\| = |\alpha| \, \|f\|$ for any scalar (constant) $\alpha$;
- $\|f + g\| \le \|f\| + \|g\|$.

Some examples of norms of functions $[a, b] \to \mathbb{R}$ follow:

- $\|f\|_\infty = \max_{a \le x \le b} |f(x)|$ for continuous $f$;
- $\|f\|_1 = \int_a^b |f(x)| \, dx$;
- $\|f\|_p = \left[ \int_a^b |f(x)|^p \, dx \right]^{1/p}$ for $1 \le p < \infty$;
- $\|f\|_{1,p} = \left[ \int_a^b \left( |f(x)|^p + |f'(x)|^p \right) dx \right]^{1/p}$ for $1 \le p < \infty$.

Some examples of norms of functions $D \to \mathbb{R}$ where $D \subset \mathbb{R}^d$ is bounded and closed follow:

- $\|f\|_\infty = \max_{\boldsymbol{x} \in D} |f(\boldsymbol{x})|$ for continuous $f$;
- $\|f\|_1 = \int_D |f(\boldsymbol{x})| \, d\boldsymbol{x}$;
- $\|f\|_p = \left[ \int_D |f(\boldsymbol{x})|^p \, d\boldsymbol{x} \right]^{1/p}$ for $1 \le p < \infty$;
- $\|f\|_{1,p} = \left[ \int_D \left( |f(\boldsymbol{x})|^p + \|\nabla f(\boldsymbol{x})\|^p \right) d\boldsymbol{x} \right]^{1/p}$ for $1 \le p < \infty$.

There are also weighted norms, using a weight function $w(\boldsymbol{x})$ that is positive except on a set of zero volume, such as

- $\|f\|_{p,w} = \left[ \int_D w(\boldsymbol{x}) \, |f(\boldsymbol{x})|^p \, d\boldsymbol{x} \right]^{1/p}$ for $1 \le p < \infty$.

**Exercises.**

(1) Show that for any vector $\boldsymbol{x}$ we have $\|\boldsymbol{x}\|_\infty \le \|\boldsymbol{x}\|_2 \le \|\boldsymbol{x}\|_1$.
(2) Show that for any vector $\boldsymbol{x} \in \mathbb{R}^n$ we have $\|\boldsymbol{x}\|_\infty \le \|\boldsymbol{x}\|_2 \le n^{1/2}\,\|\boldsymbol{x}\|_\infty$ so that the 2-norm and $\infty$-norm on $\mathbb{R}^n$ are equivalent norms.
(3) ⚠Show that *any* two norms on $\mathbb{R}^n$ are equivalent. [**Hint:** First show that a given norm $\|\cdot\|$ is equivalent to $\|\cdot\|_\infty$.]
(4) The norms on functions $\|f\|_1 = \int_a^b |f(x)|\,dx$ and $\|f\|_2 = \left[\int_a^b |f(x)|^2\,dx\right]^{1/2}$ over the interval $[a, b]$ are *not* equivalent. Show first that $\|f\|_1 \le \sqrt{b-a}\;\|f\|_2$. Now find a sequence of function $f_k\colon [0,\,1] \to \mathbb{R}$ where $\|f_k\|_2 = 1$ for all $k$, but $\|f_k\|_1 \to 0$ as $k \to \infty$. [**Hint:** Set $f_k(x) = \sqrt{k}$ on the interval $[0,\ 1/k]$ and zero elsewhere.]
(5) Show that the matrix 2-norm of an outer product $\left\|\boldsymbol{u}\boldsymbol{v}^T\right\|_2$ is given by $\left\|\boldsymbol{u}\boldsymbol{v}^T\right\|_2 = \|\boldsymbol{u}\|_2\,\|\boldsymbol{v}\|_2$.
(6) Use the Cauchy–Schwarz inequality (A.1.4) to show that $\|A\boldsymbol{x}\|_2 \le \|A\|_F\,\|\boldsymbol{x}\|_2$, and so $\|A\|_F \ge \|A\|_2$ for any matrix $A$.
(7) The integral version of the previous exercise is to show that if $g(x) = \int_a^b k(x, y)\,f(y)\,dy$, then $\|g\|_2 \le \left[\int_a^b \int_a^b |k(x, y)|^2\,dx\,dy\right]^{1/2} \|f\|_2$. Show this using the integral version of the Cauchy–Schwarz inequality (A.1.5).
(8) Show that if we treat $\boldsymbol{x}^T$ as a $1 \times n$ matrix, then $\left\|\boldsymbol{x}^T\right\|_\infty = \|\boldsymbol{x}\|_1$. From this, or directly, show that $\left|\boldsymbol{x}^T\boldsymbol{y}\right| \le \|\boldsymbol{x}\|_1\,\|\boldsymbol{y}\|_\infty$.
(9) Show that for any induced matrix norm (1.5.1) $\|I\| = 1$. From this show that the Frobenius norm $\|A\|_F$ is not an induced matrix norm.
(10) Show that the 2-norm of both matrices and vectors is orthogonally invariant. That is, if $Q$ and $W$ are orthogonal matrices ($Q^{-1} = Q^T$ and $W^{-1} = W^T$) then $\|Q\boldsymbol{x}\|_2 = \|\boldsymbol{x}\|_2$ and $\|QAW\|_2 = \|A\|_2$ for vectors $\boldsymbol{x}$ and matrices $A$ of appropriate dimensions. Also show that $\left\|A^T\right\|_2 = \|A\|_2$. [**Hint:** For the last part, use $\|\boldsymbol{x}\|_2 = \max_{\boldsymbol{y}:\|\boldsymbol{y}\|_2=1} \boldsymbol{y}^T\boldsymbol{x}$ and $\boldsymbol{y}^T A\boldsymbol{x} = \boldsymbol{x}^T A^T\boldsymbol{y}$ starting with the definition of $\|A\|_2$.]

## 1.6 Taylor Series and Taylor Polynomials

James Gregory gave the first presentations of Taylor series representations for the standard trigonometric functions in 1667, although this had already been done in the 1400s in India by Madhava of Sangamagrama. The general procedure was developed and described by Brook Taylor in 1715. The standard Taylor series is an infinite series whose convergence is required before it can be used for computations. By contrast, Taylor series with remainder give a finite Taylor polynomial and a remainder term that indicates the error in the polynomial approximating the original function. This remainder form is actually much more useful for numerical computation, as well as avoiding the convergence issues associated with the original infinite Taylor series.

### 1.6.1 Taylor Series in One Variable

Here we show how to obtain Taylor series with integral remainder.

**Theorem 1.2** *(Taylor series with remainder) If $f$ is $n+1$ times continuously differentiable, then*

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \cdots + \frac{1}{n!}f^{(n)}(a)(x-a)^n + \frac{1}{n!}\int_a^x (x-t)^n f^{(n+1)}(t)\,dt. \tag{1.6.1}$$

The polynomial $f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \cdots + (1/n!)f^{(n)}(a)(x-a)^n$ is called the *Taylor polynomial* of $f$ at $a$ of order $n$. The remaining term $(1/n!)\int_a^x (x-t)^n f^{(n+1)}(t)\,dt$ is called the *remainder in integral form.*

***Proof*** We prove (1.6.1) by induction on $n$, starting with $n = 0$.

For $n = 0$ we use the fundamental theorem of calculus:

$$f(x) = f(a) + \int_a^x f'(t)\,dt.$$

Suppose that (1.6.1) is true for $n = k$; we wish to show that it holds for $n = k+1$.

Using integration by parts,

$$\begin{aligned}\int_a^x (x-t)^k\, f^{(k+1)}(t)\,dt &= -\frac{1}{k+1}\int_a^x \frac{d}{dt}\left[(x-t)^{k+1}\right] f^{(k+1)}(t)\,dt \\ &= -\frac{1}{k+1}\,(x-t)^{k+1}\, f^{(k+1)}(t)\Big|_{t=a}^{t=x} + \frac{1}{k+1}\int_a^x (x-t)^{k+1}\, f^{(k+2)}(t)\,dt \\ &= \frac{1}{k+1}(x-a)^{k+1}\, f^{(k+1)}(a) + \frac{1}{k+1}\int_a^x (x-t)^{k+1}\, f^{(k+2)}(t)\,dt.\end{aligned}$$

Then

$$\begin{aligned}f(x) &= \sum_{j=0}^{k}\frac{1}{j!}f^{(j)}(a)\,(x-a)^j + \frac{1}{k!}\int_a^x (x-t)^k\, f^{(k+1)}(t)\,dt \\ &\qquad \text{(by the induction hypothesis)} \\ &= \sum_{j=0}^{k}\frac{1}{j!}f^{(j)}(a)\,(x-a)^j + \frac{1}{k!\,(k+1)}f^{(k+1)}(a)\,(x-a)^{k+1} \\ &\qquad + \frac{1}{k!\,(k+1)}\int_a^x (x-t)^{k+1}\, f^{(k+2)}(t)\,dt \\ &= \sum_{j=0}^{k+1}\frac{1}{j!}f^{(j)}(a)\,(x-a)^j + \frac{1}{(k+1)!}\int_a^x (x-t)^{k+1}\, f^{(k+2)}(t)\,dt,\end{aligned}$$

which is what we wanted to prove. □

The Taylor series representation of $f$

$$f(x) = f(a) + f'(a)(x-a) + \frac{1}{2}f''(a)(x-a)^2 + \cdots + \frac{1}{n!}f^{(n)}(a)(x-a)^n + \cdots$$

holds provided both the infinite series converges, and the remainder term $R_n(x) := (1/n!)\int_a^x (x-t)^n f^{(n+1)}(t)\,dt$ goes to zero as $n \to \infty$.

Another form of the remainder term that is less precise, but often easier to use, is the point form. By the integral mean value theorem, since the $(x-t)^n$ does not change sign for $a < t < x$,

$$\begin{aligned} R_n(x) &= \frac{1}{n!}\int_a^x (x-t)^n\, f^{(n+1)}(t)\,dt \\ &= \frac{1}{n!}f^{(n+1)}(c_x)\int_a^x (x-t)^n\,dt \\ &= \frac{1}{n!}f^{(n+1)}(c_x)\frac{1}{n+1}(x-a)^{n+1} \\ &= \frac{1}{(n+1)!}f^{(n+1)}(c_x)(x-a)^{n+1} \end{aligned}$$

for some $c_x$ between $a$ and $x$. It should be noted that $c_x$ depends not only on $x$, but also on $n$, $a$, and the function $f$.

The remainder term $R_n(x)$ can be bounded for many well-known functions: if $f(x) = e^x$, since $f^{(n)}(x) = (d^n/dx^n)e^x = e^x$, for $a = 0$ we have

$$R_n(x) = \frac{1}{(n+1)!}e^{c_x}(x-a)^{n+1} \qquad \text{for some } c_x \text{ between 0 and } x.$$

As the exponential function is an increasing function,

$$|R_n(x)| \le \frac{1}{(n+1)!}e^{\max(0,x)}\,|x|^{n+1}\,.$$

This bound can be used to determine, for example, the order of the Taylor polynomial needed to obtain a specified error level. For example, to guarantee an error level of no more than $10^{-12}$ for $|x| \le 1/2$, we would ensure first that

$$\frac{1}{(n+1)!}e^{\max(0,x)}\,|x|^{n+1} \le 10^{-12} \qquad \text{for all } |x| \le \frac{1}{2}.$$

Taking the worst case over $-\frac{1}{2} \le x \le +\frac{1}{2}$ gives the condition

$$\frac{1}{(n+1)!}e^{1/2}(\frac{1}{2})^{n+1} \le 10^{-12}.$$

**Table 1.6.1** Bounds on Taylor polynomial remainder term for $f(x) = e^x$, $a = 0$, and $|x| \le 1/2$

| $n$ | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | $2.061 \times 10^{-1}$ | $3.435 \times 10^{-2}$ | $4.294 \times 10^{-3}$ | $4.293 \times 10^{-4}$ | $3.577 \times 10^{-5}$ | $2.555 \times 10^{-6}$ |
| $n$ | 7 | 8 | 9 | 10 | 11 | 12 |
| | $1.597 \times 10^{-7}$ | $8.874 \times 10^{-9}$ | $4.437 \times 10^{-10}$ | $2.017 \times 10^{-11}$ | $8.403 \times 10^{-13}$ | $3.232 \times 10^{-14}$ |

Finding the smallest $n$ satisfying this condition can be carried out by trial and error; values are shown in Table 1.6.1. Clearly $n = 11$ is sufficient to achieve a reminder less than $10^{-12}$ for $|x| \le \frac{1}{2}$.

### 1.6.2 Taylor Series and Polynomials in More than One Variable

Taylor series with remainder formulas can also be developed for functions of more than one variable. The trick here is to first reduce the problem to a one-variable problem. Suppose $f\colon \mathbb{R}^m \to \mathbb{R}$. Provided $f$ is continuously differentiable $n+1$ times, the function $\phi(t) := f(\boldsymbol{a} + t\boldsymbol{d})$ is also $n+1$ times continuously differentiable and so

$$\phi(t) = \phi(0) + \phi'(0)t + \frac{1}{2}\phi''(0)t^2 + \cdots + \frac{1}{n!}\phi^{(n)}(0)t^n + \frac{1}{n!}\int_0^t (t-u)^n \phi^{(n+1)}(u)\,du.$$

Then

$$f(\boldsymbol{a}+t\boldsymbol{d}) = f(\boldsymbol{a}) + \frac{d}{ds}\, f(\boldsymbol{a}+s\boldsymbol{d})|_{s=0}\, t + \frac{1}{2}\frac{d^2}{ds^2}\, f(\boldsymbol{a}+s\boldsymbol{d})|_{s=0}\, t^2 + \cdots + \frac{1}{n!}\frac{d^n}{ds^n}\, f(\boldsymbol{a}+s\boldsymbol{d})|_{s=0}\, t^n + \frac{1}{n!}\int_0^t (t-u)^n \frac{d^{n+1}}{ds^{n+1}}\, f(\boldsymbol{a}+s\boldsymbol{d})|_{s=u}\, du.$$

Computing the derivatives $(d/ds)^k f(\boldsymbol{a}+s\boldsymbol{d})$ can be done in terms of the partial derivatives of $f$, although the formulas are not especially nice:

$$\frac{d^k}{ds^k} f(\boldsymbol{a}+s\boldsymbol{d}) = \sum_{i_1,i_2,\ldots,i_k=1}^{m} \frac{\partial^k f}{\partial x_{i_1}\partial x_{i_2}\cdots\partial x_{i_k}}(\boldsymbol{a}+s\boldsymbol{d})\, d_{i_1} d_{i_2}\cdots d_{i_k}.$$

To help us with the task of dealing with such terms, we introduce some new notation:

$$D^k f(\boldsymbol{a})[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k] = \sum_{i_1,i_2,\ldots,i_k=1}^{m} \frac{\partial^k f}{\partial x_{i_1}\partial x_{i_2}\cdots\partial x_{i_k}}(\boldsymbol{a})\,(\boldsymbol{v}_1)_{i_1}(\boldsymbol{v}_2)_{i_2}\cdots(\boldsymbol{v}_k)_{i_k}. \tag{1.6.2}$$

The values

$$\frac{\partial^k f}{\partial x_{i_1}\partial x_{i_2}\cdots\partial x_{i_k}}(\boldsymbol{a}) = \frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{a}), \tag{1.6.3}$$

where $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_d)$ is a *multi-index*: $\alpha_j$ is the number of times $j$ appears in the list $(i_1, i_2, \ldots, i_k)$. Since $\partial^2 g/\partial x_i \partial x_j = \partial^2 g/\partial x_j \partial x_i$ provided the second derivatives are continuous, where $j$ appears in the list $(i_1, i_2, \ldots, i_k)$ does not matter, only how many times it appears. The value $|\boldsymbol{\alpha}| = \alpha_1 + \alpha_2 + \cdots + \alpha_d = k$ is the order of the derivative. Multi-indexes can also be used to label multi-variable monomials: $\boldsymbol{z}^{\boldsymbol{\alpha}} = z_1^{\alpha_1} z_2^{\alpha_2} \cdots z_d^{\alpha_d}$, which has degree $|\boldsymbol{\alpha}|$. A general polynomial in $d$ variables of degree $k$ can be written as

$$p(\boldsymbol{x}) = \sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|\le k} c_{\boldsymbol{\alpha}} \boldsymbol{x}^{\boldsymbol{\alpha}}. \tag{1.6.4}$$

With the notation of (1.6.2),

$$\frac{d^k}{ds^k} f(\boldsymbol{a} + s\boldsymbol{d}) = D^k f(\boldsymbol{a} + s\boldsymbol{d})[\underbrace{\boldsymbol{d}, \boldsymbol{d}, \ldots, \boldsymbol{d}}_{k \text{ times}}].$$

Then the Taylor series representation of $f(\boldsymbol{a} + t\boldsymbol{d})$ is

$$f(\boldsymbol{a} + t\boldsymbol{d}) = f(\boldsymbol{a}) + D^1 f(\boldsymbol{a})[\boldsymbol{d}]t + \frac{1}{2} D^2 f(\boldsymbol{a})[\boldsymbol{d}, \boldsymbol{d}]t^2 + \cdots + \frac{1}{n!} D^n f(\boldsymbol{a})[\underbrace{\boldsymbol{d}, \boldsymbol{d}, \ldots, \boldsymbol{d}}_{n \text{ times}}]t^n$$

$$+ \frac{1}{n!} \int_0^t (t-u)^n D^{n+1} f(\boldsymbol{a} + u\boldsymbol{d})[\underbrace{\boldsymbol{d}, \boldsymbol{d}, \ldots, \boldsymbol{d}}_{n+1 \text{ times}}]\, du. \tag{1.6.5}$$

The quantity $D^1 f(\boldsymbol{a})$ can be understood as the *gradient vector* of $f$ at $\boldsymbol{a}$:

$$\nabla f(\boldsymbol{a}) = \begin{bmatrix} \partial f/\partial x_1(\boldsymbol{a}) \\ \partial f/\partial x_2(\boldsymbol{a}) \\ \vdots \\ \partial f/\partial x_m(\boldsymbol{a}) \end{bmatrix},$$

and $D^1 f(\boldsymbol{a})[\boldsymbol{d}] = \boldsymbol{d}^T \nabla f(\boldsymbol{a})$, while $D^2 f(\boldsymbol{a})$ can be understood as the *Hessian matrix* of $f$ at $\boldsymbol{a}$:

$$\operatorname{Hess} f(\boldsymbol{a}) = \begin{bmatrix} \partial^2 f/\partial x_1 \partial x_1(\boldsymbol{a}) & \partial^2 f/\partial x_1 \partial x_2(\boldsymbol{a}) & \cdots & \partial^2 f/\partial x_1 \partial x_m(\boldsymbol{a}) \\ \partial^2 f/\partial x_2 \partial x_1(\boldsymbol{a}) & \partial^2 f/\partial x_2 \partial x_2(\boldsymbol{a}) & \cdots & \partial^2 f/\partial x_2 \partial x_m(\boldsymbol{a}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial^2 f/\partial x_m \partial x_1(\boldsymbol{a}) & \partial^2 f/\partial x_m \partial x_2(\boldsymbol{a}) & \cdots & \partial^2 f/\partial x_m \partial x_m(\boldsymbol{a}) \end{bmatrix},$$

and $D^2 f(\boldsymbol{a})[\boldsymbol{d}, \boldsymbol{d}] = \boldsymbol{d}^T \text{Hess}\, f(\boldsymbol{a})\, \boldsymbol{d}$. Note that since $\partial^2 f/\partial x_i \partial x_j = \partial^2 f/\partial x_j \partial x_i$ provided the second-order derivatives are continuous, Hess $f(\boldsymbol{a})$ is a symmetric matrix.

The quantities $D^k f(\boldsymbol{a})$ can be considered to be higher order *tensors*, which at an elementary level can be thought of as higher dimensional arrays of numbers. However, the fact that $\partial^2 f/\partial x_i \partial x_j = \partial^2 f/\partial x_j \partial x_i$ means that these tensors have certain symmetry relations. In particular, if $(\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k)$ is a permutation of $(\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k)$,

$$D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k] = D^k f(\boldsymbol{a})[\boldsymbol{e}_1, \boldsymbol{e}_2, \ldots, \boldsymbol{e}_k]. \tag{1.6.6}$$

Also note that $D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k]$ is linear in each of the $\boldsymbol{d}_j$'s:

$$D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \ldots, \boldsymbol{u} + \boldsymbol{v}, \ldots, \boldsymbol{d}_k] = D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \ldots, \boldsymbol{u}, \ldots, \boldsymbol{d}_k] + D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \ldots, \boldsymbol{v}, \ldots, \boldsymbol{d}_k]. \tag{1.6.7}$$

This makes the function $(\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k) \mapsto D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \boldsymbol{d}_2, \ldots, \boldsymbol{d}_k]$ *multilinear* for $k > 1$ and not linear, just as the function $(x, y) \mapsto xy$ is linear in $x$ and linear in $y$ but is not linear in $(x, y)$.

The remainder term in integral form

$$\frac{1}{n!} \int_0^t (t-u)^n D^{n+1} f(\boldsymbol{a} + u\boldsymbol{d})[\underbrace{\boldsymbol{d}, \boldsymbol{d}, \ldots, \boldsymbol{d}}_{n+1 \text{ times}}]\, du$$

can be represented in point form

$$\frac{1}{(n+1)!} D^{n+1} f(\boldsymbol{a} + c_t \boldsymbol{d})[\underbrace{\boldsymbol{d}, \boldsymbol{d}, \ldots, \boldsymbol{d}}_{n+1 \text{ times}}]$$

for some $c_t$ between 0 and $t$, since we are integrating a scalar quantity.

We can bound $D^k f(\boldsymbol{a})[\boldsymbol{d}, \cdots, \boldsymbol{d}]$ in terms of the $k$th-order partial derivatives of $f$: if $\left|\partial^k f/\partial x_{i_1} \partial x_{i_2} \cdots \partial x_{i_k}(\boldsymbol{a})\right| \le M$ for all $(i_1, i_2, \ldots, i_k)$, then

$$\left|D^k f(\boldsymbol{a})[\boldsymbol{d}, \cdots, \boldsymbol{d}]\right| \le M \, \|\boldsymbol{d}\|_1^k . \tag{1.6.8}$$

In general, we can define a norm (see Section 1.5.1) to measure the size of the derivatives of order $k$ given a vector norm $\|\cdot\|$:

$$\left\|D^k f(\boldsymbol{a})\right\| = \max\left\{ \left|D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \cdots, \boldsymbol{d}_k]\right| \mid \left\|\boldsymbol{d}_j\right\| \le 1 \text{ for all } j \right\}. \tag{1.6.9}$$

From this we have the bound

$$\left|D^k f(\boldsymbol{a})[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k]\right| \le \left\|D^k f(\boldsymbol{a})\right\| \, \|\boldsymbol{v}_1\| \, \|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_k\| .$$

Note that with this definition, $\|\nabla f(\boldsymbol{a})\|_2 = \left\|D^1 f(\boldsymbol{a})\right\|_2$ and $\|\text{Hess}\, f(\boldsymbol{a})\|_2 = \left\|D^2 f(\boldsymbol{a})\right\|_2$ using the ordinary vector 2-norm and matrix 2-norm, respectively.

### *1.6.3 Vector-Valued Functions*

We can apply the above results to vector-valued functions. Suppose that $\boldsymbol{f}\colon \mathbb{R}^m \to \mathbb{R}^n$. Then for each index $i = 1, 2, \ldots, m$ we have

$$f_i(\boldsymbol{a}+t\boldsymbol{d}) = f_i(\boldsymbol{a}) + D^1 f_i(\boldsymbol{a})[\boldsymbol{d}]t + \frac{1}{2}D^2 f_i(\boldsymbol{a})[\boldsymbol{d},\boldsymbol{d}]t^2 + \cdots + \frac{1}{n!}D^n f_i(\boldsymbol{a})[\underbrace{\boldsymbol{d},\boldsymbol{d},\ldots,\boldsymbol{d}}_{n \text{ times}}]t^n$$
$$+ \frac{1}{n!}\int_0^t (t-u)^n D^{n+1} f_i(\boldsymbol{a}+u\boldsymbol{d})[\underbrace{\boldsymbol{d},\boldsymbol{d},\ldots,\boldsymbol{d}}_{n+1 \text{ times}}]\,du.$$

Stacking the components $D^k f_i(\boldsymbol{a})[\boldsymbol{d}_1,\ldots,\boldsymbol{d}_k]$ we can form

$$D^k \boldsymbol{f}(\boldsymbol{a})[\boldsymbol{d}_1,\ldots,\boldsymbol{d}_k] = \begin{bmatrix} D^k f_1(\boldsymbol{a})[\boldsymbol{d}_1,\ldots,\boldsymbol{d}_k] \\ D^k f_2(\boldsymbol{a})[\boldsymbol{d}_1,\ldots,\boldsymbol{d}_k] \\ \vdots \\ D^k f_n(\boldsymbol{a})[\boldsymbol{d}_1,\ldots,\boldsymbol{d}_k] \end{bmatrix}. \tag{1.6.10}$$

With this notation,

$$\boldsymbol{f}(\boldsymbol{a}+t\boldsymbol{d}) = f_i(\boldsymbol{a}) + D^1 \boldsymbol{f}(\boldsymbol{a})[\boldsymbol{d}]t + \frac{1}{2}D^2 \boldsymbol{f}(\boldsymbol{a})[\boldsymbol{d},\boldsymbol{d}]t^2 + \cdots + \frac{1}{n!}D^n \boldsymbol{f}(\boldsymbol{a})[\underbrace{\boldsymbol{d},\boldsymbol{d},\ldots,\boldsymbol{d}}_{n \text{ times}}]t^n$$
$$+ \frac{1}{n!}\int_0^t (t-u)^n D^{n+1} \boldsymbol{f}(\boldsymbol{a}+u\boldsymbol{d})[\underbrace{\boldsymbol{d},\boldsymbol{d},\ldots,\boldsymbol{d}}_{n+1 \text{ times}}]\,du. \tag{1.6.11}$$

The mean value theorem and the integral mean value theorem do not apply to vector-valued functions. If $\boldsymbol{f}\colon \mathbb{R} \to \mathbb{R}^n$ with $n > 1$ with $\boldsymbol{f}$ differentiable, we cannot conclude that $\boldsymbol{f}(b) - \boldsymbol{f}(a) = \boldsymbol{f}'(c)(b-a)$ for some $c$ between $a$ and $b$. Take, for example, $\boldsymbol{f}(\theta) = [\cos\theta,\ \sin\theta]^T$. Then $\boldsymbol{f}(2\pi) - \boldsymbol{f}(0) = \boldsymbol{0}$ but $\boldsymbol{f}'(c) \neq \boldsymbol{0}$ for any $c$. Because of this, when we are dealing with vector-valued functions, we must use the integral form of the remainder, rather than the point form.

For vector-valued functions, we can define the norm

$$\left\| D^k f(\boldsymbol{a}) \right\| = \max \left\{ \left\| D^k f(\boldsymbol{a})[\boldsymbol{d}_1, \cdots, \boldsymbol{d}_k] \right\| \mid \|\boldsymbol{d}_j\| \le 1 \text{ for all } j \right\} \quad \text{so that} \tag{1.6.12}$$

$$\left\| D^k f(\boldsymbol{a})[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k] \right\| \le \left\| D^k f(\boldsymbol{a}) \right\| \|\boldsymbol{v}_1\| \|\boldsymbol{v}_2\| \cdots \|\boldsymbol{v}_k\| . \tag{1.6.13}$$

**Exercises.**

(1) The Taylor cubic of $\tan x$ around $x = 0$ is $x + x^3/3$. Give a bound on the maximum error for $|x| \le \pi/8$.

(2) The exponential function $e^x$ can be approximated by the Taylor polynomial of degree $n$: $e^x \approx 1 + x + x^2/2! + \cdots + x^n/n!$. What value of $n$ will guarantee that the error in this approximation is no more than $10^{-10}$ for any $x$ with $|x| \le 1/2$.

(3) The natural logarithm function $\ln(1 + x)$ has a Taylor series $x - x^2/2 + x^3/3 - \cdots + (-1)^n x^n/n + \cdots$. What is the remainder for the degree $n$ Taylor polynomial? What value of $n$ will guarantee an error of no more than $10^{-7}$ for $|x| \le 1/2$?

(4) Suppose we have the task of writing a natural logarithm function. A strategy we use is to extract the exponent field so we can use $\ln((1 + x) \times 2^e) = \ln(1 + x) + e \ln 2$, and we store $\ln 2$ to high accuracy. If we just use the exponent field we get $\frac{1}{2} < 1 + x \le 1$. What degree of Taylor polynomial will guarantee an error in $\ln(1 + x)$ of less than $10^{-16}$ for $x$ in this range?

(5) A modification of the method of the previous question is to use either the exponent field or the exponent field minus one: write $z = (1 + x) \times 2^e$ with $\frac{1}{2}a \le 1 + x \le a$ and $a > 1$ chosen to minimize the worst-case value of $|x|$. What is that value of $a$? With this value of $a$, what degree of Taylor polynomial would be needed to guarantee an error in $\ln(1 + x)$ of less than $10^{-16}$ for $x$ in this range?

(6) Let $g(x) = \exp(-1/x^2)$ for $x \ne 0$ and $g(0) = 0$. Show that the $k$th-order derivative $g^{(k)}(x) = p_k(1/x) \exp(-1/x^2)$ where $p_k$ is a polynomial. Show that $g^{(k)}(0) = 0$ for all $k$. [**Hint:** Use l'Hospital's rule.] Find the fourth-order remainder term. Give a bound on $\left|g^{(4)}(c)\right|$ for $|c| \le \frac{1}{4}$.

(7) For the function $f(x) = 1/\sqrt{1 + x^2}$, obtain the first four terms of the Taylor series of $f(1/u)$ around $u = 0$ for $u > 0$.

(8) The $\Gamma$ function is a generalization of the factorial function ($\Gamma(n + 1) = n!$ for $n = 0, 1, 2, \ldots$) given by the integral $\Gamma(n + 1) = \int_0^\infty t^n e^{-t}\, dt$. Write $\Gamma(n + 1) = \int_0^\infty \exp(-t + n \ln t)\, dt$ and set $\phi_n(t) = -t + n \ln t$. Show that for $t^* = n$, $\phi_n'(t^*) = 0$. Expand $\phi_n(t) = \phi_n(t^*) + \phi_n'(t^*)(t - t^*) + \frac{1}{2}\phi_n''(t^*)(t - t^*)^2 + \mathcal{O}((t - t^*)^3)$. Use the approximation

$$\begin{aligned}
&\Gamma(n + 1) \\
&\quad = \int_0^\infty \exp(\phi_n(t^*) + \phi_n'(t^*)(t - t^*) + \frac{1}{2}\phi_n''(t^*)(t - t^*)^2) \left[1 + \mathcal{O}((t - t^*)^3)\right] dt \\
&\quad \approx \int_{-\infty}^{+\infty} \exp(\phi_n(t^*) + \phi_n'(t^*)(t - t^*) + \frac{1}{2}\phi_n''(t^*)(t - t^*)^2)\, dt
\end{aligned}$$

to give Stirling's approximation: $n! \sim \sqrt{2\pi n}\,(n/e)^n$ as $n \to \infty$. Note that the integral can be expanded to being over $\int_{-\infty}^{+\infty}$ as this adds an exponentially small term to the value.

(9) In Exercise 8, show that

$$\exp(\phi_n(t)) = \exp(\phi_n(t^*) + \frac{1}{2}\phi_n''(t^*)(t-t^*)^2) \times \left[1 + \frac{1}{6}\phi_n'''(t^*)(t-t^*)^3 + \left(\frac{1}{24}\phi_n^{(4)}(t^*) - \frac{1}{8}\phi_n''(t^*)^2\right)(t-t^*)^4 + \mathcal{O}(t-t^*)^5\right].$$

Use this to get an improved asymptotic estimate for $n!$. Note that $\int_{-\infty}^{+\infty} \exp(a(t-t^*)^2)\,(t-t^*)^3\,dt = 0$ for any $a < 0$.

(10) Use Taylor series to estimate $\int_x^\infty \exp(-t^2/2)\,dt$: put $t = x + u$ so that $\int_x^\infty \exp(-t^2/2)\,dt = \exp(-x^2/2)\int_0^\infty \exp(-xu)\exp(-u^2/2)\,du$. Use the Taylor series expansion of $\exp(-u^2/2)$ around $u = 0$ and integrate term by term. Note that $\int_0^\infty \exp(-xu)\,u^k\,du = (k-1)!\,x^{-k-1}$. Note that integrating with the full Taylor series expansion *does not* give a sum that converges. Instead the resulting series is only asymptotic.

## Project

Create a suite of functions exp, ln, sin, and cos using Taylor series. To make this work, we need to reduce the range of the input. For example, we can use $\exp(x+y) = \exp(x)\exp(y)$ to restrict the Taylor series to evaluating $\exp(x)$ for $|x| \le \frac{1}{2}$ by computing $\exp(k) = e^k$ for integer $k$ by using repeated doubling. For natural logarithms, we use $\ln((1+x)\times 2^e) = \ln(1+x) + e\,\ln 2$. For $\sin(x)$ and $\cos(x)$ we first reduce $x$ to $|x| \le \pi$, and then use trigonometric addition rules to further reduce the range to $|x| \le \pi/4$ or even $|x| \le \pi/8$.

# Chapter 2
# Computing with Matrices and Vectors


This chapter is about numerical linear algebra, that is, matrix computations. Numerical computations with matrices (and vectors) is central to a great many algorithms, so there has been a great deal of work on this topic. We can only scratch the surface here, but you should feel free to use this as the starting point for finding the methods and analysis most appropriate for your application(s).

The three central problems in numerical linear algebra are as follows:

- Solving a linear systems of equations: solving $A\boldsymbol{x} = \boldsymbol{b}$ for $\boldsymbol{x}$.
- Minimizing the sum of squares of the errors: minimize $\sum_{i=1}^{m} (A\boldsymbol{x} - \boldsymbol{b})_i^2 = (A\boldsymbol{x} - \boldsymbol{b})^T (A\boldsymbol{x} - \boldsymbol{b})$.
- Finding eigenvalues and eigenvectors: find $\boldsymbol{x}$ and $\lambda$ where $A\boldsymbol{x} = \lambda\boldsymbol{x}$ and $\boldsymbol{x} \neq 0$.

These are not the only problems in numerical linear algebra, but they are the ones that have the most uses.

## 2.1 Solving Linear Systems

The most common operation in numerical linear algebra is solving a square system of linear equations $A\boldsymbol{x} = \boldsymbol{b}$. This is a fundamental computational task in scientific computation. If the linear systems are large, solving the linear systems can easily have the greatest computational burden of any part of a numerical computation.

On the other hand, solving a linear system can be easy. If it were not for roundoff error, we could solve a linear system exactly, provided that the matrix is square and invertible. Many students learn row echelon form for solving linear systems as undergraduates and often have solved linear systems of two equations in two unknowns in high school.

Here we must be systematic and turn techniques into pseudo-code that can be implemented.



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_2

### 2.1.1 Gaussian Elimination

Consider the linear system of equations:

$$\begin{aligned} 2x - \ y + \ z &= +6, \\ -2x + 2y - 3z &= -9, \\ 4x - \ y - \ z &= +8. \end{aligned}$$

To solve this system of equations, we perform operations on these equations, which correspond to standard row operations on the augmented matrix containing the coefficients and the right-hand side:

$$\left[\begin{array}{ccc|c} 2 & -1 & 1 & 6 \\ -2 & 2 & -3 & -9 \\ 4 & -1 & -1 & 8 \end{array}\right].$$

The standard row operations are as follows:

- swapping rows;
- multiplying a row a non-zero scalar; and
- adding a multiple of one row to another.

The last option is the most often used. So for the above linear system, we add the first row to the second, and subtract twice the first row from the third:

$$\left[\begin{array}{ccc|c} 2 & -1 & 1 & 6 \\ 0 & +1 & -2 & -3 \\ 0 & +1 & -3 & -4 \end{array}\right].$$

This eliminates the first variable ($x$) from the last two equations. We then subtract the second row from the third:

$$\left[\begin{array}{ccc|c} 2 & -1 & 1 & 6 \\ 0 & +1 & -2 & -3 \\ 0 & 0 & -1 & -1 \end{array}\right].$$

The last variable ($z$) can then be solved easily: $-z = -1$ so $z = 1$. This can be substituted into the second equation $y - 2z = -3$ and solved for $y = -1$. Finally, these results can be substituted into the first equation $2x - y + z = 6$ to give $x = (6 + y - z)/2 = 2$.

In general, we consider the coefficients of a linear system

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} & b_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} & b_n \end{array}\right].$$

We subtract multiples of the first row from the rows below in order to set the entries $a_{21}$,$a_{31}$, …, $a_{n1}$ to zero. That is, row $k$ is replaced by row $k$ minus $(a_{k1}/a_{11})$ times the first row. This results in a new augmented matrix

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & a'_{32} & a'_{33} & \cdots & a'_{3n} & b'_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & a'_{n2} & a'_{n3} & \cdots & a'_{nn} & b'_n \end{array}\right].$$

This has eliminated the first variable from all but the first equation. We can repeat the process using the second row to zero out the entries below the $(2, 2)$ entry. That is, subtract $(a'_{k2}/a'_{22})$ times the second row from row $k$ for $k = 3, 4, \ldots, n$. This gives a matrix

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & a''_{33} & \cdots & a''_{3n} & b''_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & a''_{n3} & \cdots & a''_{nn} & b''_n \end{array}\right].$$

Continuing in this way we eventually come to

$$\left[\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ 0 & a'_{22} & a'_{23} & \cdots & a'_{2n} & b'_2 \\ 0 & 0 & a''_{33} & \cdots & a''_{3n} & b''_3 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{nn}^{(n-1)} & b_n^{(n-1)} \end{array}\right].$$

The matrix of coefficients is now an upper triangular one, that is, all non-zeros are on or above the main diagonal. This system of equations is now easily solvable: $a_{nn}^{(n-1)} x_n = b_n^{(n-1)}$ gives $x_n = b_n^{(n-1)}/a_{nn}^{(n-1)}$. The second last equation is $a_{n-1,n-1}^{(n-2)} x_{n-1} + a_{n-1,n}^{(n-2)} x_n = b_{n-1}^{(n-2)}$ which can be solved for $x_{n-1}$: $x_{n-1} = (b_{n-1}^{(n-2)} - a_{n-1,n}^{(n-2)} x_n)/a_{n-1,n-1}^{(n-2)}$. This process can be repeated, working our way up the $\boldsymbol{x}$ vector until all of its entries are computed. This process is called *backward substitution*.

**Algorithm 7** Gaussian elimination; overwrites $A$ and $\boldsymbol{b}$

```
1  function GE(A, b)
2    n ← dim(b)
3    for k = 1, 2, ..., n
4      for i = k + 1, ..., n
5        m_ik ← a_ik / a_kk
6        for j = k, k + 1, ..., n
7          a_ij ← a_ij − m_ik a_kj
8        end for
9        b_i ← b_i − m_ik b_k
10     end for
11   end for
12   return (A, b)
13 end function
```

**Algorithm 8** Backward substitution; overwrites $\boldsymbol{b}$

```
1  function backsubst(U, b)
2    n ← dim(b)
3    for k = n, n − 1, ..., 2, 1
4      s ← b_k
5      for j = k + 1, ..., n
6        s ← s − u_kj b_j
7      end for
8      b_k ← s / u_kk
9    end for
10   return b // returns solution
11 end function
```

Pseudo-code for the elimination process, called *Gaussian elimination*, is in Algorithm 7. This code for Gaussian elimination overwrites $A$ and the right-hand side vector $\boldsymbol{b}$.

The pseudo-code for backward substitution shown in Algorithm 8 overwrites the right-hand side $\boldsymbol{b}$ with the solution $\boldsymbol{x}$ for $U\boldsymbol{x} = \boldsymbol{b}$ where $U$ is upper triangular.

The computational cost of these methods can be determined in terms of the numbers of floating point operations ($+$, $-$, $\times$, $/$). These numbers are only a proxy for the total time taken, even when you factor in the rating for the number of floating point operations per second for a given computer. Since additions, subtractions and multiplication typically can be done at a rate of one per clock cycle, data movement costs are often much more important, and processor dependent. Division operations typically require 10–15 clock cycles each.

Nevertheless, counting floating point operations gives us a good sense as to the asymptotic time taken by the algorithms at least for large $n$. For Gaussian elimination, lines 4–10 require $2(n-k)(n-k+1)+3$ flops for each iteration over $k$. The total number of floating point operations for Gaussian elimination is therefore

$$\sum_{k=1}^{n-1} (2(n-k)(n-k+1)+3) \sim \frac{2}{3}n^3 \text{ flops.}$$

By comparison, the number of flops required by backward substitution is

$$\sum_{k=1}^{n} (2(n-k)+1) = n^2 \text{ flops.}$$

Clearly, the more expensive part for large $n$ is Gaussian elimination.

### 2.1.2 LU Factorization

The process of Gaussian elimination for the entries of the matrix $A$ does not depend on $\boldsymbol{b}$, although the changes to $\boldsymbol{b}$ do depend on the entries in $A$. We can separate the two parts of this process. This leads us to the LU factorization, the most common method for solving small-to-moderate systems of linear equations. The main difference between Gaussian elimination and LU factorization is just saving the multipliers $m_{ik}$. We can put the multipliers into a matrix

$$L = \begin{bmatrix} 1 & & & & \\ m_{21} & 1 & & & \\ m_{31} & m_{32} & 1 & & \\ \vdots & \vdots & \vdots & \ddots & \\ m_{n1} & m_{n2} & m_{n3} & \cdots & 1 \end{bmatrix}.$$

The remarkable property of this matrix is that $LU = A$ where $U$ is the upper triangular matrix remaining after $A$ is overwritten in Gaussian elimination, that is, storing the multipliers enables us to reconstruct the matrix $A$. While it can be somewhat difficult to see this directly, there is a recursive version of the LU factorization that makes this easier to see.

Write

$$L = \begin{bmatrix} 1 & \\ \boldsymbol{m} & \widetilde{L} \end{bmatrix}, \qquad A = \begin{bmatrix} \alpha & \boldsymbol{r}^T \\ \boldsymbol{c} & \widetilde{A} \end{bmatrix} \qquad (n \times n).$$

Note that $\widetilde{L}$ is also lower triangular (that is, all non-zeros occur on or below the main diagonal). Note that the first row of $A$ is not changed by Gaussian elimination. The remaining upper triangular matrix after elimination is

$$U = \begin{bmatrix} \alpha & \boldsymbol{r}^T \\ & \widetilde{U} \end{bmatrix},$$

where $\widetilde{U}$ is also upper triangular. Note that the multipliers in Gaussian elimination are obtained by setting $\boldsymbol{m}$ to $\boldsymbol{c}/\alpha$ (since $\alpha = a_{11}$ and $m_{i1} \leftarrow a_{i1}/a_{11}$). The effect of the first stage of Gaussian elimination (for $k = 1$) is to replace $a_{ij} \leftarrow a_{ij} - m_{i1}a_{1,j}$ for $i, j > 1$. This corresponds to replacing $\widetilde{A}$ with $\widetilde{A}' = \widetilde{A} - \boldsymbol{m}\boldsymbol{r}^T$. Under our implicit induction hypothesis, Gaussian elimination to $\widetilde{A}'$ is equivalent to factoring $\widetilde{A}' = \widetilde{L}\widetilde{U}$.

**Algorithm 9** Forward substitution; overwrites $\boldsymbol{b}$

```
1  function forwardsubst(L, b)
2    n ← dim(b)
3    for k = 1, 2, ..., n
4      s ← b_k
5      for j = 1, 2, ..., k − 1
6        s ← s − ℓ_kj b_j
7      end for
8      b_k ← s/ℓ_kk
9    end for
10   return b // returns solution
11 end function
```

**Algorithm 10** LU factorization; overwrites $A$ with $U$

```
1  function LU(A)
2    n ← num_rows(A); L ← I
3    for k = 1, 2, ..., n
4      for i = k + 1, ..., n
5        m_ik ← a_ik/a_kk
6        for j = k, k + 1, ..., n
7          a_ij ← a_ij − m_ik a_kj
8        end for
9      end for
10   end for
11   ℓ_ik ← m_ik
12   return (L, A)
13 end function
```

Then

$$LU = \begin{bmatrix} 1 & \\ \boldsymbol{m} & \widetilde{L} \end{bmatrix} \begin{bmatrix} \alpha & \boldsymbol{r}^T \\ & \widetilde{U} \end{bmatrix} = \left[\begin{array}{c|c} \alpha & \boldsymbol{r}^T \\ \hline \boldsymbol{m}\alpha & \widetilde{L}\widetilde{U} + \boldsymbol{m}\boldsymbol{r}^T \end{array}\right] = \left[\begin{array}{c|c} \alpha & \boldsymbol{r}^T \\ \hline \boldsymbol{c} & \widetilde{A}' + \boldsymbol{m}\boldsymbol{r}^T \end{array}\right] = \begin{bmatrix} \alpha & \boldsymbol{r}^T \\ \boldsymbol{c} & \widetilde{A} \end{bmatrix} = A$$

and we have reconstructed $A$. Although we have not gone through a formal proof by induction, we see that we do indeed have $A = LU$, which is the LU factorization.

The strategy of LU factorization then is to ignore $\boldsymbol{b}$ in the elimination process, but save the multipliers in $L$. To compensate for ignoring $\boldsymbol{b}$ in the elimination, when we need to solve $A\boldsymbol{x} = \boldsymbol{b}$ we carry out two triangular solves: $L(U\boldsymbol{x}) = \boldsymbol{b}$. Setting $\boldsymbol{z} = U\boldsymbol{x}$ we first solve $L\boldsymbol{z} = \boldsymbol{b}$ for $\boldsymbol{z}$. Then we solve $U\boldsymbol{x} = \boldsymbol{z}$ for $\boldsymbol{x}$:

solve $L\boldsymbol{z} = \boldsymbol{b}$ for $\boldsymbol{z}$
solve $U\boldsymbol{x} = \boldsymbol{z}$ for $\boldsymbol{x}$.

To solve $L\boldsymbol{z} = \boldsymbol{b}$ for $\boldsymbol{z}$ we use a variant of backward substitution called *forward substitution*, as shown in Algorithm 9.

The cost of forward substitution in terms of floating point operations is identical to that of backward substitution. For the version of LU factorization developed here,

the diagonal entries $\ell_{kk} = 1$, so that a division could be removed for each $k$. This gives a small reduction in the cost of forward substitution.

Here we give a non-recursive pseudo-code for LU factorization in Algorithm 10.

Often $A$ is overwritten with both $L$ and $U$. Since the diagonal entries of $L$ are all ones, they do not need to be stored. Instead we can store the multipliers in the strictly lower triangular part of $A$, that is, store $m_{ik} = \ell_{ik}$ for $i > k$ in $a_{ik}$. At the end of the LU factorization, the overwritten $A$ contains

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ \ell_{21} & u_{22} & u_{23} & \cdots & u_{2n} \\ \ell_{31} & \ell_{32} & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{2n} & \ell_{3n} & \cdots & u_{nn} \end{bmatrix}.$$

### 2.1.3 Errors in Solving Linear Systems

If we could carry out LU factorization with forward and backward substitution with exact arithmetic we could solve linear systems exactly. However, we still have the effect of roundoff error. This can affect the construction of $A$ and $\boldsymbol{b}$ as well as the solution process. Further, it is worth understanding how errors in $A$ and $\boldsymbol{b}$ produce errors in the computed solution $\boldsymbol{x}$. These errors may come from other sources, such as measurement error and approximations in other numerical procedures.

To understand errors in solving linear systems, we separate the issue of errors in the original data $A$ and $\boldsymbol{b}$ from the issue of errors generated by the solution process. To deal primarily with errors in the original data, we look to the *perturbation theorem for linear systems* (Theorem 2.1). To deal with errors generated by the solution process, we look to the *backward error analysis* originally due to James Wilkinson (Theorem 2.5).

#### 2.1.3.1 Perturbation Theorem for Linear Systems

**Theorem 2.1** *(Perturbation theorem for linear systems) Suppose $A$ is invertible and*

$$A\boldsymbol{x} = \boldsymbol{b},$$
$$(A + E)\widehat{\boldsymbol{x}} = \boldsymbol{b} + \boldsymbol{d}.$$

*Then*

$$\frac{\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\kappa(A)}{1 - \kappa(A)(\|E\| / \|A\|)} \left[ \frac{\|E\|}{\|A\|} + \frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|} \right], \tag{2.1.1}$$

*where $\kappa(A) = \|A\| \left\|A^{-1}\right\|$ provided the denominator is positive.*

We can consider the quantities $\|\boldsymbol{d}\| / \|\boldsymbol{b}\|$ and $\|E\| / \|A\|$ to be the relative errors in the right-hand side and the matrix. The quantity $\kappa(A) = \|A\| \left\|A^{-1}\right\|$ is called the *condition number* of $A$ and represents in a general sense how much relative errors in the data are amplified by solving the system of equations. This theorem says nothing about solution process or roundoff error.

Before we prove the perturbation theorem, we need a basic result in inverses on perturbations of the identity matrix.

**Lemma 2.2** *If* $\|F\| < 1$ *then* $(I + F)^{-1} = I - F + F^2 - F^3 + \cdots$, $\left\|(I + F)^{-1}\right\| \leq 1/(1 - \|F\|)$, *and* $\left\|(I + F)^{-1} - I\right\| \leq \|F\| / (1 - \|F\|)$.

***Proof*** (Of Lemma 2.2). Suppose that $\|F\| < 1$. By induction using properties of matrix norms, we can see that $\left\|F^k\right\| = \left\|F\, F^{k-1}\right\| \leq \|F\| \left\|F^{k-1}\right\| \leq \cdots \leq \|F\|^k$. The infinite matrix series $I - F + F^2 - F^3 + \cdots$ converges as its partial sums are a Cauchy sequence: if $M > N$ then

$$\begin{aligned}
&\left\| \sum_{k=0}^{M} (-F)^k - \sum_{k=0}^{N} (-F)^k \right\| \\
&\quad = \left\| \sum_{k=N+1}^{M} (-F)^k \right\| \leq \sum_{k=N+1}^{M} \|F\|^k \\
&\quad = \|F\|^{N+1} \frac{1 - \|F\|^{M-N-1}}{1 - \|F\|} < \frac{\|F\|^{N+1}}{1 - \|F\|}
\end{aligned}$$

which goes to zero as $N \to \infty$. To show that

$$(I + F)^{-1} = I - F + F^2 - F^3 + \cdots,$$

we note that

$$(I + F) \sum_{k=0}^{N} (-F)^k = I - (-F)^{N+1}.$$

Thus, taking limits as $N \to \infty$ gives

$$(I + F) \sum_{k=0}^{\infty} (-F)^k = I$$

and thus $(I + F)^{-1} = \sum_{k=0}^{\infty} (-F)^k$.

To obtain the bounds we note that

$$\left\|(I+F)^{-1}\right\| = \left\|\sum_{k=0}^{\infty}(-F)^k\right\| \le \sum_{k=0}^{\infty}\|F\|^k = \frac{1}{1-\|F\|}, \quad \text{and}$$
$$\left\|(I+F)^{-1} - I\right\| = \left\|F(I+F)^{-1}\right\| \le \frac{\|F\|}{1-\|F\|},$$

as we wanted. □

With this Lemma ready, we can start the main course.

***Proof*** (*Of Theorem* 2.1). Premultiply $(A+E)\widehat{\boldsymbol{x}} = \boldsymbol{b}+\boldsymbol{d}$ by $A^{-1}$:

$$\begin{aligned}(I+A^{-1}E)\widehat{\boldsymbol{x}} &= A^{-1}(\boldsymbol{b}+\boldsymbol{d}) = A^{-1}\boldsymbol{b} + A^{-1}\boldsymbol{d}\\ &= \boldsymbol{x} + A^{-1}\boldsymbol{d}\\ &= (I+A^{-1}E)\boldsymbol{x} - A^{-1}E\boldsymbol{x} + A^{-1}\boldsymbol{d}\\ &= (I+A^{-1}E)\boldsymbol{x} + A^{-1}(\boldsymbol{d}-E\boldsymbol{x}).\end{aligned} \tag{2.1.2}$$

The matrix $I + A^{-1}E$ is invertible:

$$\left\|A^{-1}E\right\| \le \left\|A^{-1}\right\|\|E\| = \kappa(A)\frac{\|E\|}{\|A\|} < 1,$$

so $I + A^{-1}E$ is invertible by Lemma 2.2. Pre-multiplying (2.1.2) by $(I+A^{-1}E)^{-1}$ and subtracting $\boldsymbol{x}$, we get

$$\widehat{\boldsymbol{x}} - \boldsymbol{x} = (I+A^{-1}E)^{-1}A^{-1}(\boldsymbol{d}-E\boldsymbol{x}).$$

Taking norms:

$$\begin{aligned}\|\widehat{\boldsymbol{x}}-\boldsymbol{x}\| &= \left\|(I+A^{-1}E)^{-1}A^{-1}(\boldsymbol{d}-E\boldsymbol{x})\right\|\\ &\le \left\|(I+A^{-1}E)^{-1}\right\|\left\|A^{-1}\right\|\|\boldsymbol{d}-E\boldsymbol{x}\|\\ &\le \left\|(I+A^{-1}E)^{-1}\right\|\left\|A^{-1}\right\|(\|\boldsymbol{d}\|+\|E\|\,\|\boldsymbol{x}\|)\\ &\le \frac{\left\|A^{-1}\right\|}{1-\left\|A^{-1}E\right\|}\left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|}\|\boldsymbol{b}\| + \|E\|\,\|\boldsymbol{x}\|\right]\\ &\le \frac{\left\|A^{-1}\right\|}{1-\left\|A^{-1}E\right\|}\left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|}\|A\boldsymbol{x}\| + \|E\|\,\|\boldsymbol{x}\|\right]\\ &\le \frac{\left\|A^{-1}\right\|}{1-\left\|A^{-1}E\right\|}\left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|}\|A\|\,\|\boldsymbol{x}\| + \frac{\|E\|}{\|A\|}\|A\|\,\|\boldsymbol{x}\|\right]\\ &\le \frac{\left\|A^{-1}\right\|\|A\|\,\|\boldsymbol{x}\|}{1-\left\|A^{-1}E\right\|}\left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|} + \frac{\|E\|}{\|A\|}\right].\end{aligned}$$

Then

$$\frac{\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \frac{\left\|A^{-1}\right\| \|A\|}{1 - \left\|A^{-1}E\right\|} \left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|} + \frac{\|E\|}{\|A\|}\right].$$

Note that $\|A\| \left\|A^{-1}\right\| = \kappa(A)$, the condition number of $A$ in the given matrix norm. Also,

$$\left\|A^{-1}E\right\| \leq \left\|A^{-1}\right\| \|E\| = \left\|A^{-1}\right\| \|A\| \left(\|E\| / \|A\|\right) = \kappa(A) \left(\|E\| / \|A\|\right).$$

So provided $\kappa(A) \left(\|E\| / \|A\|\right) < 1$, we have

$$1 - \left\|A^{-1}E\right\| \geq 1 - \kappa(A) \left(\|E\| / \|A\|\right), \text{ and}$$
$$\frac{1}{1 - \left\|A^{-1}E\right\|} \leq \frac{1}{1 - \kappa(A) \left(\|E\| / \|A\|\right)}.$$

That is,

$$\begin{aligned}\frac{\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|}{\|\boldsymbol{x}\|} &\leq \frac{\left\|A^{-1}\right\| \|A\|}{1 - \left\|A^{-1}E\right\|} \left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|} + \frac{\|E\|}{\|A\|}\right] \\ &\leq \frac{\kappa(A)}{1 - \kappa(A) \left(\|E\| / \|A\|\right)} \left[\frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|} + \frac{\|E\|}{\|A\|}\right]\end{aligned}$$

as we wanted. □

How can we use Theorem 2.1?

First we need to note that $\kappa(A) = \|A\| \left\|A^{-1}\right\| \geq \left\|AA^{-1}\right\| = \|I\|$. Note that $\|I\| = 1$ in any induced norm; even in a matrix norm that is not induced, $\|I\| = \|I\,I\| \leq \|I\| \|I\| = \|I\|^2$ so $1 \leq \|I\|$. In either case, $\kappa(A) \geq 1$.

If $\kappa(A) \left(\|E\| / \|A\|\right) \ll 1$ then the denominator $1 - \kappa(A) \left(\|E\| / \|A\|\right) \approx 1$ and the relative errors in the data $\|\boldsymbol{d}\| / \|\boldsymbol{b}\|$ and $\|E\| / \|A\|$ are amplified by a factor of close to $\kappa(A)$. The bound on the relative error in the solution $(\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\| / \|\boldsymbol{x}\|)$ can never be reduced by a small $\kappa(A)$. Instead, we aim to prevent $\kappa(A)$ from becoming "too large". How large is "too large" depends on the application, but we definitely want to keep $\kappa(A)$ from becoming larger than 1/(unit roundoff), as then any roundoff error in the data or solution process can completely destroy any accuracy in the solution.

#### 2.1.3.2 Wilkinson's Backward Error Analysis

In the years 1947–1948, there were two articles about the error in numerical solution of linear systems of equations by Gaussian elimination or LU factorization with floating point arithmetic: one by John von Neumann and H. Goldstine [109, 256] and the other by Alan Turing [249]. The conclusions of these articles tended to be

limited in scope, or even pessimistic, about the accuracy of the computed solutions. A few years later, the English mathematician James H. Wilkinson [261] discovered an example of LU factorization in his work which had lost 5 digits of accuracy, but the reconstruction error $A - \widehat{L}\widehat{U}$ showed more than 10 digits of accuracy, where $\widehat{L}$ and $\widehat{U}$ are the computed $L$ and $U$ matrices. A more modern example of this phenomenon can be seen in the following example:

$n \leftarrow 20$
$A \leftarrow [1/(i+j-1) \mid i, j = 1, 2, \ldots, n]$ `// n × n Hilbert matrix`
$\widehat{\boldsymbol{x}} \leftarrow$ *random n–dimensional vector*
$\boldsymbol{b} \leftarrow A\widehat{\boldsymbol{x}}$
`solve` $A\boldsymbol{x} = \boldsymbol{b}$ `for` $\boldsymbol{x}$
`print` $\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|$ `// can be big . . .`
`print` $\|A\boldsymbol{x} - \boldsymbol{b}\|$ `// always small!`

As above, we let $\widehat{L}$, $\widehat{U}$ denote the computed $L$ and $U$ factors in the LU factorization of $A$.

Wilkinson's method of analysis shows that the computed $L$ and $U$ factors as well as the computed solution $\widehat{\boldsymbol{x}}$ are, in fact, the *exact* solutions for nearby systems : $\left(A + E'\right)\widehat{\boldsymbol{x}} = \boldsymbol{b}$ and $A + E = \widehat{L}\widehat{U}$, with $E$ and $E'$ "small".

The basis of these results is the following model of floating point arithmetic:

$$fl(x \circ y) = (x \circ y)\,(1 + \epsilon)\,, \qquad |\epsilon| \leq \mathbf{u}$$

for the arithmetic operations $\circ = +,\ -,\ \times,\ /$. Here $fl$(expression) denotes the result of the expression with floating point operations. Also $\mathbf{u} > 0$ is the unit roundoff of the floating point system. For IEEE double precision, $\mathbf{u} \approx 2.2 \times 10^{-16}$.

The approach taken here is based on that of Higham (see, for example, [125]), which in turn is based on an analysis in the textbook of Stoer and Bulirsch [240] for LU factorization. To start, let

$$\gamma_n = \frac{n\mathbf{u}}{1 - n\mathbf{u}}, \qquad \text{if } n\mathbf{u} < 1. \tag{2.1.3}$$

The important properties of $\gamma_n$ are that $1 + \mathbf{u} < (1 - \mathbf{u})^{-1} \leq 1 + \gamma_1$ and $(1 + \gamma_k)(1 + \gamma_\ell) \leq 1 + \gamma_{k+\ell}$ for $k, \ell = 1, 2, 3, \ldots$. Note that an alternative formula for $\gamma_n$ that has these properties is $\gamma_n = \exp(n\mathbf{u}(1 + \mathbf{u})) - 1$ provided $\mathbf{u} \leq 1/2$.

**Lemma 2.3** *If* $|\delta_i| \leq \mathbf{u}$ *and* $p_i = \pm 1$*, for all* $i$ *with* $n\mathbf{u} < 1$*, then*

$$\prod_{i=1}^{n} (1 + \delta_i)^{p_i} = 1 + \theta_n, \qquad |\theta_n| \leq \gamma_n. \tag{2.1.4}$$

***Proof*** To start mathematical induction, note that condition (2.1.4) is clearly true for $n = 1$. Assume that (2.1.4) is true for $n = k$. We show that it is true for $n = k + 1$. Note that $\gamma_k + \gamma_\ell + \gamma_k\gamma_\ell \leq \gamma_{k+\ell}$. Consider

$$\begin{aligned}\prod_{i=1}^{k+1}(1+\delta_i)^{p_i} &= \left[\prod_{i=1}^{k}(1+\delta_i)^{p_i}\right](1+\delta_{k+1})^{p_{k+1}} \\ &= (1+\theta_k)\,(1+\delta_{k+1})^{p_{k+1}}\,.\end{aligned}$$

If $p_{k+1} = +1$, $\theta_{k+1} = \theta_k + \delta_{k+1} + \theta_k\delta_{k+1}$ and so

$$\begin{aligned}|\theta_{k+1}| &\leq |\theta_k| + |\delta_{k+1}| + |\delta_{k+1}|\,|\theta_k| \\ &\leq \gamma_k + \gamma_1 + \gamma_k\gamma_1 \leq \gamma_{k+1}.\end{aligned}$$

If $p_{k+1} = -1$, $\theta_{k+1} = \theta_k + \delta_{k+1}\,(1+\theta_k)\,/\,(1+\delta_k)$ and then

$$\begin{aligned}|\theta_{k+1}| &\leq |\theta_k| + \frac{|\delta_{k+1}|\,(1+|\theta_k|)}{1-|\delta_{k+1}|} \\ &\leq \gamma_k + \gamma_1(1+\gamma_k) \leq \gamma_{k+1}.\end{aligned}$$

Thus, by the principle of mathematical induction, $|\theta_n| \leq \gamma_n$ for all $n$ with $n\mathbf{u} < 1$. □

We can use this lemma to produce another lemma which treats common calculations in numerical linear algebra.

**Lemma 2.4** *If in an algorithm, $s \leftarrow \left(c - \sum_{i=1}^{k-1} a_i b_i\right)/b_k$ is evaluated in the common order by correctly rounded floating point arithmetic, then the computed value of $s$ (denoted $\widehat{s}$) satisfies the following inequality:*

$$\left|c - \sum_{i=1}^{k-1} a_i b_i - \widehat{s}b_k\right| \leq \gamma_k\left(|\widehat{s}b_k| + \sum_{i=1}^{k-1}|a_i b_i|\right).$$

***Proof*** Consider the algorithm

$s_0 \leftarrow c$
`for` $i = 1,\ 2,\ \ldots,\ k-1$
  $s_i \leftarrow s_{i-1} - a_i b_i$
`end for`
$s_k \leftarrow s_{k-1}/b_k$.

The computed value of $s_i$ $(i = 1, 2, \ldots, k-1)$ is

$$\begin{aligned}\widehat{s_i} &= fl(\widehat{s_{i-1}} - a_i b_i) \\ &= (\widehat{s_{i-1}} - fl(a_i b_i))\,(1+\delta_i) \\ &= (\widehat{s_{i-1}} - a_i b_i\,(1+\epsilon_i))\,(1+\delta_i)\end{aligned}$$

with $|\delta_i|\,,\ |\epsilon_i| \leq \mathbf{u}$. Thus

$$\widehat{s}_{k-1} = c\prod_{i=1}^{k-1}(1+\delta_i) - \sum_{i=1}^{k-1} a_i b_i\,(1+\epsilon_i)\prod_{j=i+1}^{k-1}\left(1+\delta_j\right).$$

Finally,

$$\begin{aligned}\widehat{s_k} &= fl\,(\widehat{s}_{k-1}/b_k) \\ &= (\widehat{s}_{k-1}/b_k)\,(1+\delta_k)\end{aligned}$$

with $|\delta_k| \leq \mathbf{u}$. Because of this,

$$\begin{aligned}\widehat{s}_k b_k &= \widehat{s}_{k-1}\,(1+\delta_k) \\ &= c\prod_{i=1}^{k}(1+\delta_i) - \sum_{i=1}^{k-1} a_i b_i\,(1+\epsilon_i)\prod_{j=i+1}^{k}\left(1+\delta_j\right).\end{aligned}$$

Multiplying by $1+\theta_k = \prod_{i=1}^{k}(1+\delta_i)^{-1}$ produces

$$\begin{aligned}\widehat{s}b_k\,(1+\theta_k) &= c - \sum_{i=1}^{k-1} a_i b_i\,(1+\epsilon_i)\prod_{j=1}^{i}\left(1+\delta_j\right)^{-1} \\ &= c - \sum_{i=1}^{k-1} a_i b_i\,(1+\theta_i)\end{aligned}$$

with $|\theta_i| \leq \gamma_i \leq \gamma_k$ for all $i \leq k$. Re-arranging then gives the desired conclusion. □

Now we are ready for the proof of Wilkinson's theorem. One of the important implications of this result is that we should try to avoid having large entries in $\widehat{L}$ and $\widehat{U}$ matrices.

**Theorem 2.5** *(Wilkinson's backward error result.) Suppose that $A$ is an $n \times n$ matrix and $\boldsymbol{b}$ an $n$-dimensional vector of floating point numbers. Further suppose that $\widehat{L}$ and $\widehat{U}$ are the computed matrices for the LU factorization of $A$, and $\widehat{\boldsymbol{x}}$ is the computed vector for the solution of $A\boldsymbol{x} = \boldsymbol{b}$ through $\widehat{L}$, $\widehat{U}$ and forward and backward substitution. Then*

$$A + E' = \widehat{L}\widehat{U}, \qquad \left|E'\right| \le \gamma_n \left|\widehat{L}\right| \left|\widehat{U}\right|,$$
$$(A + E)\,\widehat{\boldsymbol{x}} = \boldsymbol{b}, \qquad |E| \le \gamma_n\,(3 + \gamma_n) \left|\widehat{L}\right| \left|\widehat{U}\right|,$$

*where $|B|$ is the matrix with entries $|B|_{ij} = \left|b_{ij}\right|$ being the absolute values of the entries of $B$.*

***Proof*** Algorithm 10 for the LU factorization is equivalent to the computations (including roundoff errors):

$$u_{ij} \leftarrow a_{ij} - \sum_{k=1}^{i-1} \ell_{ik} u_{kj}, \qquad j \ge i,$$
$$\ell_{ij} \leftarrow \left(a_{ij} - \sum_{k=1}^{j-1} \ell_{ik} u_{kj}\right) / u_{jj}, \qquad j < i$$

with evaluation of the sums in the natural order. Recall that $\ell_{ii} = \widehat{\ell}_{ii} = 1$. From the previous lemma,

$$\left|a_{ij} - \sum_{k=1}^{i-1} \widehat{\ell}_{ik}\widehat{u}_{kj} - \widehat{\ell}_{ii}\widehat{u}_{ij}\right| \le \gamma_i \sum_{k=1}^{i} \left|\widehat{\ell}_{ik}\right| \left|\widehat{u}_{kj}\right|, \qquad j \ge i,$$
$$\left|a_{ij} - \sum_{k=1}^{j} \widehat{\ell}_{ik}\widehat{u}_{kj}\right| \le \gamma_j \sum_{k=1}^{j} \left|\widehat{\ell}_{ik}\right| \left|\widehat{u}_{kj}\right|, \qquad j < i.$$

Then

$$\left|A - \widehat{L}\widehat{U}\right| \le \gamma_n \left|\widehat{L}\right| \left|\widehat{U}\right|,$$

or equivalently

$$A + E' = \widehat{L}\widehat{U}, \qquad \left|E'\right| \le \gamma_n \left|\widehat{L}\right| \left|\widehat{U}\right|.$$

Using forward and backward substitution, we solve $\widehat{L}\boldsymbol{y} = \boldsymbol{b}$ (with result $\widehat{\boldsymbol{y}}$) and $\widehat{U}\boldsymbol{x} = \widehat{\boldsymbol{y}}$ (with result $\widehat{\boldsymbol{x}}$). These equations are equivalent to

$$y_i + \sum_{j=1}^{i-1} \widehat{\ell}_{ij} y_j = b_i, \qquad \text{and}$$
$$\widehat{u}_{ii} x_i + \sum_{j=i+1}^{n} \widehat{u}_{ij} x_j = \widehat{y}_i.$$

Following the previous lemma, we have

$$\left| b_i - \sum_{j=1}^{i-1} \widehat{\ell}_{ij} \widehat{y}_j \right| \leq \gamma_n \sum_{j=1}^{i} \left| \widehat{\ell}_{ij} \right| \left| \widehat{y}_j \right| \qquad \text{for all } i,$$

$$\left| \widehat{y}_i - \sum_{j=i}^{n} \widehat{u}_{ij} \widehat{x}_j \right| \leq \gamma_n \sum_{j=i}^{n} \left| \widehat{u}_{ij} \right| \left| \widehat{x}_j \right| \qquad \text{for all } i.$$

Then, we have $\delta \ell_{ij}$ and $\delta u_{ij}$ such that

$$b_i = \sum_{j=1}^{i} \left( \widehat{\ell}_{ij} + \delta \ell_{ij} \right) \widehat{y}_j, \qquad \left| \delta \ell_{ij} \right| \leq \gamma_n \left| \widehat{\ell}_{ij} \right|,$$

$$\widehat{y}_i = \sum_{j=i}^{n} \left( \widehat{u}_{ij} + \delta u_{ij} \right) \widehat{x}_j, \qquad \left| \delta u_{ij} \right| \leq \gamma_n \left| \widehat{u}_{ij} \right|,$$

for all $i$ and $j$. Expressed in terms of the matrices $\widehat{L}$ and $\widehat{U}$, $\left( \widehat{L} + \delta L \right) \widehat{\boldsymbol{y}} = \boldsymbol{b}$ and $\left( \widehat{U} + \delta U \right) \widehat{\boldsymbol{x}} = \widehat{\boldsymbol{y}}$ with $|\delta L| \leq \gamma_n \left| \widehat{L} \right|$ and $|\delta U| \leq \gamma_n \left| \widehat{U} \right|$; so

$$\left( \widehat{L} + \delta L \right) \left( \widehat{U} + \delta U \right) \widehat{\boldsymbol{x}} = \boldsymbol{b}.$$

Thus, we have $(A + E) \widehat{\boldsymbol{x}} = \boldsymbol{b}$ with $E = \left( \widehat{L} \widehat{U} - A \right) + \delta L \, \widehat{U} + \widehat{L} \, \delta U + \delta L \, \delta U$. We have already shown that $\left| \widehat{L} \widehat{U} - A \right| \leq \gamma_n \left| \widehat{L} \right| \left| \widehat{U} \right|$, $|\delta L| \leq \gamma_n \left| \widehat{L} \right|$ and $|\delta U| \leq \gamma_n \left| \widehat{U} \right|$. Therefore

$$\begin{aligned} |E| &\leq 3 \gamma_n \left| \widehat{L} \right| \left| \widehat{U} \right| + \gamma_n^2 \left| \widehat{L} \right| \left| \widehat{U} \right| \\ &= \gamma_n \left( 3 + \gamma_n \right) \left| \widehat{L} \right| \left| \widehat{U} \right|, \end{aligned}$$

as required. □

### 2.1.4 Pivoting and $PA = LU$

Wilkinson's backward error analysis showed the importance of keeping the size of the factor matrices from growing excessively large. However, LU factorization does not in itself guarantee that. The weakness comes from the division on line 5 of Algorithm 10: $m_{ik} \leftarrow a_{ik}/a_{kk}$. If $a_{kk} = 0$ then the algorithm fails. If $a_{kk} \approx 0$ then the algorithm can complete, but we may have very large entries in $L$, and therefore also in $U$. This can cause numerical problems.

The LU factorization will fail for

$$A_0 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

since for $k = 1$ we have $a_{11} = 0$, and we get division by zero. But this matrix is invertible. Its inverse is $\begin{bmatrix} -1 & 1 \\ 1 & 0 \end{bmatrix}$. The condition number $\kappa_\infty(A_0) = \|A_0\|_\infty \left\|A_0^{-1}\right\|_\infty = 2 \times 2 = 4$ which is not large. We could replace this zero entry with something small, call it $\epsilon \neq 0$. The perturbation theorem for linear systems (Theorem 2.1) indicates this would only result in an error of size $\approx 2\epsilon$ using the $\infty$-norm. Let us take $\epsilon$ to be a power of two, but smaller than unit roundoff. Taking $\epsilon$ to be a power of two ensures that no roundoff error will occur in multiplication or division with $\epsilon$.

Then computing the LU factorization of

$$A_\epsilon = \left[\begin{array}{c|c} \epsilon & 1 \\ \hline 1 & 1 \end{array}\right]$$

in exact arithmetic will result in

$$L = \left[\begin{array}{c|c} 1 & \\ \hline 1/\epsilon & 1 \end{array}\right], \qquad U = \left[\begin{array}{c|c} \epsilon & 1 \\ \hline & 1 - (1/\epsilon) \end{array}\right].$$

Unfortunately, since $\epsilon < \mathbf{u}$ (unit roundoff), the computed value of $1 - (1/\epsilon)$ is actually $-1/\epsilon$ as shifting the "1" to align the exponents results in the mantissa being completely shifted off. So the computed $L$ and $U$ are

$$\widehat{L} = \left[\begin{array}{c|c} 1 & \\ \hline (1/\epsilon) & 1 \end{array}\right], \qquad \widehat{U} = \left[\begin{array}{c|c} \epsilon & 1 \\ \hline & (-1/\epsilon) \end{array}\right].$$

These are not close to a factorization of $A_\epsilon$ or $A_0$: $\widehat{L}\widehat{U} = \left[\begin{array}{c|c} \epsilon & 1 \\ \hline 1 & 0 \end{array}\right]$. Why does this occur? Wilkinson's backward error analysis (Theorem 2.5) shows that the error in the reconstruction of $A_\epsilon$ is bounded by approximately $n\mathbf{u}\left(|A| + \left|\widehat{L}\right|\left|\widehat{U}\right|\right)$ with $n = 2$. But, in this case, $\left|\widehat{L}\right|\left|\widehat{U}\right| = \left[\begin{array}{c|c} \epsilon & 1 \\ \hline 1 & (2/\epsilon) \end{array}\right]$ and the $(2, 2)$ entry is very large.

We need to avoid these large values by avoiding division by (relatively) small pivot entries. We can do this by using row swaps, even if $a_{kk} \neq 0$. The result is not quite an LU factorization. Instead we create a permuted LU factorization:

$$PA = LU, \tag{2.1.5}$$

where $P$ is a *permutation matrix*.

#### 2.1.4.1 Permutation Matrices

A permutation matrix is the identity matrix with the rows shuffled. As a result, every entry is either zero or one; every column has exactly one entry that is one; and every row has exactly one entry that is one. Examples include the following:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Permutation matrices can be represented efficiently in memory using simple arrays of integers. The above examples would be represented by

$$[1, 2], \quad [2, 1, 3], \quad [3, 1, 2, 4].$$

An array $[\pi(1), \pi(2), \ldots, \pi(n)]$ represents the permutation matrix $P$ where $P\boldsymbol{e}_j = \boldsymbol{e}_{\pi(j)}$, $j = 1, 2, \ldots, n$. Note that every integer from one to $n$ is listed in the array exactly once, so that $\pi$ is a permutation of $\{1, 2, \ldots, n\}$.

Permutation matrices $P$ shuffle the entries of a vector, so that $P\boldsymbol{x}$ has the same entries as $\boldsymbol{x}$, but in a different order. This means that $\|P\boldsymbol{x}\|_p = \|\boldsymbol{x}\|_p$ for any $1 \leq p \leq \infty$. In particular, we can take $p = 2$, so that $\|P\boldsymbol{x}\|_2^2 = \|\boldsymbol{x}\|_2^2$ and $\boldsymbol{x}^T P^T P\boldsymbol{x} = \boldsymbol{x}^T I\boldsymbol{x}$ for all $\boldsymbol{x}$. Thus $P^T P = I$ and $P^T = P^{-1}$. That is, permutation matrices are orthogonal.

The vector array subscripting features of MATLAB, Julia, and R enable us to apply permutation matrices without forming an actual permutation matrix. Array comprehensions in Python can achieve the same effect.

#### 2.1.4.2 LU Factorization with Partial Pivoting

To see how to modify our LU factorization algorithm let's use the recursive approach: what we will do is compute $P$, $L$, and $U$ satisfying $PA = LU$. For $n > 1$ we write $A = \left[\begin{array}{c|c} \alpha & \boldsymbol{r}^T \\ \hline \boldsymbol{c} & \widetilde{A} \end{array}\right]$.

In order to ensure that (at least) the entries of $L$ are not large, at the start of each stage (for each $k$) we find the row $i^*$ which maximizes $|a_{ik}|$ over $i \geq k$, and swap rows $k$ and $i^*$. This strategy is called *partial pivoting* and ensures that after swapping these rows, $|m_{ik}| = |a_{ik}/a_{kk}| \leq 1$. That is, the entries of $L$ are never more than one in magnitude. If, before swapping rows for stage $k$, we have $|a_{kk}| \geq |a_{ik}|$ for all $i \geq k$, then no swap is carried out.

Suppose we apply our row swap algorithm for the first stage ($k = 1$) swapping rows $i_1^*$ and $k = 1$:

$$P_1 A = \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline \boldsymbol{c}' & \widetilde{A}' \end{array}\right].$$

Then we want to factor $P_1 A$: setting $\boldsymbol{m} \leftarrow \boldsymbol{c}'/\alpha'$, and $B \leftarrow \widetilde{A}' - \boldsymbol{m}(\boldsymbol{r}')^T$ we get

$$\left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline \boldsymbol{c}' & \widetilde{A}' \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{m} & I \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & B \end{array}\right].$$

We can recursively factor $\widetilde{P}B = \widetilde{L}\widetilde{U}$ with a permutation matrix $\widetilde{P}$. This gives (noting that $\widetilde{P}^{-1} = \widetilde{P}^T$):

$$\begin{aligned} P_1 A &= \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline \boldsymbol{c}' & \widetilde{A}' \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{m} & I \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{P}^T \widetilde{L}\widetilde{U} \end{array}\right] \\ &= \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{m} & I \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P}^T \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{L}\widetilde{U} \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{m} & \widetilde{P}^T \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{L}\widetilde{U} \end{array}\right] \\ &= \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P}^T \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline \widetilde{P}\boldsymbol{m} & I \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{L}\widetilde{U} \end{array}\right] \\ &= \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P}^T \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline \widetilde{P}\boldsymbol{m} & I \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{L} \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{U} \end{array}\right] \\ &= \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P}^T \end{array}\right] \underbrace{\left[\begin{array}{c|c} 1 & \\ \hline \widetilde{P}\boldsymbol{m} & \widetilde{L} \end{array}\right]}_{=L} \underbrace{\left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{U} \end{array}\right]}_{=U}. \end{aligned}$$

So

$$\left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P} \end{array}\right] P_1 A = \left[\begin{array}{c|c} 1 & \\ \hline \widetilde{P}\boldsymbol{m} & \widetilde{L} \end{array}\right] \left[\begin{array}{c|c} \alpha' & (\boldsymbol{r}')^T \\ \hline & \widetilde{U} \end{array}\right] = LU.$$

Putting $P = \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{P} \end{array}\right] P_1$ gives the permutation matrix for $PA = LU$. Note that we need to use the permuted multipliers $\widetilde{P}\boldsymbol{m}$ instead of the original $\boldsymbol{m}$. This means that any permutation of the rows that we apply, we have to apply to the *previously computed multipliers*. Algorithm 11 shows a non-recursive pseudo-code for LU factorization with partial pivoting.

Since part of the purpose of using partial pivoting is to keep the size of the entries in $L$ and $U$ from becoming large, we should investigate just how large these entries could become. It is clear from the way the algorithm is designed that the entries of $L$ cannot have magnitude greater than one. But what about $U$? The entries of $U$ are the entries of $A$ being overwritten. If we use a superscript $a_{ij}^{(k)}$ to indicate the value of $a_{ij}$ after stage $k$, then

$$a_{ij}^{(k+1)} \leftarrow a_{ij}^{(k)} - m_{ik} a_{kj}^{(k)} \qquad \text{for } i, j > k.$$

Since $|m_{ik}| \leq 1$ we get $\left|a_{ij}^{(k+1)}\right| \leq \left|a_{ij}^{(k)}\right| + \left|a_{kj}^{(k)}\right| \leq 2 \max_{p,q} \left|a_{pq}^{(k)}\right|$. So we obtain a bound

$$\max_{p,q} \left|a_{pq}^{(k+1)}\right| \leq 2 \max_{p,q} \left|a_{pq}^{(k)}\right|.$$

**Algorithm 11** LU factorization with partial pivoting, overwrites $A$

```
1  function LUPP(A) // overwrite A
2    for k = 1, 2, ..., n − 1
3      i* ← arg max_{i≥k} |a_ik|
4      if a_{i*k} = 0 then skip rest of loop
5      for j = k, k + 1, ..., n
6        swap a_{i*j} with a_kj
7      end for
8      for j = 1, ..., k − 1
9        swap m_{i*j} with m_kj
10     end for
11     for i = k + 1, ..., n
12       ℓ_ik ← a_ik/a_kk
13       // do row operations
14       for j = k + 1, ..., n
15         a_ij ← a_ij − ℓ_ik a_kj
16       end for
17     end for
18   end for
19 end function
```

Consequently $\max_{p,q} |u_{pq}| \leq 2^{n-1} \max_{p,q} |a_{pq}|$. Wilkinson [261] found an example in which this actually occurs

$$W_n = \begin{bmatrix} 1 & 0 & 0 & \cdots & 1 \\ -1 & 1 & 0 & \cdots & 1 \\ -1 & -1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & \cdots & 1 \end{bmatrix}.$$

In the first stage of LU factorization, the entries below the $(1, 1)$ entries are zeroed out; no pivoting is done since all entries below the $(1, 1)$ entry have the same magnitude as the $(1, 1)$ entry. No entry in columns 2 through $n - 1$ are changed. However, the entries below the $(1, n)$ entry are doubled. This gives the matrix

$$W_n^{(1)} = \begin{bmatrix} 1 & 0 & 0 & \cdots & 1 \\ 0 & 1 & 0 & \cdots & 2 \\ 0 & -1 & 1 & \cdots & 2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & -1 & -1 & \cdots & 2 \end{bmatrix}.$$

Applying stage $k$ of the LU factorization we see the doubling of entry $(i, n)$ for $i \geq k$. At the end of the factorization, the $(n, n)$ entry has the value $2^{n-1}$.

This exponential growth of the size of entries during the LU factorization process is rarely seen in practice.

#### 2.1.4.3 Complete Pivoting

Complete pivoting requires finding the pair $(i, j)$ that maximizes $|a_{ij}|$ over $i, j \geq k$, and swapping both rows and columns. This means searching over $(n-k+1)^2$ entries of the matrix, making a total of $\sim \frac{1}{3}n^3$ comparisons just for carrying out the pivot search. This has the same asymptotics as the number of flops needed to carry out LU factorization. For this reason, complete pivoting is usually not used.

While partial pivoting gives a factorization $PA = LU$ with $P$ a permutation matrix, complete pivoting involves column and row swaps which can be made independently, leading to a factorization $PAQ^T = LU$ with both $P$ and $Q$ being permutation matrices.

Wilkinson [261] was able to show that $\max_{p,q} |u_{pq}| / \max_{p,q} |a_{pq}|$ is bounded above by

$$n^{1/2}(2 \cdot 3^{1/2} \cdot 4^{1/3} \cdots n^{1/(n-1)})^{1/2} \sim C\, n^{(1/2)(1+\ln n)} \qquad \text{as } n \to \infty.$$

The actual asymptotic behavior of $\max_{p,q} |u_{pq}| / \max_{p,q} |a_{pq}|$ appears to be much better than this, although the true asymptotic behavior is unknown.

#### 2.1.4.4 Diagonally Dominant Matrices

A (row) *diagonally dominant* matrix $A$ is a square matrix where

$$|a_{ii}| \geq \sum_{j:j\neq i} |a_{ij}| \qquad \text{for all } i. \tag{2.1.6}$$

The matrix is called *strongly diagonally dominant* if the inequality in (2.1.6) is strict for all $i$.

Strongly diagonally dominant matrices are invertible: if $D$ is the diagonal part of $A$ then strong diagonal dominance implies that $\left\| D^{-1}(A-D) \right\|_\infty < 1$. By Lemma 2.2, $I + D^{-1}(A-D)$ is invertible. Pre-multiplying by $D$, which is invertible as $D$ has non-zero diagonal entries, we see that $D + (A-D) = A$ is also invertible.

Partial pivoting is not necessary for diagonally dominant matrices.

**Theorem 2.6** *If $A$ is an invertible diagonally dominant matrix, then the LU factorization without partial pivoting will succeed.*

***Proof*** We show that if $A$ is an invertible diagonally dominant matrix and

$$A = \left[\begin{array}{c|c} \alpha & \boldsymbol{r}^T \\ \hline \boldsymbol{c} & \widetilde{A} \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{c}/\alpha & I \end{array}\right] \left[\begin{array}{c|c} \alpha & \boldsymbol{r}^T \\ \hline & B \end{array}\right],$$

then $B$ is also invertible and diagonally dominant. Note that $\det A = \alpha \det B \neq 0$ so $\alpha \neq 0$ and $B$ invertible. We now show that $B$ is diagonally dominant. Note that for all $i$, $|c_i| + \sum_{j:i\neq j} |\widetilde{a}_{ij}| \leq |\widetilde{a}_{ii}|$ and $|r_i| + \sum_{j:i\neq j} |r_j| \leq |\alpha|$ by diagonal dominance of $A$. So

$$\begin{aligned}\sum_{j:i\neq j}|b_{ij}| &= \sum_{j:i\neq j}|\widetilde{a}_{ij}-c_i r_j/\alpha| \le \sum_{j:i\neq j}|\widetilde{a}_{ij}|+\left|\frac{c_i}{\alpha}\right|\sum_{j:i\neq j}|r_j| \\ &\le (|\widetilde{a}_{ii}|-|c_i|)+\left|\frac{c_i}{\alpha}\right|(|\alpha|-|r_i|) \\ &\le |\widetilde{a}_{ii}|-|c_i|+|c_i|-\left|\frac{c_i}{\alpha}\right||r_i| \\ &= |\widetilde{a}_{ii}|-\left|\frac{c_i}{\alpha}\right||r_i| \le \left|\widetilde{a}_{ii}-\frac{c_i}{\alpha}r_i\right| = |b_{ii}|\,,\end{aligned}$$

as we wanted. □

### *2.1.5 Variants of LU Factorization*

(2.1.7) $A = LL^T$ Cholesky factorization, $L$ factorization

(2.1.8) $A = LDL^T$ $L$ lower triangular, $D$ diagonal.

The *Cholesky factorization* of a *symmetric* matrix requires the matrix $A$ be *positive definite* as well:

(2.1.9) $$z^T Az > 0 \quad \text{for all } z \neq \mathbf{0}.$$

While the $LDL^T$ factorization does not require that $A$ is positive definite, the factorization can be numerically unstable if $A$ is not positive definite. An improvement is the *Bunch–Kaufman* (BK) *factorization* [38, 39] which is an $LDL^T$ factorization where $D$ is a block-diagonal matrix with $1 \times 1$ or $2 \times 2$ diagonal blocks along with symmetric row and column swaps. The BK factorization is intended for situations where $A$ is symmetric, but not positive definite.

There are variants of LU factorization that use block matrices for performance improvements. These have no effect on the total number of floating point operations, but do change the way memory is accessed.

#### 2.1.5.1 Positive-Definite Matrices

Positive-definite matrices arise in a number of contexts: in statistics where, for example, variance–covariance matrices are positive definite; in optimization where convex functions can be identified by having positive-definite Hessian matrices; in physics, quadratic energy functions are generated by positive-definite matrices; in partial differential equations, where the discretization of elliptic equations leads to positive-definite matrices.

We take (2.1.9) as the definition that $A$ is positive definite. Many authors *assume* that when a matrix is described as positive definite, it must also be symmetric. Here we do not.[1] If a matrix is both positive definite *and* symmetric, we will say so explicitly. We do assume, unless otherwise stated, that a matrix is real. For a complex matrix $A$ we modify definition (2.1.9) to

$$\text{Re}\,\overline{z}^T Az > 0 \qquad \text{for all complex } z \neq \mathbf{0}. \tag{2.1.10}$$

Here $\overline{z}$ is the vector $z$ with the entries of $z$ replaced by their complex conjugates. Note that the condition "$\overline{z}^T Az > 0$" implies that $\overline{z}^T Az$ is real. In the complex case, instead of asking for $A$ to be symmetric ($A^T = A$), we ask for $A$ to be *Hermitian*:

$$\overline{A}^T = A; \qquad \text{that is, } \overline{a_{\ell k}} = a_{k\ell} \qquad \text{for all } k, \ell. \tag{2.1.11}$$

A real matrix $A$ is positive definite if and only if its symmetric part $\frac{1}{2}(A + A^T)$ is positive definite. For complex matrices, $A$ is positive definite if and only if $\frac{1}{2}(A + \overline{A}^T)$. As an example of a positive-definite matrix that is not symmetric, consider

$$A = \begin{bmatrix} 2 & 1 \\ 0 & 1 \end{bmatrix}.$$

From definition (2.1.9), positive-definite matrices have a number of important properties: positive-definite matrices are invertible; if $A$ and $B$ are positive-definite matrices, so are $A + B$, $\alpha A$ for $\alpha > 0$, and $A^{-1}$. Symmetric positive-definite matrices can also be understood in terms of eigenvalues: all the eigenvalues of a symmetric matrix are real, but the eigenvalues of a symmetric positive-definite matrix are positive: if $A\boldsymbol{v} = \lambda \boldsymbol{v}$ and $\boldsymbol{v} \neq \mathbf{0}$ then $\boldsymbol{v}^T A\boldsymbol{v} = \lambda \boldsymbol{v}^T \boldsymbol{v} > 0$ so $\lambda > 0$. In fact, a symmetric matrix is positive definite if and only if all its eigenvalues are positive.

Clearly the $n \times n$ identity matrix is positive definite. Also if $A$ is positive definite, and $X$ is invertible, then $X^T AX$ is also positive definite. The diagonal entries of a positive-definite matrix are positive.

In spite of the ubiquity of positive-definite matrices, actually proving that a given matrix is positive definite (or not) can be a challenge. Fortunately, Sylvester's criterion gives a fairly easy way to tell (apart from looking at all the eigenvalues).

**Theorem 2.7** *(Sylvester's criterion) If $A$ is an $n \times n$ symmetric real or complex Hermitian matrix, then it is positive definite if and only if*

$$\det \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix} > 0 \qquad \textit{for } k = 1, 2, \ldots, n. \tag{2.1.12}$$

[1] Stoer and Bulirsch [241, pp. 180–181] assume "positive definite" implies "symmetric" while Golub and van Loan [105, pp. 14–142] explicitly allow non-symmetric positive-definite matrices.

Related to positive-definite matrices are (real) positive semi-definite matrices:

$$z^T Az \geq 0 \qquad \text{for all } z. \tag{2.1.13}$$

Sums of positive semi-definite matrices $A + B$ are positive semi-definite, and if in addition, either $A$ or $B$ is positive definite, then $A + B$ is positive definite. For any $X$ (even rectangular), if $A$ is positive semi-definite then so is $X^T AX$ provided the matrix product is well defined. A symmetric positive semi-definite matrix $A$ that is also invertible is positive definite. Symmetry is important here: $\begin{bmatrix} 0 & +1 \\ -1 & 0 \end{bmatrix}$ is both positive semi-definite and invertible, but it is *not* positive definite.

Symmetric matrices $B$ have real eigenvalues and an orthonormal basis of eigenvectors. This is equivalent to the existence of an orthogonal matrix $Q$ where

$$Q^T BQ = \Lambda = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}.$$

The columns of $Q$ are eigenvectors of $B$. If $B$ is symmetric and positive definite, then all these eigenvalues are positive: $\lambda_i = \boldsymbol{q}_i^T B\boldsymbol{q}_i > 0$ where $\boldsymbol{q}_i$ is the $i$th column of $Q$. Also, $\|B\|_2 = \max_i \lambda_i$. To see this, any vector $\boldsymbol{x}$ can be written $\boldsymbol{x} = \sum_{i=1}^n c_i \boldsymbol{q}_i$ so for symmetric positive definite $B$,

$$\|B\|_2 = \max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\|B\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} = \max_{\boldsymbol{c}\neq\boldsymbol{0}} \frac{\sqrt{\sum_{i=1}^n (\lambda_i c_i)^2}}{\sqrt{\sum_{i=1}^n c_i^2}} = \max_i \lambda_i, \qquad \text{and} \tag{2.1.14}$$

$$\max_{\boldsymbol{x}\neq\boldsymbol{0}} \frac{\boldsymbol{x}^T B\boldsymbol{x}}{\boldsymbol{x}^T\boldsymbol{x}} = \max_{\boldsymbol{c}\neq\boldsymbol{0}} \frac{\sum_{i=1}^n \lambda_i c_i^2}{\sum_{i=1}^n c_i^2} = \max_i \lambda_i. \tag{2.1.15}$$

#### 2.1.5.2 Cholesky Factorization

The Cholesky factorization of a symmetric matrix $A$ has the form

$$A = LL^T, \qquad L \text{ lower triangular.}$$

If $A$ has a Cholesky factorization, then $A$ is symmetric and positive semi-definite:

$$z^T Az = z^T LL^T z = (L^T z)^T (L^T z) = \left\|L^T z\right\|_2^2 \geq 0.$$

**Algorithm 12** Cholesky factorization; overwrites $A$

```
1  function cholesky(A)
2    for k = 1, 2, ..., n
3      ℓ_kk ← √a_kk
4      for j = k + 1, ..., n
5        ℓ_jk ← a_jk / ℓ_kk
6        for i = k + 1, ..., j
7          a_ij ← a_ij − ℓ_ik ℓ_jk
8          a_ji ← a_ij  // optional
9        end for
10     end for
11   end for
12   return L
13 end function
```

But if $A$ is invertible and has a Cholesky factorization, then $A$ must be positive definite. Our algorithm for the Cholesky factorization of a symmetric matrix will assume that $A$ is positive definite, rather than just positive semi-definite.

We can recursively construct a Cholesky factorization of a symmetric positive-definite matrix as follows. Let $A = \left[\begin{array}{c|c} \alpha & \boldsymbol{a}^T \\ \hline \boldsymbol{a} & \widetilde{A} \end{array}\right]$ be symmetric and positive definite. The base case for the recursion is where $A$ is a $1 \times 1$ matrix: $A = [\alpha] = [\lambda]\,[\lambda]$ where $\lambda = \sqrt{\alpha}$. If $A$ is $n \times n$ with $n > 1$, we note that $\alpha = \boldsymbol{e}_1^T A \boldsymbol{e}_1 > 0$. Let $\lambda = \sqrt{\alpha}$, and $\boldsymbol{\ell} = \boldsymbol{a}/\lambda$. Then

$$A = \left[\begin{array}{c|c} \alpha & \boldsymbol{a}^T \\ \hline \boldsymbol{a} & \widetilde{A} \end{array}\right] = \left[\begin{array}{c|c} \lambda & \\ \hline \boldsymbol{\ell} & I \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{A} - \boldsymbol{\ell}\boldsymbol{\ell}^T \end{array}\right] \left[\begin{array}{c|c} \lambda & \boldsymbol{\ell}^T \\ \hline & I \end{array}\right].$$

The matrix in the middle is also positive definite, since

$$\left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{A} - \boldsymbol{\ell}\boldsymbol{\ell}^T \end{array}\right] = \left[\begin{array}{c|c} \lambda & \\ \hline \boldsymbol{\ell} & I \end{array}\right]^{-1} A \left[\begin{array}{c|c} \lambda & \boldsymbol{\ell}^T \\ \hline & I \end{array}\right]^{-1}.$$

The inverse of $\left[\begin{array}{c|c} \lambda & \\ \hline \boldsymbol{\ell} & I \end{array}\right]$ exists as its determinant is $\lambda > 0$. Thus $\widetilde{A} - \boldsymbol{\ell}\boldsymbol{\ell}^T$ is also positive definite. It is symmetric as $\widetilde{A}$ and $\boldsymbol{\ell}\boldsymbol{\ell}^T$ are both symmetric. Thus, we can recursively compute a Cholesky factorization $\widetilde{A} - \boldsymbol{\ell}\boldsymbol{\ell}^T = \widetilde{L}\widetilde{L}^T$. That is,

$$A = \left[\begin{array}{c|c} \lambda & \\ \hline \boldsymbol{\ell} & I \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline & \widetilde{L}\widetilde{L}^T \end{array}\right] \left[\begin{array}{c|c} \lambda & \boldsymbol{\ell}^T \\ \hline & I \end{array}\right] = \left[\begin{array}{c|c} \lambda & \\ \hline \boldsymbol{\ell} & \widetilde{L} \end{array}\right] \left[\begin{array}{c|c} \lambda & \boldsymbol{\ell}^T \\ \hline & \widetilde{L}^T \end{array}\right] = L\,L^T.$$

A non-recursive pseudo-code for Cholesky factorization is given in Algorithm 12.

The number of floating point operations for Cholesky factorization is $\sim \frac{1}{3}n^3$as $n \to \infty$, about half the number needed for LU factorization of $A$.

While halving the number of operations may seem like a major benefit for this method, in practice, memory movement techniques can be much more important for performance. Cholesky factorization, however, is able to take advantage of many of these issues. One of the benefits of the Cholesky factorization is the pivoting is not necessary for numerical stability. In particular, if $A = LL^T$ then

$$a_{ii} = \boldsymbol{e}_i^T A \boldsymbol{e}_i = \boldsymbol{e}_i^T LL^T \boldsymbol{e}_i = \left\| L^T \boldsymbol{e}_i \right\|_2^2 \geq \ell_{ij}^2,$$

so the entries of the $i$th row of $L$ are bounded by $\sqrt{a_{ii}}$. Thus the entries of $L$ can never exceed $\|A\|_2^{1/2}$. Because no pivoting is necessary, the additional costs of pivot searches and swapping rows and/or columns are avoided. Block and parallel algorithms can exploit this certainty. Consider the recursive block version: given

$$A = \left[\begin{array}{c|c} A_{11} & A_{21}^T \\ \hline A_{21} & A_{22} \end{array}\right] = \left[\begin{array}{c|c} L_{11} & \\ \hline L_{21} & L_{22} \end{array}\right] \left[\begin{array}{c|c} L_{11}^T & L_{21}^T \\ \hline & L_{22}^T \end{array}\right],$$

so $A_{11} = L_{11} L_{11}^T$ (Cholesky factorization of a block). Then $A_{21} = L_{21} L_{11}^T$ so $L_{21} = A_{21} L_{11}^{-T}$, and we have $A_{22} - L_{21} L_{21}^T = L_{22} L_{22}^T$ (recursive Cholesky factorization). By choosing $A_{11}$ to be the top-left $b \times b$ submatrix of $A$, we can perform a $b \times b$ Cholesky factorization using the standard algorithm, and then use matrix–matrix operations. The matrix $L_{11}^{-T}$ can be explicitly computed in $\mathcal{O}(b^3)$ flops. We can thereby leverage the power of BLAS-3 operations (see Section 1.1.7).

By comparison, LU factorization with partial pivoting must work with the entire block column $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$, rather than just $A_{11}$, to compute $L_{11}$ and $U_{11}$. This means that a much larger amount of data has to be processed, which might not fit in cache memory.

#### 2.1.5.3 $LDL^T$ Factorization and BK Factorization

Both $LDL^T$ and BK (Bunch–Kaufman) [39] factorizations involve having a diagonal or block-diagonal matrix $D$, and $L$ matrices that have one's on the diagonal. The $LDL^T$ factorization of a symmetric positive-definite matrix is equivalent to the Cholesky factorization, as then the diagonal matrix $D$ must have positive diagonal entries, and

$$A = LDL^T = (LD^{1/2})(LD^{1/2})^T,$$
$$(D^{1/2})_{kk} = \sqrt{d_{kk}} \quad \text{and} \quad (D^{1/2})_{k\ell} = 0 \quad \text{if } k \neq \ell.$$

If $A$ is positive definite, the advantage of the $LDL^T$ factorization is avoiding computing square roots. Square roots take roughly 10–20 times as long to compute as addition, subtraction, or multiplication in modern architectures, so this could improve performance. On the other hand, there are only $n$ square root computations in com-

puting the Cholesky factorization of an $n \times n$ matrix compared with $\sim \frac{1}{3}n^3$ other floating point operations. The cost of these $n$ square roots is small compared to the other floating point operations in Cholesky factorization for $n > 10$.

To see how the $LDL^T$ factorization works, consider the recursive decomposition

$$A = \left[\begin{array}{c|c} \alpha & \boldsymbol{a}^T \\ \hline \boldsymbol{a} & \widetilde{A} \end{array}\right] = \left[\begin{array}{c|c} 1 & \\ \hline \boldsymbol{\ell} & \widetilde{L} \end{array}\right] \left[\begin{array}{cc} \delta & \\ & \widetilde{D} \end{array}\right] \left[\begin{array}{c|c} 1 & \boldsymbol{\ell}^T \\ \hline & \widetilde{L}^T \end{array}\right] = LDL^T.$$

From this, we have the equations

$$\begin{aligned} \delta &= \alpha, \\ \delta\boldsymbol{\ell} &= \boldsymbol{a}, \qquad \text{so } \boldsymbol{\ell} = \boldsymbol{a}/\delta = \boldsymbol{a}/\alpha \\ \widetilde{A} - \alpha\boldsymbol{\ell}\boldsymbol{\ell}^T &= \widetilde{L}\widetilde{D}\widetilde{L}^T \qquad \text{(recursive } LDL^T \text{ factorization)}. \end{aligned}$$

This algorithm can work for some matrices that are not positive definite or positive semi-definite, such as

$$\begin{bmatrix} -2 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \\ -\frac{1}{2} & 1 \end{bmatrix} \begin{bmatrix} -2 & \\ & \frac{3}{2} \end{bmatrix} \begin{bmatrix} 1 & -\frac{1}{2} \\ & 1 \end{bmatrix}.$$

However, this approach will not work for

$$A_0 = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}.$$

Indeed, this approach will fail for any symmetric matrix where there is a zero submatrix in the top-left corner. To handle this, we need to incorporate symmetric row and column pivoting. We also need to allow $2 \times 2$ diagonal blocks. This is the approach of the Bunch–Kaufman (BK) factorization.

The *BK factorization* [39] of a matrix $A$ is

$$PAP^T = LDL^T,$$

where $P$ is a permutation matrix, $L$ is a lower triangular matrix with ones on the diagonal, and $D$ is a block-diagonal matrix with diagonal blocks that are either $1 \times 1$ or $2 \times 2$. The BK algorithm is essentially a block $LDL^T$ factorization combined with a special symmetric pivoting strategy. The BK pivoting strategy at stage $k$ involves scanning columns $k$ and $k + 1$ and checking the diagonal entries as well. There is still the possibility of exponential growth in the entries of the factored matrices, although like the possible exponential growth in entries for partial pivoting, it is rare.

**Algorithm 13** Block LU factorization

```
1   function blockLU(A)
2     for k = 1, 2, ..., p
3       A_kk = L_kk U_kk  // LU fact'n
4       for i = k + 1, ..., p
5         L_ik ← A_ik U_kk^{-1}  // forward subst'n
6         U_ki ← L_kk^{-1} A_ki  // forward subst'n
7         for j = k + 1, ..., p
8           A_ij ← A_ij − L_ik U_kj
9         end for
10      end for
11    end for
12    return ([L_ij], [U_ij])
13  end function
```

#### 2.1.5.4 Block Factorizations

Block factorizations can exploit the advantages of matrix–matrix operations on modern computer architectures. Computing the matrix–matrix product $AB$ where $A$ and $B$ are $n \times n$ matrices using the standard algorithm takes $\mathcal{O}(n^3)$ floating point operations, but involves only $\mathcal{O}(n^2)$ data transfers. If both $A$ and $B$ and the product can fit in cache simultaneously, the number of computations per data transfer is $\mathcal{O}(n)$. Since in modern architectures, the time needed for transferring just one floating point number can be used for many arithmetic operations, matrix–matrix multiplications can be performed very efficiently.

In comparison, computing matrix–vector products $A\boldsymbol{x}$ takes $\mathcal{O}(n^2)$ floating point operations and $\mathcal{O}(n^2)$ data transfers. This means that no matter how large $n$ is, data transfers will take a significant amount of time. With the high cost of data transfers, most of the time spent in computing $A\boldsymbol{x}$ will be in transferring data, rather than performing numerical operations.

LAPACK [5] makes use of block matrix–matrix operations to increase efficiency.

To see how to use them, consider computing the LU factorization of a block matrix $A$ without pivoting:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1p} \\ A_{21} & A_{22} & \cdots & A_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & A_{pp} \end{bmatrix} = \begin{bmatrix} L_{11} & & & \\ L_{21} & L_{22} & & \\ \vdots & \vdots & \ddots & \\ L_{p1} & L_{p2} & \cdots & L_{pp} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & \cdots & U_{1p} \\ & U_{22} & \cdots & U_{2p} \\ & & \ddots & \vdots \\ & & & U_{pp} \end{bmatrix} = LU.$$

Each block matrix $A_{ij}$ is $b \times b$ or smaller; the number of block rows and columns is $p = \lceil n/b \rceil$. We choose $b$ so that two $b \times b$ matrices can be kept in either the level-one or level-two cache. The block LU factorization algorithm is then given by Algorithm 13.

If $b$ is too large for two $b \times b$ matrices to fit in cache, then each block operation will overflow the cache. This results in cache misses, and data transfer from main

memory or higher level cache. The additional data transfers takes substantial time. Keeping $b$ as large as possible but avoiding cache overflow gives an optimal blocking version.

For LU factorization *with pivoting*, we start by the LU factorization with pivoting for the first block column:

$$P_1 \begin{bmatrix} A_{11} \\ A_{21} \\ \vdots \\ A_{p1} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \\ \vdots \\ L_{p1} \end{bmatrix} U_{11}.$$

Then we can perform the block operations $U_{1j} \leftarrow L_{11}^{-1} A_{1j}$ for $j = 2, \ldots, p$ and $A_{ij} \leftarrow A_{ij} - L_{i1} U_{1j}$ for $i, j = 2, \ldots, p$. Then we recursively apply the method to $\big[A_{ij} \mid i, j = 2, \ldots, p\big]$.

For block QR factorization of $A$, like the LU factorization with pivoting, we apply the QR factorization to the first block column of $A$. An efficient way of storing the $Q$ matrix is the $WY$ form [105, Sec. 5.1.7, pp. 213–215].

There are also block-sparse matrices which consist of blocks $A_{ij}$ for $(i, j)$ in a sparse set. Block-sparse matrices spread the cost of navigating the sparse matrix data structure across more floating point operations. This also reduces the memory overhead compared to ordinary sparse matrices.

#### 2.1.5.5 Sherman–Morrison–Woodbury Formula

Although the *Sherman–Morrison–Woodbury formula* [116] is not a factorization, it can be very helpful in solving systems of linear equations, especially for solving a sequence of linear equations where the matrix changes by a low-rank matrix. The idea is that if we know how to solve $A\boldsymbol{x} = \boldsymbol{b}$ for $\boldsymbol{x}$ given any $\boldsymbol{b}$, we can efficiently solve $(A + \boldsymbol{u}\boldsymbol{v}^T)\boldsymbol{z} = \boldsymbol{y}$ for $\boldsymbol{z}$ given any $\boldsymbol{y}$. We say that $A + \boldsymbol{u}\boldsymbol{v}^T$ is a rank-1 modification of $A$ since $\boldsymbol{u}\boldsymbol{v}^T$ is a rank-1 matrix.

The basic *Sherman–Morrison formula* is

$$(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^T A^{-1}}{1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u}} \quad \text{provided } \boldsymbol{v}^T A^{-1}\boldsymbol{u} \neq -1. \tag{2.1.16}$$

To see why this is so, suppose that $(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} + \boldsymbol{r}\boldsymbol{s}^T$. Then

$$(A + \boldsymbol{u}\boldsymbol{v}^T)(A^{-1} + \boldsymbol{r}\boldsymbol{s}^T) = I.$$

Expanding gives

$$A\,A^{-1} + A\boldsymbol{r}\,\boldsymbol{s}^T + \boldsymbol{u}\boldsymbol{v}^T A^{-1} + \boldsymbol{u}\boldsymbol{v}^T\boldsymbol{r}\boldsymbol{s}^T = I.$$

Since $A\,A^{-1} = I$, subtracting this from both sides gives

$$(A\boldsymbol{r} + \boldsymbol{u}\boldsymbol{v}^T\boldsymbol{r})\boldsymbol{s}^T = -\boldsymbol{u}(\boldsymbol{v}^T A^{-1}).$$

Put $\boldsymbol{s}^T = -\boldsymbol{v}^T A^{-1}$. Then we have to solve $A\boldsymbol{r} = \boldsymbol{u} - \boldsymbol{u}\boldsymbol{v}^T\boldsymbol{r} = (1 - \boldsymbol{v}^T\boldsymbol{r})\boldsymbol{u}$. So $\boldsymbol{r} = (1 - \boldsymbol{v}^T\boldsymbol{r})A^{-1}\boldsymbol{u}$. Let $\gamma = 1 - \boldsymbol{v}^T\boldsymbol{r}$. Then $\boldsymbol{r} = \gamma A^{-1}\boldsymbol{u}$ and $\gamma = 1 - \boldsymbol{v}^T\gamma A^{-1}\boldsymbol{u}$; solving for $\gamma$ gives $\gamma = 1/(1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u})$ provided the denominator is not zero. Combining the results for $\boldsymbol{r}$ and $\boldsymbol{s}$ gives

$$(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} - \frac{A^{-1}\boldsymbol{u}\boldsymbol{v}^T A^{-1}}{1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u}}$$

as we wanted.

To use this for solving $(A + \boldsymbol{u}\boldsymbol{v}^T)\boldsymbol{z} = \boldsymbol{y}$ efficiently, we can first compute $\boldsymbol{w} = A^{-1}\boldsymbol{u}$, $\boldsymbol{s}^T = -\boldsymbol{v}^T A^{-1}$ and $\boldsymbol{r} = \boldsymbol{w}/(1 + \boldsymbol{v}^T\boldsymbol{w})$ provided the denominator is not zero. Assuming we have the LU factorization of $A$, this can be computed in $\mathcal{O}(n^2)$ flops. The solution to our rank-1 modified equations is

$$\boldsymbol{z} = A^{-1}\boldsymbol{y} + \boldsymbol{r}(\boldsymbol{s}^T\boldsymbol{y}).$$

If we have the LU factorization of $A$, we can compute $A^{-1}\boldsymbol{y}$ in $\mathcal{O}(n^2)$ flops.

Numerical difficulties can occur if the denominator $1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u}$ is small. This is likely at some point in chains of rank-1 modifications $A + \sum_{j=1}^{k} \boldsymbol{u}_j\boldsymbol{v}_j^T$, $k = 1, 2, \ldots$, which occur in some applications. In that case, a chain of rank-1 modifications of the inverse can be created. But to avoid numerical instability, the values of $\|\boldsymbol{r}_j\|_2 \|\boldsymbol{s}_j\|_2$ should be monitored; if they become large, the chain should be terminated, and a fresh LU factorization of $A + \sum_{j=1}^{k} \boldsymbol{u}_j\boldsymbol{v}_j^T$ should be computed.

A generalization to rank-$r$ modifications of a matrix is the Sherman–Morrison–Woodbury formula:

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^T A^{-1}U)^{-1}V^T A^{-1}, \tag{2.1.17}$$

provided the matrix $I + V^T A^{-1}U$ is invertible. If $U$ and $V$ are $n \times r$ matrices with linearly independent columns, then $UV^T$ is a rank-$r$ matrix, and $A + UV^T$ is a rank-$r$ modification of $A$.

**Exercises.**

(1) Let $H_n$ be the $n \times n$ Hilbert matrix: $(H_n)_{ij} = 1/(i + j - 1)$. Given $n$, generate $\boldsymbol{x}$ as a random vector in $\mathbb{R}^n$ and set $\boldsymbol{b} \leftarrow H_n\boldsymbol{x}$. Solve $H_n\widehat{\boldsymbol{x}} = \boldsymbol{b}$ numerically. Compute $\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|_2 / \|\boldsymbol{x}\|_2$ and $\|H_n\widehat{\boldsymbol{x}} - \boldsymbol{b}\|_2 / \|\boldsymbol{b}\|_2$. Do this for $n = 6, 8, 10, \ldots, 20$. Report how the relative errors change as $n$ increases. Compare their growth with $\kappa_2(H_n)$.

(2) An $n \times n$ matrix is strictly diagonally dominant if $|a_{ii}| > \sum_{j:j\neq i} |a_{ij}|$ for all $i$. Show that strictly row diagonally dominant matrices are invertible. [**Hint:** Write $A = D + F$ where $D$ is the diagonal part of $A$, so that $f_{ij} = a_{ij}$

if $i \neq j$ and $f_{ii} = 0$. Then note that $A = D(I + D^{-1}F)$ and $\left\|D^{-1}F\right\|_\infty = \max_i \left(\sum_{j:j\neq i} |a_{ij}|\right) / |a_{ii}| < 1$.] Give an example of a row diagonally dominant matrix (but not strictly dominant) that is *not* invertible: $|a_{ii}| \geq \sum_{j:j\neq i} |a_{ij}|$ for all $i$.

(3) The inverse $B$ of a matrix $A$ can be computed from its LU factorization: solve $LU\boldsymbol{b}_j = \boldsymbol{e}_j$, $j = 1, 2, \ldots, n$ for $\boldsymbol{b}_j$, the $j$th column of $B$. Here $\boldsymbol{e}_j$ is the $j$th standard basis vector: $(\boldsymbol{e}_j)_i = 1$ if $i = j$ and zero otherwise. Determine how many floating point operations are needed for this computation. See if you can improve these counts by using the fact that $(\boldsymbol{e}_j)_i = 0$ for $i < j$.

(4) Develop a block LU factorization algorithm without pivoting by starting with the decomposition

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] = \left[\begin{array}{c|c} L_{11} & \\ \hline L_{21} & L_{22} \end{array}\right] \left[\begin{array}{c|c} U_{11} & U_{12} \\ \hline & U_{22} \end{array}\right],$$

with $A_{11}$ a $b \times b$ matrix. Use the standard LU factorization algorithm for factoring $A_{11} = L_{11}U_{11}$.

(5) Modify the block LU factorization of the previous question to include partial pivoting. Note that $A_{11}$ might not be invertible, so that swapping rows in $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$ is necessary. Use the standard LU factorization algorithm with partial pivoting for $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$.

(6) Tail-end recursion is where the final statement in a function is a recursive call: Re-write this as an equivalent while loop. Note that the "$x$" here may actually be a very complex object, such as a partial matrix factorization.

(7) Re-write the code below using a pair of while loops with no recursion:

```
function g(x, n)
  if n = 0: return x
  else:     return k(g(h(x), n − 1))
end function
```

[**Hint:** First try computing what the return value should be for $n = 0,\ 1,\ 2,\ 3$, and then generalize.]

(8) Using a block LU factorization of $A$,

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] = \left[\begin{array}{c|c} I & \\ \hline L_{21} & I \end{array}\right] \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline & U_{22} \end{array}\right]$$

show that $A$ is invertible if $A_{11}$ and the *Schur complement* $S := A_{22} - A_{21}A_{11}^{-1}A_{12}$ are both invertible.

(9) In question 7, show that $\det A = \det A_{11} \det S$.

(10) In question 7, show that if $A$ is symmetric, so is $S$. If, in addition, $A$ is positive definite, show that $S$ is also positive definite. [**Hint:** Try creating a symmetric "$LDL^T$" version of the block factorization in Exercise 8.]

(11) Show that for any positive semi-definite matrix $A$ there is a Cholesky factorization $A = LL^T$ with $L$ lower triangular. [**Hint:** For any $\alpha > 0$, $A + \alpha I = L_\alpha L_\alpha^T$ by the standard Cholesky factorization. Show from the bounds on the entries of $L_\alpha$ that the $L_\alpha$ matrices belong to a closed and bounded subset of $\mathbb{R}^{n\times n}$; therefore, there is a convergent subsequence with limit $L$ satisfying $A = LL^T$.]

(12) Show that the $LDL^T$ factorization can be numerically unstable even when it succeeds, by considering the matrix $\begin{bmatrix} \epsilon & 1 \\ 1 & 1 \end{bmatrix}$ with $0 \neq \epsilon \approx 0$.

(13) Show that the LU factorization with partial pivoting is invariant under column scaling, that is, if $D$ is diagonal and $PA = LU$ is the LU factorization of $A$ with partial pivoting, then $P(AD) = L(UD)$ is the LU factorization of $AD$ with partial pivoting.

(14) Show that for any matrix norm and $n \times n$ matrices $A$ and $B$, the condition number $\kappa(AB) \leq \kappa(A)\,\kappa(B)$.

(15) Show that if $D$ is diagonal and $P$ a permutation matrix, then $\widetilde{D} := PDP^T$ is also diagonal. Then show that if $PA = LU$ is the LU factorization of $A$ with partial pivoting, then $P(DA) = (\widetilde{D}L)U$ is a factorization of the row-scaled matrix $DA$ with row swaps. Give an example where $P(DA) = (\widetilde{D}L\widetilde{D}^{-1})\widetilde{D}U$ is *not* the LU factorization of $DA$ *with partial pivoting* as some entry of $\widetilde{D}L\widetilde{D}^{-1}$ has absolute value greater than one. (We need to have the extra factor of $\widetilde{D}^{-1}$ so that the diagonal entries of $\widetilde{D}L\widetilde{D}^{-1}$ are one.)

(16) Sylvester's criterion gives a way of identifying symmetric positive-definite matrices. However, replacing "$>$" with "$\geq$" in inequalities (2.1.12) does *not* give a proper test for symmetric positive *semi-definite* matrices. Show this with the example matrix $\begin{bmatrix} 0 & 0 \\ 0 & -1 \end{bmatrix}$.

(17) The Sherman–Morrison formula $(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} - A^{-1}\boldsymbol{u}\boldsymbol{v}^T A^{-1}/(1 + \boldsymbol{v}^T A^{-1}\boldsymbol{u})$ gives a way of inverting a rank-1 update of an invertible matrix. Show that we can write it in compact form $(A + \boldsymbol{u}\boldsymbol{v}^T)^{-1} = A^{-1} + \boldsymbol{r}\boldsymbol{s}^T$. Implement this as a function that, given a factorization of $A$ (the LU factorization with partial pivoting, for example) and vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, returns the pair $\boldsymbol{r}$ and $\boldsymbol{s}$. Give an operation count for your method.

(18) Extend the idea of question 17 so that given $A$, a factorization of $A$, and $(\boldsymbol{u}_i, \boldsymbol{v}_i)$, $i = 1, 2, \ldots, m$, you compute $(\boldsymbol{r}_i, \boldsymbol{s}_i)$, $i = 1, 2, \ldots, m$, so that $\left(A + \sum_{i=1}^m \boldsymbol{u}_i \boldsymbol{v}_i^T\right)^{-1} = A^{-1} + \sum_{i=1}^m \boldsymbol{r}_i \boldsymbol{s}_i^T$. Give an operation count for your method.

## 2.2 Least Squares Problems

Least squares problems are commonly used in statistics for fitting data to a model. For example, we might want to fit a set of data points $(x_i, y_i)$, $i = 1, 2, \ldots, n$, to a straight line, or a quadratic $q(x) = ax^2 + bx + c$: $y_i \approx q(x_i) = ax_i^2 + bx_i + c$. The

"best" fit is typically taken to be the one that minimizes the sum of the squares of the errors: $\sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (q(x_i) - y_i)^2$ over all possible choices of the unknown coefficients $a$, $b$, and $c$.

The conditions for solving least squares problems can be related to orthogonal complements: the orthogonal complement of a vector space $V \subseteq \mathbb{R}^n$ is the set

$$V^{\perp} := \left\{ \boldsymbol{y} \mid \boldsymbol{v}^T \boldsymbol{y} = 0 \text{ for all } \boldsymbol{v} \in V \right\}. \tag{2.2.1}$$

### 2.2.1 The Normal Equations

If we want to fit a quadratic function $q(x) = ax^2 + bx + c$ to a data set $(x_i, y_i)$, $i = 1, 2, \ldots, n$, we typically aim to minimize $\sum_{i=1}^{n} e_i^2$ where $e_i = y_i - q(x_i)$. Here the unknowns are the coefficients $a$, $b$, $c$. Setting $\boldsymbol{c} = [a, b, c]^T$, we see that $e_i = y_i - [1, x_i, x_i^2]\boldsymbol{c}$ and so

$$\boldsymbol{e} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ 1 & x_3 & x_3^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} = A\,\boldsymbol{c} - \boldsymbol{y}.$$

That is, we want to minimize

$$\sum_{i=1}^{n} e_i^2 = \boldsymbol{e}^T \boldsymbol{e} = \|\boldsymbol{e}\|_2^2 = \|A\boldsymbol{c} - \boldsymbol{y}\|_2^2.$$

If $\boldsymbol{c}^*$ minimizes $\varphi(\boldsymbol{c}) := \|A\boldsymbol{c} - \boldsymbol{y}\|_2^2$, then we have the first-order necessary condition

$$\frac{d}{ds}\, \varphi(\boldsymbol{c}^* + s\boldsymbol{d})\big|_{s=0} = 0 \qquad \text{for any } \boldsymbol{d}.$$

For our objective function

$$\begin{aligned} \frac{d}{ds}\varphi(\boldsymbol{c} + s\boldsymbol{d}) &= \frac{d}{ds}\left(\|A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y}\|_2^2\right) \\ &= \frac{d}{ds}\left((A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y})^T\,(A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y})\right) \\ &= 2\,(A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y})^T\,\frac{d}{ds}\,(A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y}) \\ &= 2\,(A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y})^T\,A\boldsymbol{d}. \end{aligned} \tag{2.2.2}$$

In particular,

$$0 = \frac{d}{ds}\,\varphi(\boldsymbol{c}^* + s\boldsymbol{d})\big|_{s=0} = 2\left(A\boldsymbol{c}^* - \boldsymbol{y}\right)^T A\boldsymbol{d} = 2(A\boldsymbol{d})^T\left(A\boldsymbol{c}^* - \boldsymbol{y}\right)$$
$$= 2\boldsymbol{d}^T A^T\left(A\boldsymbol{c}^* - \boldsymbol{y}\right) \qquad \text{for all } \boldsymbol{d}.$$

That is,

$$A^T\left(A\boldsymbol{c}^* - \boldsymbol{y}\right) = \boldsymbol{0}, \qquad \text{so}$$
$$A^T A\boldsymbol{c}^* = A^T\boldsymbol{y}. \tag{2.2.3}$$

Equation (2.2.3) is called the *normal equations*.

The condition that $A^T\,(A\boldsymbol{c}^* - \boldsymbol{y}) = \boldsymbol{0}$ is equivalent to requiring that $A\boldsymbol{c}^* - \boldsymbol{y}$ is in the orthogonal complement to range$(A)$: $A\boldsymbol{c}^* - \boldsymbol{y} \in \text{range}(A)^\perp$.

These are necessary conditions, but they are also sufficient:

$$\frac{d^2}{ds^2}\varphi(\boldsymbol{c} + s\boldsymbol{d}) = \frac{d}{ds}2\,(A(\boldsymbol{c} + s\boldsymbol{d}) - \boldsymbol{y})^T A\boldsymbol{d} \qquad \text{by (2.2.2)}$$
$$= 2\,(A\boldsymbol{d})^T(A\boldsymbol{d}) = 2\,\|A\boldsymbol{d}\|_2^2 \geq 0.$$

Using Taylor series with second-order remainder (1.6.1):

$$\varphi(\boldsymbol{c}^* + t\boldsymbol{d}) = \varphi(\boldsymbol{c}^*) + \frac{d}{ds}\,\varphi(\boldsymbol{c}^* + s\boldsymbol{d})\big|_{s=0} \cdot t + \int_0^t (t-s)\frac{d^2}{ds^2}\varphi(\boldsymbol{c} + s\boldsymbol{d})\,ds$$
$$\geq \varphi(\boldsymbol{c}^*) + 0 \cdot t = \varphi(\boldsymbol{c}^*)$$

for any $t$ or $\boldsymbol{d}$. Thus, the normal equations are sufficient as well as necessary.

#### 2.2.1.1 An Example: Fitting a Quadratic

As an example, suppose we wish to find a least squares quadratic fit to the data in Table 2.2.1. This data was generated in MATLAB from the quadratic function $q(x) = x^2 - 3x + 1$ for $0 \leq x \leq 2$ with pseudo-random noise added. This data leads to the linear system

$$\begin{bmatrix} 20.0000 & 19.4807 & 26.3727 \\ 19.4807 & 26.3727 & 40.5860 \\ 26.3727 & 40.5860 & 66.5787 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -12.8015 \\ -19.7733 \\ -29.6592 \end{bmatrix} \quad \text{with solution}$$
$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} +1.0113 \\ -3.1477 \\ +1.0727 \end{bmatrix}.$$

**Table 2.2.1** Data for quadratic fit

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $x_i$ | 0.9569 | 0.6410 | 1.2032 | 1.8263 | 1.3650 | 1.8935 | 0.1982 | 1.0221 | 0.2203 | 1.0905 |
| $y_i$ | −1.0774 | −0.5767 | −1.2611 | −1.2107 | −1.3714 | −1.0651 | 0.4468 | −1.0731 | 0.3227 | −1.1482 |
| $x_i$ | 1.3776 | 0.2948 | 1.5551 | 0.7981 | 1.7966 | 0.6141 | 0.1221 | 0.4389 | 0.1657 | 1.9008 |
| $y_i$ | −1.2489 | 0.1013 | −1.2266 | −0.7618 | −1.1791 | −0.4325 | 0.6659 | −0.1414 | 0.5320 | −1.0960 |

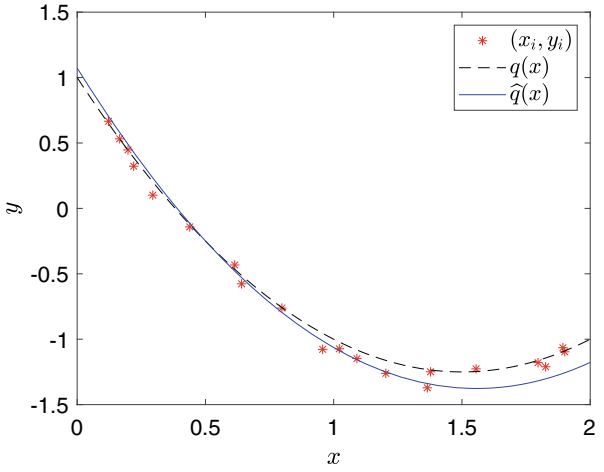



**Fig. 2.2.1** Quadratic fit for data in Table 2.2.1

The solution gives the quadratic fit $\widehat{q}(x)$ compared to $q(x)$ used to create the data, as shown in Figure 2.2.1.

The matrix $A^T A$ in the normal equations has a number of important properties:

- $A^T A$ is symmetric: $(A^T A)^T = A^T A^{TT} = A^T A$.
- $A^T A$ is *positive semi-definite*: $z^T A^T A z = (Az)^T (Az) = \|Az\|_2^2 \geq 0$ for any $z$.

Furthermore, if the columns of $A$ are linearly independent, then for $z \neq \mathbf{0}$, $Az \neq \mathbf{0}$ as well, and so $z^T A^T A z = \|Az\|_2^2 > 0$. That is, $A^T A$ is *positive definite*. Thus, provided the columns of $A$ are linearly independent, $A^T A$ is invertible.

### 2.2.1.2 Perturbations, Conditioning, etc.

To see how sensitive the least squares problem is to perturbations in the data, consider the problem of minimizing $\|(A + E)\widehat{\boldsymbol{x}} - (\boldsymbol{y} + \boldsymbol{z})\|_2$ over $\widehat{\boldsymbol{x}}$. We would like to have some analogy to the condition number for rectangular matrices. Since least squares problems are tied to the 2-norm, we generalize $\kappa_2(A)$ to rectangular matrices. If $A$ has linearly independent columns, then $A^T A$ is invertible, and the solution of the least squares problem is $\boldsymbol{x} = (A^T A)^{-1} A^T \boldsymbol{y}$. The $n \times m$ matrix

$$A^+ = (A^T A)^{-1} A^T \tag{2.2.4}$$

is the *pseudo-inverse* of $A$, provided the columns of $A$ are linearly independent. For solving a system of linear equations $B\boldsymbol{x} = \boldsymbol{b}$, the solution is given by $\boldsymbol{x} = B^{-1}\boldsymbol{b}$, so the condition number of $B$ is $\kappa(B) = \|B\| \left\|B^{-1}\right\|$ using the appropriate matrix norm; the solution of the least squares problem $\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is given by $\boldsymbol{x} = A^+\boldsymbol{b}$ so we define the least squares condition number

$$\kappa_2(A) = \|A\|_2 \left\|A^+\right\|_2 . \tag{2.2.5}$$

If $A$ is square with linearly independent columns, then $A$ is invertible, and

$$A^+ = (A^T A)^{-1} A^T = A^{-1} A^{-T} A^T = A^{-1}.$$

That is, for square invertible matrices, the pseudo-inverse is the ordinary inverse, and the least squares condition number is the ordinary condition number. The least squares condition number also satisfies

$$\kappa_2(A) = \|A\|_2 \left\|A^+\right\|_2 \geq \left\|A^+ A\right\|_2 = \left\|(A^T A)^{-1} A^T A\right\|_2 = \|I\|_2 = 1.$$

**Lemma 2.8** *If $A$ has linearly independent columns and $\|E\|_2 \left\|A^+\right\|_2 < 1$ then $A + E$ also has linearly independent columns.*

This generalizes a result that comes out of the proof of the perturbation theorem for linear systems (Theorem 2.1): if $A$ is invertible and $\|E\| \left\|A^{-1}\right\| < 1$, then $A + E$ is invertible.

***Proof*** If $A$ has linearly independent columns, then the solution of the least squares problem $\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2$ is $\boldsymbol{x} = A^+\boldsymbol{b}$. Thus the solution of $\min_{\widehat{\boldsymbol{x}}} \|(A + E)\widehat{\boldsymbol{x}} - \boldsymbol{y}\|_2$ is the solution of $\min_{\widehat{\boldsymbol{x}}} \|A\widehat{\boldsymbol{x}} - (\boldsymbol{y} - E\boldsymbol{x})\|_2$ where $\boldsymbol{x} = \widehat{\boldsymbol{x}}$. That is, $\widehat{\boldsymbol{x}} = A^+(\boldsymbol{y} - E\widehat{\boldsymbol{x}})$, or equivalently $(I + A^+E)\widehat{\boldsymbol{x}} = \boldsymbol{y}$. This is invertible if $\left\|A^+E\right\|_2 \leq \left\|A^+\right\|_2 \|E\| < 1$. Thus $\min_{\widehat{\boldsymbol{x}}} \|(A + E)\widehat{\boldsymbol{x}} - \boldsymbol{y}\|_2$ has a unique solution under these conditions, and therefore $A + E$ has linearly independent columns. □

**Theorem 2.9** *(Perturbation theorem for least squares) Suppose that*

$$\begin{aligned} &\boldsymbol{x} \quad \textit{minimizes } \|A\boldsymbol{x} - \boldsymbol{b}\|_2 , \textit{ and} \\ &\widehat{\boldsymbol{x}} \quad \textit{minimizes } \|(A + E)\widehat{\boldsymbol{x}} - (\boldsymbol{b} + \boldsymbol{d})\|_2 , \end{aligned}$$

*where $A$ is $m \times n$ with linearly independent columns. If*

$$\epsilon = \max\left\{\frac{\|E\|}{\|A\|}, \frac{\|\boldsymbol{d}\|}{\|\boldsymbol{b}\|}\right\} < \frac{1}{\kappa_2(A)}, \quad \textit{and} \quad \sin\theta = \frac{\|A\boldsymbol{x} - \boldsymbol{b}\|_2}{\|\boldsymbol{b}\|_2} \neq 1,$$

*with $\boldsymbol{b} \neq \boldsymbol{0}$, then*

$$\frac{\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} \leq \epsilon \left[ \frac{2\,\kappa_2(A)}{\cos\theta} + \tan\theta\,\kappa_2(A)^2 \right] + \mathcal{O}(\epsilon^2).$$

*Furthermore, if* $\boldsymbol{r} = A\boldsymbol{x} - \boldsymbol{b}$ *and* $\widehat{\boldsymbol{r}} = A\widehat{\boldsymbol{x}} - \boldsymbol{b}$*, then*

$$\frac{\|\widehat{\boldsymbol{r}} - \boldsymbol{r}\|_2}{\|\boldsymbol{r}\|_2} \leq \epsilon\,(1 + 2\,\kappa_2(A))\ \min(1, m-n) + \mathcal{O}(\epsilon^2).$$

Before we give the proof, we note that if $B$ is a symmetric positive-definite matrix, there is a symmetric positive-definite matrix $C$ where $C^2 = B$. We write $C = B^{1/2}$. This is because there is an orthogonal matrix $Q$ where

(2.2.6)

$$B = Q^T \begin{bmatrix} \lambda_1 & & & & \\ & \lambda_2 & & & \\ & & \lambda_3 & & \\ & & & \ddots & \\ & & & & \lambda_n \end{bmatrix} Q, \text{ so we can set } C = Q^T \begin{bmatrix} \lambda_1^{1/2} & & & & \\ & \lambda_2^{1/2} & & & \\ & & \lambda_3^{1/2} & & \\ & & & \ddots & \\ & & & & \lambda_n^{1/2} \end{bmatrix} Q.$$

We say that $C$ is the symmetric positive-definite *square root* of $B$.

**Lemma 2.10** *If* $A$ *has linearly independent column then* $\left\|A^+\right\|_2^2 = \left\|(A^TA)^{-1}\right\|_2$.

***Proof*** First note that $A^+A = (A^TA)^{-1}A^T\,A = I$. On the other hand, $A\,A^+ = A(A^TA)^{-1}A^T$ is an $m \times m$ matrix, which cannot be the identity matrix if $m > n$: the rank of $A\,A^+$ cannot be more than the rank of $(A^TA)^{-1}$ which is $n$. However, $P = A\,A^+$ is a projection: $P^2 = (A\,A^+)(A\,A^+) = A(A^+A)A^+ = A\,I\,A^+ = A\,A^+ = P$. Furthermore,

$$P^T = (A(A^TA)^{-1}A^T)^T = A^{TT}(A^TA)^{-1}A^T = A(A^TA)^{-1}A^T = P$$

since $A^TA$ and $(A^TA)^{-1}$ are symmetric. Since $P$ is symmetric, it has real eigenvalues with a basis of orthogonal eigenvectors. Since $P^2 = P$, every eigenvalue of $P$ satisfies $\lambda^2 = \lambda$, that is, $\lambda = 1$ or $\lambda = 0$.

We show that $\left\|A^+\right\|_2^2 = \left\|(A^TA)^{-1}\right\|_2$:

$$\left\|A^+\right\|_2^2 = \max_{z \neq \mathbf{0}} \frac{\left\|A^+z\right\|_2^2}{\|z\|_2^2} = \max_{z \neq \mathbf{0}} \frac{z^TA(A^TA)^{-1}(A^TA)^{-1}A^Tz}{z^Tz}.$$

But $P = A(A^TA)^{-1}A^T$ is a symmetric projection matrix. In fact, it is the orthogonal projection onto range$(A)$. To see this, suppose $z \in$ range$(A)$; write $z = A\boldsymbol{w}$. Then $Pz = A(A^TA)^{-1}A^T(A\boldsymbol{w}) = A(A^TA)^{-1}(A^TA)\boldsymbol{w} = A\boldsymbol{w} = z$. On the other hand, if $z$ is orthogonal to range$(A)$, then $z$ is orthogonal to every column of $A$ and so $A^Tz = (z^TA)^T = \mathbf{0}$; thus $Pz = A(A^TA)^{-1}(A^Tz) = \mathbf{0}$.

Write $z = u + v$ where $u \in \text{range}(A)$ and $v$ is orthogonal to range$(A)$. Then $z^T z = u^T u + v^T v$; also $z^T P z = (u+v)^T u = u^T u$, and $A^T z = A^T u$. Then

$$\begin{aligned}
\left\|A^+\right\|_2^2 &= \sup_{u\neq 0, v\neq 0} \frac{(u+v)^T A(A^TA)^{-2}A^T(u+v)}{(u+v)^T(u+v)} \\
&= \sup_{u\neq 0, v\neq 0} \frac{u^T A(A^TA)^{-2}A^T u}{u^T u + v^T v} \\
&= \sup_{u\neq 0, v\neq 0} \frac{u^T A(A^TA)^{-2}A^T u}{u^T u} \frac{u^T u}{u^T u + v^T v} \\
&= \sup_{u\neq 0} \frac{u^T A(A^TA)^{-2}A^T u}{u^T P u} \\
&= \sup_{u\neq 0} \frac{u^T A(A^TA)^{-2}A^T u}{u^T A(A^TA)^{-1}A^T u}
\end{aligned}$$

If we put $w = (A^TA)^{-1/2}A^T u$ then $u^T A(A^TA)^{-2}A^T u = w^T(A^TA)^{-1}w$. On the other hand, $w^T w = u^T A(A^TA)^{-1}A^T u$. Then

$$\left\|A^+\right\|_2^2 = \sup_{w\neq 0} \frac{w^T(A^TA)^{-1}w}{w^T w}.$$

The maximum value of the ratio $w^T B w / w^T w$ for a symmetric matrix $B$ is the maximum eigenvalue of $B$. If $B$ is also positive definite, then the maximum value is $\|B\|_2$. So by (2.1.14) and (2.1.15),

$$\left\|A^+\right\|_2^2 = \lambda_{max}((A^TA)^{-1}) = \left\|(A^TA)^{-1}\right\|_2,$$

as we wanted. □

Now we can continue the proof of the perturbation theorem for least squares, Theorem 2.9.

***Proof*** (Proof of Theorem 2.9.) This proof follows [105, Sec. 5.3.7, p. 242]. Let $\widehat{E} = E/\epsilon$ and $\widehat{d} = d/\epsilon$. Since $\|E\|_2 < 1/\left\|A^+\right\|_2$, and so by Lemma 2.8, $A + t\widehat{E}$ has linearly independent columns for all $0 \le t \le \epsilon$. Thus

$$(A + t\widehat{E})^T(A + t\widehat{E})x(t) = (A + t\widehat{E})^T(b + t\widehat{d}) \tag{2.2.7}$$

is continuously differentiable for $0 \le t \le \epsilon$. Note that $x = x(0)$ and $\widehat{x} = x(\epsilon)$, so

$$\widehat{x} = x + \epsilon \frac{dx}{dt}(0) + \mathcal{O}(\epsilon^2).$$

Let $v = dx/dt\,(0)$. Since $b \neq 0$ and $\sin\theta \neq 1$ we have $x \neq 0$, and

$$\frac{\|\widehat{\boldsymbol{x}}-\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}=\epsilon\frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{x}\|_2}+\mathcal{O}(\epsilon^2).$$

Differentiating (2.2.7) with respect to $t$ at $t=0$ gives

$$\begin{aligned}\widehat{E}^T A\boldsymbol{x}+A^T\widehat{E}\boldsymbol{x}+A^T A\boldsymbol{v}&=A^T\widehat{\boldsymbol{d}}+\widehat{E}^T\boldsymbol{b};\qquad\text{that is,}\\ \boldsymbol{v}&=(A^TA)^{-1}A^T(\widehat{\boldsymbol{d}}-\widehat{E}\boldsymbol{x})-(A^TA)^{-1}\widehat{E}^T\boldsymbol{r}.\end{aligned}$$

Taking norms, we get bounds on $\boldsymbol{v}$:

$$\|\boldsymbol{v}\|_2\le\left\|(A^TA)^{-1}A^T\right\|_2\left(\left\|\widehat{\boldsymbol{d}}\right\|_2+\left\|\widehat{E}\right\|_2\|\boldsymbol{x}\|_2\right)+\left\|(A^TA)^{-1}\right\|_2\left\|\widehat{E}\right\|_2\|\boldsymbol{r}\|_2.$$

Since $\epsilon=\max\{\|\boldsymbol{d}\|_2/\|\boldsymbol{b}\|_2,\,\|E\|_2/\|A\|_2\}$, using $\left\|\widehat{\boldsymbol{d}}\right\|_2=\|\boldsymbol{d}\|_2/\epsilon\le\|\boldsymbol{b}\|_2$ and $\left\|\widehat{E}\right\|_2=\|E\|_2/\epsilon\le\|A\|_2$ gives

$$\begin{aligned}\frac{\|\widehat{\boldsymbol{x}}-\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}&=\epsilon\frac{\|\boldsymbol{v}\|_2}{\|\boldsymbol{x}\|_2}+\mathcal{O}(\epsilon^2)\\ &\le\epsilon\left\{\|A\|_2\left\|A^+\right\|_2\left(\frac{\|\boldsymbol{b}\|_2}{\|A\|_2\|\boldsymbol{x}\|_2}+1\right)\right.\\ &\qquad\left.+\frac{\|\boldsymbol{r}\|_2}{\|A\|_2\|\boldsymbol{x}\|_2}\|A\|_2^2\left\|(A^TA)^{-1}\right\|_2\right\}+\mathcal{O}(\epsilon^2).\end{aligned}$$

Since $A^T(A\boldsymbol{x}-\boldsymbol{b})=\boldsymbol{0}$, we see that $\boldsymbol{x}^TA^T(A\boldsymbol{x}-\boldsymbol{b})=0$, so $A\boldsymbol{x}$ is perpendicular to $A\boldsymbol{x}-\boldsymbol{b}$. Then

$$\begin{aligned}\|\boldsymbol{b}\|_2^2=\|A\boldsymbol{x}\|_2^2+\|A\boldsymbol{x}-\boldsymbol{b}\|_2^2&=\|A\boldsymbol{x}\|_2^2+\|\boldsymbol{r}\|_2^2\qquad\text{and so}\\ \|A\|_2^2\|\boldsymbol{x}\|_2^2\ge\|A\boldsymbol{x}\|_2^2&\ge\|\boldsymbol{b}\|_2^2-\|\boldsymbol{r}\|_2^2.\end{aligned}$$

But $\|\boldsymbol{r}\|_2=\sin\theta\,\|\boldsymbol{b}\|_2$, so $\|\boldsymbol{b}\|_2^2-\|\boldsymbol{r}\|_2^2=\|\boldsymbol{b}\|_2^2\cos^2\theta\le\|A\|_2^2\|\boldsymbol{x}\|_2^2$. This means $\|\boldsymbol{b}\|_2/(\|A\|_2\|\boldsymbol{x}\|_2)\le1/\cos\theta$ and $\|\boldsymbol{r}\|_2/(\|A\|_2\|\boldsymbol{x}\|_2)\le\sin\theta/\cos\theta=\tan\theta$. Since $\left\|(A^TA)^{-1}\right\|_2=\left\|A^+\right\|_2^2$ by Lemma 2.10, $\|A\|_2^2\left\|(A^TA)^{-1}\right\|_2=\|A\|_2^2\left\|A^+\right\|_2^2=\kappa_2(A)^2$. Substituting these above gives

$$\frac{\|\widehat{\boldsymbol{x}}-\boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2}\le\epsilon\left\{\kappa_2(A)\left(\frac{1}{\cos\theta}+1\right)+\tan\theta\,\kappa_2(A)^2\right\}+\mathcal{O}(\epsilon^2).$$

For the residuals, let

$$\boldsymbol{r}(t)=(\boldsymbol{b}+t\widehat{\boldsymbol{d}})-(A+t\widehat{E})\boldsymbol{x}(t).$$

Differentiating with respect to $t$ and taking $t=0$ give

$$\boldsymbol{s}:=\frac{d\boldsymbol{r}}{dt}(0)=(I-A\,A^+)(\widehat{\boldsymbol{d}}-\widehat{E}\boldsymbol{x})-A(A^TA)^{-1}\widehat{E}\boldsymbol{r}.$$

So

$$\begin{aligned}\frac{\|\widehat{\boldsymbol{r}}-\boldsymbol{r}\|_2}{\|\boldsymbol{b}\|_2} &= \epsilon\frac{\|\boldsymbol{s}\|_2}{\|\boldsymbol{b}\|_2}+\mathcal{O}(\epsilon^2)\\ &\le \epsilon\left\{\left\|I-A\,A^+\right\|_2\left(\frac{\|A\|_2\,\|\boldsymbol{x}\|_2}{\|\boldsymbol{b}\|_2}+1\right)\right.\\ &\quad\left.+\left\|A(A^TA)^{-1}\right\|_2\|A\|_2\frac{\|\boldsymbol{r}\|_2}{\|\boldsymbol{b}\|_2}\right\}+\mathcal{O}(\epsilon^2).\end{aligned}$$

Note that $I - A\,A^+ = I - P$ is a symmetric projection as $P = A\,A^+$ is a symmetric projection: $(I-P)^2 = I - 2P + P^2 = I - 2P + P = I - P$. In this case, $P$ is the orthogonal projection onto range$(A)$; then $I - P$ is the orthogonal projection onto the orthogonal complement of range$(A)$. Thus $\left\|I - A\,A^+\right\|_2 = 0$ if range$(A) = \mathbb{R}^m$, and one otherwise. That is, $\left\|I - A\,A^+\right\|_2 = \min(m-n,\,1)$. Note that $\|A\|_2\,\|\boldsymbol{x}\|_2 \le \|A\|_2\left\|A^+\boldsymbol{b}\right\|_2 \le \kappa_2(A)\,\|\boldsymbol{b}\|_2$, and $\|\boldsymbol{r}\|_2 = \|\boldsymbol{b} - A\boldsymbol{x}\|_2 = \left\|(I - AA^+)\boldsymbol{b}\right\| \le \min(m-n,\,1)\,\|\boldsymbol{b}\|_2$. Then

$$\frac{\|\widehat{\boldsymbol{r}}-\boldsymbol{r}\|_2}{\|\boldsymbol{b}\|_2} \le \epsilon\,\{\min(m-n,\,1)\,(1+2\,\kappa_2(A))\}+\mathcal{O}(\epsilon^2),$$

as we wanted. □

An important aspect to note is that if $\boldsymbol{r} = \boldsymbol{0}$ so that $A\boldsymbol{x} = \boldsymbol{b}$, then the bound on $\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|_2 / \|\boldsymbol{x}\|_2$ is linear in the condition number $\kappa_2(A)$, but if $\|\boldsymbol{r}\|_2 / \|\boldsymbol{b}\|_2$ is *not* small, then the bound grows like $\kappa_2(A)^2$.

#### 2.2.1.3 Cholesky Factorization and Normal Equations

A common method to solve the normal equations is to use the Cholesky factorization of $A^TA = LL^T$ (see Section 2.1.5.2). Provided the columns of $A$ are linearly independent, then $A^TA$ is positive definite and $L$ is invertible. Then $I = L^{-1}A^TAL^{-T} = \left(AL^{-T}\right)^T\left(AL^{-T}\right)$ so $\widetilde{Q} = AL^{-T}$ satisfies $\widetilde{Q}^T\widetilde{Q} = I$. However, this $\widetilde{Q}$ is not orthogonal in general, because if $A$ is $m \times n$ then $\widetilde{Q}$ is $m \times n$ and is not necessarily square. However, if $\widetilde{\boldsymbol{q}}_i$ is the $i$th column of $\widetilde{Q}$ and since $\widetilde{Q}^T\widetilde{Q} = I$, then $\widetilde{\boldsymbol{q}}_i^T\widetilde{\boldsymbol{q}}_j = 1$ if $i = j$ and zero if $i \ne j$. That is, the columns of $\widetilde{Q}$ are orthonormal. Also,

$$A = \widetilde{Q}\,L^T.$$

This is a version of the QR factorization, which we consider in the next section.

We can try to use the results of the Wilkinson backward error theorem (Theorem 2.5) and its supporting lemmas (Lemmas 2.3 and 2.4) to provide a backward error result for using normal equations and Cholesky factorization. If we use a standard matrix multiplication algorithm, then the computed value of $A^TA$ is

$$fl(A^T A) = (A + E)^T A, \qquad |E| \le \gamma_m |A|,$$

where $\gamma_m = m\mathbf{u} + \mathcal{O}(m\mathbf{u}^2)$ and $\mathbf{u}$ is unit roundoff by Lemma 2.4. Note that $|A|$ is the matrix of absolute values $|a_{ij}|$. There is no guarantee that $fl(A^T A)$ is symmetric or positive definite even if the columns of $A$ are linearly independent. For example, consider

$$A^T A = \left[\begin{array}{c|c} 1 & 1+\eta \\ \hline 0 & \eta \end{array}\right]^T \left[\begin{array}{c|c} 1 & 1+\eta \\ \hline 0 & \eta \end{array}\right] = \left[\begin{array}{c|c} 1 & 1+\eta \\ \hline 1+\eta & 1+2\eta+2\eta^2 \end{array}\right].$$

If $\eta$ is a power of two and $\mathbf{u} \le \eta < \frac{1}{2}\sqrt{\mathbf{u}}$, then $fl(1 + 2\eta + 2\eta^2) = 1 + 2\eta$ in IEEE arithmetic. In this case, $\det fl(A^T A) = (1 + 2\eta) - (1 + \eta)^2 = -\eta^2$ so $fl(A^T A)$ must have a negative eigenvalue.

The roundoff errors incurred by Cholesky factorization can also turn a matrix from being positive definite to positive semi-definite or neither positive definite nor positive semi-definite. For example, $B = \left[\begin{array}{c|c} 3+2\mathbf{u} & 3 \\ \hline 3 & 3 \end{array}\right]$ is positive definite, but its Cholesky factorization fails in IEEE double precision as the computed value of $a_{22} - (a_{21}/\sqrt{a_{11}})^2$ is exactly zero.

However, *if the Cholesky factorization succeeds*, Theorem 2.5 can be adapted to show that the computed solution $\widehat{\boldsymbol{x}}$ satisfies

$$(A^T A + E)\widehat{\boldsymbol{x}} = (A + F)^T \boldsymbol{b}$$

with $|E| \le \mathcal{O}(m\mathbf{u})\,|A|^T\,|A|$ and $|F| \le \mathcal{O}(m\mathbf{u})\,|A|^T\,|\boldsymbol{b}|$. The perturbation theorem for linear systems (Theorem 2.1) then gives the relative error in the solution

$$\frac{\|\widehat{\boldsymbol{x}} - \boldsymbol{x}\|_2}{\|\boldsymbol{x}\|_2} = \mathcal{O}(m\mathbf{u})\,\kappa_2(A^T A) = \mathcal{O}(m\mathbf{u})\,\kappa_2(A)^2.$$

The fact that we have the *square* of the condition number can be cause for concern for ill-conditioned problems. The QR factorization of the following section gives a way of improving this for situations where the residual is small compared to the right-hand side: $\|\boldsymbol{r}\|_2 \ll \|\boldsymbol{b}\|_2$.

### 2.2.2 *QR Factorization*

The QR factorization of an $m \times n$ matrix $A$ is

$$A = QR, \qquad Q \text{ orthogonal and } R \text{ upper triangular.} \tag{2.2.8}$$

Note that $Q$ is $m \times m$ while $R$ is $m \times n$. If $m > n$ then we can split both $Q$ and $R$ consistently:

$$A = QR = [Q_1 \mid Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1, \tag{2.2.9}$$

where $Q_1$ is $m \times n$ and has orthonormal columns and $R_1$ is $n \times n$ and upper triangular. This is called the *skinny*, *thin*, or *reduced QR factorization* of $A$.

Recall that a matrix $Q$ is *orthogonal* if $Q^T = Q^{-1}$, or equivalently, $Q$ is square and $Q^T Q = I$. Note that orthogonal matrices preserve the 2-norms of vectors:

$$\|Qz\|_2^2 = (Qz)^T (Qz) = z^T Q^T Q z = z^T I z = z^T z = \|z\|_2^2 . \tag{2.2.10}$$

**Lemma 2.11** *If $A$ has linearly independent columns, then $R_1$ is invertible.*

***Proof*** If $A$ has linearly independent columns, then $A\boldsymbol{x} = \boldsymbol{0}$ implies $\boldsymbol{x} = \boldsymbol{0}$; then $Q_1 R_1 \boldsymbol{x} = \boldsymbol{0}$ implies $\boldsymbol{x} = \boldsymbol{0}$. But $Q_1$ has orthonormal columns so $Q_1$ has linearly independent columns; therefore, $Q_1 R_1 \boldsymbol{x} = \boldsymbol{0}$ implies $R_1 \boldsymbol{x} = \boldsymbol{0}$. Then the only way that $R_1 \boldsymbol{x} = \boldsymbol{0}$ is if $\boldsymbol{x} = \boldsymbol{0}$. Since $R_1$ is square, this means that $R_1$ is invertible. That is, if $A$ has linearly independent columns, then $R_1$ is invertible. □

If $A$ is complex, then the QR factorization of $A$ is $A = QR$ where $Q$ is unitary ($Q^{-1} = \overline{Q}^T$) and $R$ is upper triangular.

#### 2.2.2.1 Using QR Factorizations

QR factorizations are most frequently used to solve least squares problems. Consider the problem

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2$$

with $A$ an $m \times n$ matrix with $m > n$. Then

$$\begin{aligned} \|A\boldsymbol{x} - \boldsymbol{b}\|_2 &= \|QR\boldsymbol{x} - \boldsymbol{b}\|_2 = \left\|Q(R\boldsymbol{x} - Q^T\boldsymbol{b})\right\|_2 \\ &= \left\|R\boldsymbol{x} - Q^T\boldsymbol{b}\right\|_2 \qquad \text{by (2.2.10)} \\ &= \left\| \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \boldsymbol{x} - \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \boldsymbol{b} \right\|_2 \\ &= \sqrt{\left\|R_1\boldsymbol{x} - Q_1^T\boldsymbol{b}\right\|_2^2 + \left\|Q_2^T\boldsymbol{b}\right\|_2^2}. \end{aligned}$$

Assuming that $A$ has linearly independent columns, so that $R_1$ is invertible by Lemma 2.11, we minimize $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ by setting $R_1\boldsymbol{x} - Q_1^T\boldsymbol{b} = \boldsymbol{0}$. That is, we set $\boldsymbol{x} = R_1^{-1} Q_1^T \boldsymbol{b}$, and

$$\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2 = \left\|Q_2^T\boldsymbol{b}\right\|_2 = \sqrt{\|\boldsymbol{b}\|_2^2 - \left\|Q_1^T\boldsymbol{b}\right\|_2^2}.$$

Note that solving the least squares problem only requires the reduced QR factorization.

There are some useful properties of these matrices: since $Q^T = Q^{-1}$,

$$I = Q^T Q = \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} = \begin{bmatrix} Q_1^T Q_1 & Q_1^T Q_2 \\ Q_2^T Q_1 & Q_2^T Q_2 \end{bmatrix} = \begin{bmatrix} I & 0 \\ 0 & I \end{bmatrix}$$

and so $Q_1^T Q_1 = I$, $Q_2^T Q_2 = I$ while $Q_1^T Q_2 = 0$. Also,

$$I = QQ^T = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} Q_1^T \\ Q_2^T \end{bmatrix} = Q_1 Q_1^T + Q_2 Q_2^T.$$

**Lemma 2.12** *Suppose $A$ has linearly independent columns. Then the matrix $Q_1 Q_1^T$ is the orthogonal projection onto range$(A)$, while $Q_2 Q_2^T$ is the orthogonal projection onto range$(A)^{\perp}$.*

***Proof*** Suppose $z \in \text{range}(A)$, so that $z = A\boldsymbol{u}$ for some unique $\boldsymbol{u}$. Then $\|A\boldsymbol{x} - z\|_2$ is minimized by $\boldsymbol{x} = \boldsymbol{u} = R_1^{-1} Q_1^T z$. That is, $R_1 \boldsymbol{u} = Q_1^T z$. Pre-multiplying by $Q_1$ gives $Q_1 Q_1^T z = Q_1 R_1 \boldsymbol{u} = A\boldsymbol{u} = z$.

Now suppose that $z$ is orthogonal to range$(A)$. Then $A^T z = \mathbf{0}$. Substituting using the reduced QR factorization gives $R_1^T Q_1^T z = \mathbf{0}$. Since $A$ has linearly independent columns, $R_1$ is invertible, and so $Q_1^T z = \mathbf{0}$. Therefore, $Q_1 Q_1^T z = Q_1 \mathbf{0} = \mathbf{0}$.

Regarding $Q_2 Q_2^T$, we note that $Q_2 Q_2^T = I - Q_1 Q_1^T$ so $Q_2 Q_2^T$ is the complementary projection to $Q_1 Q_1^T$. If $z \in \text{range}(A)$ then $Q_2 Q_2^T z = (I - Q_1 Q_1^T) z = z - Q_1 Q_1^T z = z - z = \mathbf{0}$ while if $z$ is orthogonal to range$(A)$, $Q_2 Q_2^T z = (I - Q_1 Q_1^T) z = z - Q_1 Q_1^T z = z - \mathbf{0} = z$. □

#### 2.2.2.2 Gram–Schmidt Orthogonalization

A form of the QR factorization was developed with the work of Erhard Schmidt [230] (1907) citing the method of Jorgen Pedersen Gram [108] (1883), although Laplace had already used this idea by 1820. (See Laplace's Supplement *Sur l'application du calcul des probabilités á là philosophie naturelle* to [155]. Translation and modern interpretation can be found in [153].)

Given a sequence of vectors $\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \ldots, \boldsymbol{a}_k$ that are linearly independent, we wish to produce a sequence $\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \ldots, \boldsymbol{q}_k$ of vectors that are either orthogonal or orthonormal, and span $\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\} = \text{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\}$ for $j = 1, 2, \ldots, k$. We will focus on making the $\boldsymbol{q}_j$'s orthonormal.

The Gram–Schmidt process is shown in Algorithm 14.

**Algorithm 14** Gram–Schmidt process

```
1  function gramschmidt(a_1, a_2, ..., a_k)
2    for j = 1, 2, ..., k
3      for i = 1, 2, ..., j − 1: r_ij ← a_j^T q_i end
4      b_j ← a_j − Σ_{i=1}^{j−1} r_ij q_i
5      r_jj ← ‖b_j‖_2
6      q_j ← b_j / r_jj
7    end for
8    return (q_1, q_2, ..., q_k)
9  end function
```

**Theorem 2.13** *The Gram–Schmidt process produces a sequence* $\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k$ *where*

$$\boldsymbol{q}_i \boldsymbol{q}_j = \begin{cases} 1, & \text{if } i = j, \\ 0, & \text{if } i \neq j, \end{cases} \qquad \text{for } i, j = 1, 2 \ldots, k,$$

$$\|\boldsymbol{q}_i\|_2 = 1 \quad \text{for } i = 1, 2, \ldots, k, \quad \text{and}$$

$$\text{span}\{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_j\} = \text{span}\{\boldsymbol{q}_1, \ldots, \boldsymbol{q}_j\} \quad \text{for } j = 1, 2, \ldots, k.$$

***Proof*** We proceed by induction on $k$. The base case for $k = 1$ just requires that $\|\boldsymbol{q}_1\|_2 = 1$, which is guaranteed by lines 5 and 6.

Suppose that the statement is true for $k = p$. We show that the statement is true for $k = p + 1$.

To see that Algorithm 14 does indeed produce an orthonormal basis, we note that

$$\boldsymbol{b}_{p+1} = \boldsymbol{a}_{p+1} - \sum_{i=1}^{p} (\boldsymbol{a}_j^T \boldsymbol{q}_i) \boldsymbol{q}_i \qquad \text{from lines 3 and 4.}$$

Then for $\ell < p + 1$,

$$\begin{aligned} \boldsymbol{q}_\ell^T \boldsymbol{b}_{p+1} &= \boldsymbol{q}_\ell^T \boldsymbol{a}_{p+1} - \sum_{i=1}^{p} (\boldsymbol{a}_{p+1}^T \boldsymbol{q}_i) \boldsymbol{q}_\ell^T \boldsymbol{q}_i \\ &= \boldsymbol{q}_\ell^T \boldsymbol{a}_{p+1} - \sum_{i=1}^{j-1} (\boldsymbol{a}_{p+1}^T \boldsymbol{q}_i) \begin{cases} 1, & \text{if } \ell = i, \\ 0, & \text{if } \ell \neq i \end{cases} \\ &= \boldsymbol{q}_\ell^T \boldsymbol{a}_{p+1} - \boldsymbol{a}_{p+1}^T \boldsymbol{q}_\ell = 0. \end{aligned}$$

That is, $\boldsymbol{b}_{p+1}$ is orthogonal to span $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_p\}$. Since $\boldsymbol{q}_{p+1} = \boldsymbol{b}_{p+1} / \|\boldsymbol{b}_{p+1}\|_2$, we see that $\boldsymbol{q}_{p+1}$ has unit length and is orthogonal to span $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_p\} =$ span $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_p\}$; thus $\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_p, \boldsymbol{q}_{p+1}\}$ is an orthonormal set. Furthermore,

$$\boldsymbol{q}_{p+1} = \left( \boldsymbol{a}_{p+1} - \sum_{i=1}^{p} (\boldsymbol{a}_j^T \boldsymbol{q}_i) \boldsymbol{q}_i \right) / \left\| \boldsymbol{b}_{p+1} \right\|_2, \qquad \text{and}$$

$$\boldsymbol{a}_{p+1} = \left\| \boldsymbol{b}_{p+1} \right\|_2 \boldsymbol{q}_{p+1} + \sum_{i=1}^{p} (\boldsymbol{a}_{p+1}^T \boldsymbol{q}_i) \boldsymbol{q}_i, \tag{2.2.11}$$

so span $\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_{p+1}\} = \text{span}\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_{p+1}\}$.

Thus, by the principle of induction, the conclusions we want hold for $k = 1, 2, 3, \ldots$. □

Suppose that $\boldsymbol{a}_j$ is column $j$ of an $m \times n$ matrix $A$. Then Equation (2.2.11) means that

$$\begin{aligned} \boldsymbol{a}_j &= [\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3, \cdots, \boldsymbol{q}_j] \begin{bmatrix} \boldsymbol{a}_j^T \boldsymbol{q}_1 \\ \boldsymbol{a}_j^T \boldsymbol{q}_2 \\ \boldsymbol{a}_j^T \boldsymbol{q}_3 \\ \vdots \\ \left\| \boldsymbol{b}_j \right\|_2 \end{bmatrix} \\ &= [\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_j, \boldsymbol{q}_{j+1}, \cdots, \boldsymbol{q}_n] \begin{bmatrix} \boldsymbol{a}_j^T \boldsymbol{q}_1 \\ \boldsymbol{a}_j^T \boldsymbol{q}_2 \\ \vdots \\ \left\| \boldsymbol{b}_j \right\|_2 \\ 0 \\ \vdots \\ 0 \end{bmatrix}. \end{aligned} \tag{2.2.12}$$

The column in (2.2.12) is the $j$th column of an upper triangular matrix we call $R_1$. Putting the columns together

$$A = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n] = \left[\boldsymbol{q}_1, \boldsymbol{q}_2, \cdots, \boldsymbol{q}_n\right] R_1 = Q_1 R_1.$$

This is not the full QR factorization, but rather the *reduced* QR factorization: $Q_1$ is $m \times n$ while $R_1$ is $n \times n$. As noted above, $Q_1$ has orthonormal columns rather than being an orthogonal matrix.

### 2.2.2.3 Modified Versus Classical Gram–Schmidt

Algorithm 14 is not the only way to orthonormalize a family of vectors. In fact, since

$$\boldsymbol{q}_\ell^T (\boldsymbol{a}_j - \sum_{i=1}^{\ell-1} (\boldsymbol{a}_j^T \boldsymbol{q}_i) \boldsymbol{q}_i) = \boldsymbol{q}_\ell^T \boldsymbol{a}_j,$$

**Algorithm 15** Modified Gram–Schmidt process

```
1  function mgs(a_1, a_2, ..., a_k)
2    for j = 1, 2, ..., k
3      b_j ← a_j
4      for i = 1, 2, ..., j − 1
5        r_ij ← b_j^T q_i
6        b_j ← b_j − r_ij q_i
7      end for
8      r_jj ← ‖b_j‖_2
9      q_j ← b_j / r_jj
10   end for
11   return (q_1, q_2, ..., q_k)
12 end function
```

we can change Algorithm 14 to Algorithm 15.

It is easy to confuse the modified with the original (or classical) Gram–Schmidt algorithms. In fact, Laplace used the modified process. While the classical Gram–Schmidt (CGS) and modified Gram–Schmidt (MGS) are identical in exact arithmetic, they behave differently in the presence of roundoff error.

The different behavior between CGS and MGS is discussed in [16], where they show that the $R_{CGS}$ matrix computed by CGS is the exact Cholesky factor of a perturbed normal equations matrix $A^T A + E$ where $\|E\|_2 \leq \mathcal{O}(mn^2)\mathbf{u}\,\|A\|_2^2$. Further, the resulting loss of orthogonality is bounded by $\left\|I - Q_{CGS}^T Q_{CGS}\right\|_2 = \mathcal{O}(mn^2)\mathbf{u}\,\kappa_2(A)^2$. MGS, on the other hand, as noted in [105, Sec 5.2.9, p. 232] has $\left\|I - Q_{MGS}^T Q_{MGS}\right\|_2 = \mathcal{O}(mn^2)\mathbf{u}\,\kappa_2(A)$ (see [23]).

According to [16], the loss of orthogonality can be remedied by repeating the CGS algorithm one more time. That is, if CGS gives $A \approx Q_{CGS}^{(1)} R_{CGS}^{(1)}$, then by computing the QR factorization of $Q_{CGS}^{(1)}$ by CGS once again ($Q_{CGS}^{(1)} \approx Q_{CGS}^{(2)} R_{CGS}^{(2)}$) we get a matrix $Q_{CGS}^{(2)}$ that is much closer to having orthogonal columns: $\left\|I - Q_{CGS}^{(2)\,T} Q_{CGS}^{(2)}\right\|_2 = \mathcal{O}(mn^2)\mathbf{u}$.

This result is of more than theoretical interest. While other algorithms for QR factorization are available, MGS and CGS are more suitable for "online" algorithms where each vector $\boldsymbol{a}_j$ is added when it is available. When a new vector $\boldsymbol{a}_j$ is presented, MGS is inherently a sequential algorithm while CGS can schedule the dot products $r_{ij} = \boldsymbol{a}_j^T \boldsymbol{q}_i$ for $i < j$ in parallel, and then the linear combination $\boldsymbol{b}_j \leftarrow \boldsymbol{b}_j - \sum_{i=1}^{j-1} r_{ij}\boldsymbol{q}_i$ can also be computed in parallel using $\mathcal{O}(m \log j)$ time. A "CGS$^2$" algorithm can be used to obtain a good orthonormal basis efficiently in parallel. However, care should be taken with the $R$ matrix computed by CGS if $A$ is ill-conditioned.

#### 2.2.2.4 Householder Reflectors

In 1958, Alston Householder discovered a way of carrying out QR factorizations with high accuracy [182]. At the time, it was presented as an improvement of Givens'

rotation approach, which is discussed in the next section. We present this through a recursive algorithm for the QR factorization based on *Householder reflectors*. A Householder reflector is a matrix

$$P = I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}. \tag{2.2.13}$$

The matrix $P$ is both symmetric and orthogonal. Symmetry is clear from (2.2.13), but to show orthogonality we need some calculations:

$$\begin{aligned} P^T P = P^2 &= \left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)\left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right) \\ &= I - 4\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} + 4\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\,\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} \\ &= I - 4\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} + 4\frac{\boldsymbol{v}(\boldsymbol{v}^T\boldsymbol{v})\boldsymbol{v}^T}{(\boldsymbol{v}^T\boldsymbol{v})^2} \\ &= I - 4\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} + 4\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}} = I. \end{aligned}$$

The main step we need is, given a vector $\boldsymbol{a} \in \mathbb{R}^n$ to find a $\boldsymbol{v}$ so that the Householder reflector $P$ satisfies $P\boldsymbol{a} = \alpha\boldsymbol{e}_1$ where $\boldsymbol{e}_1 = [1, 0, 0, \ldots, 0]^T$. Since $P$ is orthogonal, $|\alpha| = \|\boldsymbol{a}\|_2$. In real arithmetic, we have $\alpha = \pm\|\boldsymbol{a}\|_2$. If

$$\begin{aligned} \alpha\boldsymbol{e}_1 = P\boldsymbol{a} &= \left(I - 2\frac{\boldsymbol{v}\boldsymbol{v}^T}{\boldsymbol{v}^T\boldsymbol{v}}\right)\boldsymbol{a} \\ &= \boldsymbol{a} - 2\frac{\boldsymbol{v}^T\boldsymbol{a}}{\boldsymbol{v}^T\boldsymbol{v}}\boldsymbol{v} \end{aligned}$$

we let $\gamma = 2\,(\boldsymbol{v}^T\boldsymbol{a})/(\boldsymbol{v}^T\boldsymbol{v})$ and so

$$\alpha\boldsymbol{e}_1 = \boldsymbol{a} - \gamma\boldsymbol{v}.$$

Assuming $\gamma \neq 0$ we can re-arrange this to

$$\gamma\boldsymbol{v} = \boldsymbol{a} - \alpha\boldsymbol{e}_1.$$

Note, however, that scaling $\boldsymbol{v} \mapsto \rho\,\boldsymbol{v}$ does not change $P$. So we can arbitrarily set $\gamma = 1$. In real arithmetic, we then have

$$\boldsymbol{v} = \boldsymbol{a} \pm \|\boldsymbol{a}\|_2\,\boldsymbol{e}_1. \tag{2.2.14}$$

**Algorithm 16** Householder QR factorization (recursive, overwrites $A$)

```
function hhQR(A)
  a ← 1st column of A
  compute v via (2.2.15)
  β ← 2/(v^T v)
  A ← P A
  let A = [ α | b^T ]
          [ 0 | Ã   ]
  if A has one column
    return (P, [ α ]
               [ 0 ])
  end if
  (Q̃, R̃) ← hhQR(Ã)
  Q ← P [ 1  0 ]
        [ 0  Q̃ ];   R ← [ α | b^T ]
                        [ 0 | R̃   ]
  return (Q, R)
end function
```

Once we have this formula, we just need to choose the sign. Since subtracting nearly equal quantities reduces relative accuracy, we choose the sign to be the sign of $a_1$ so that this first component is not subtracted. That is,

$$\boldsymbol{v} = \boldsymbol{a} + \operatorname{sign}(a_1)\ \|\boldsymbol{a}\|_2\, \boldsymbol{e}_1. \tag{2.2.15}$$

For the efficient use of $P$, we can pre-compute $\beta := 2/(\boldsymbol{v}^T\boldsymbol{v})$, so $P = I - \beta\boldsymbol{v}\boldsymbol{v}^T$. Computing $\boldsymbol{v}$ and $\beta$ given $\boldsymbol{a}$ takes $2n+5$ flops. Computing $P\boldsymbol{x} = \boldsymbol{x} - \beta(\boldsymbol{v}^T\boldsymbol{x})\boldsymbol{v}$ given $\boldsymbol{x}$, $\boldsymbol{v}$ and $\beta$ takes $4n+1$ flops.

Usually we avoid storing or computing $Q$ explicitly. Instead we store the $\boldsymbol{v}$ vectors and possibly the $\beta$ values for each Householder reflector. Efficient storage schemes include the compact WY storage method to represent the $Q$ matrix [105, Sec. 5.1.7, pp. 213–215]: $Q = I + WY^T$ with both $W$ and $Y$ being $n \times r$ matrices. Updating $Q_+ = (I - \gamma\boldsymbol{v}\boldsymbol{v}^T)Q$ can be done by setting $Q_+ = I + W_+Y_+^T$ with $W_+ = [W \mid \boldsymbol{v}]$ and $Y_+ = [Y \mid \boldsymbol{z}]$ with $\boldsymbol{z} = -\gamma\,(I + YW^T)\boldsymbol{v}$. This update takes $\mathcal{O}(rn)$ operations. Complementary scaling of $\boldsymbol{v}$ and $\boldsymbol{z}$ can be done to prevent the columns of $W$ and $Y$ become badly scaled.

Using Householder reflectors gives a QR factorization where the computed $\widehat{Q}$ and computed $\widehat{R}$ satisfy

$$\left\|I - \widehat{Q}^T\widehat{Q}\right\|_2 = \mathcal{O}(m\mathbf{u}), \qquad \text{and}$$
$$\left\|A - \widehat{Q}\widehat{R}\right\|_2 = \mathcal{O}(m\mathbf{u})\,\|A\|_2\,.$$

#### 2.2.2.5 Givens' Rotations

An alternative approach to computing QR factorizations is to use Givens' rotations. Two-dimensional rotations have the form

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \qquad c^2 + s^2 = 1.$$

We can use Givens' rotations to shape the non-zeros in a vector:

$$\begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \alpha \\ 0 \end{bmatrix},$$

where $\alpha = \pm\sqrt{x^2 + y^2}$ since rotations preserve the 2-norm. So we need $s\,x + c\,y = 0$. This can be satisfied with

$$s = \frac{-y}{\sqrt{x^2 + y^2}},$$
$$c = \frac{x}{\sqrt{x^2 + y^2}}.$$

The computation of $\sqrt{x^2 + y^2}$ can be done using the `hypot` function to avoid overflow or underflow. By systematically zeroing out entries in $A$ we can create a QR factorization. For this we need to choose indexes $(i, j)$ where

$$G(i, j; c, s) = \begin{bmatrix} \ddots & & & & \vdots \\ & c & \cdots & -s & \text{row } i \\ & \vdots & \ddots & \vdots & \vdots \\ & s & \cdots & c & \text{row } j \\ & & & \ddots & \vdots \end{bmatrix}$$

and is otherwise equal to the identity matrix. Then

$$G(i, j; c, s)_{pq} = \begin{cases} 1, & \text{if } p = q \neq i, j, \\ 0, & \text{if } p \neq q \text{ and } \{p, q\} \cap \{i, j\} = \emptyset, \\ c, & \text{if } (i = p \text{ and } j = q) \text{ or } (i = q \text{ and } j = p), \\ s, & \text{if } p = j \text{ and } q = i, \\ -s, & \text{if } p = i \text{ and } q = j. \end{cases}$$

The QR factorization using Givens' rotation is shown in Algorithm 17.

Rather than save the $(c, s)$ pair for each Givens' rotation, we can save $\theta$ where $c = \cos\theta$ and $s = \sin\theta$, or an equivalent encoding.

From a computational point of view, Givens' rotations are fairly expensive as each requires the computation of a square root, which takes about 10 times as long as a multiplication, addition, or subtraction. Otherwise, it has essentially the same performance and error behavior as for Householder QR.

**Algorithm 17** Givens' rotation QR factorization (overwrites $A$)

```
function givensQR(A)
  for i = 1, 2, ..., n
    for j = i + 1, ..., m
      c ← a_ii / sqrt(a_ii^2 + a_ij^2);  s ← −a_ij / sqrt(a_ii^2 + a_ij^2)
      A ← G(i, j; c, s) A;  Q ← Q G(i, j; c, s)
    end for
  end for
  return (Q, A)
end function
```

**Exercises.**

(1) Generate $N = 20$ values $x_i$ randomly and uniformly in $[0, 1]$, and set $y_i = f(x_i) + \alpha\,\epsilon_i$ with $\epsilon_i$ generated randomly with a standard normal distribution, $f(x) = \frac{1}{2}x^2 - x + 1$, and $\alpha = 0.2$. Plot the data as a scatterplot (no lines joining the points). Solve the least squares problem to find the coefficients of the best fit quadratic (minimize $\sum_{i=1}^{N}(y_i - (ax_i^2 + bx_i + c))^2$) using either normal equations of the QR factorization. Plot $g(x) = ax^2 + bx + c$ for the best fit as well as $f(x)$. How close are $f(x)$ and $g(x)$ over $[0, 1]$?

(2) Generate $N = 20$ vectors $\boldsymbol{x}_i$ randomly and uniformly in $[0, 1]^5 \subset \mathbb{R}^5$, and set $y_i = \boldsymbol{c}_0^T\boldsymbol{x}_i + \alpha\,\epsilon_i$ where $\boldsymbol{c}_0 = [1, 0, -2, \frac{1}{2}, -1]^T$ and $\epsilon_i$ generated by a standard normal distribution, and $\alpha = 0.2$. If $X$ is the data matrix $[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]^T$, solve the least squares $\min_{\boldsymbol{c}} \|X\boldsymbol{c} - \boldsymbol{y}\|_2$ via either normal equations or the QR factorization. Look at $\|\boldsymbol{c} - \boldsymbol{c}_0\|_2$ to see if the least squares estimate is close to the vector generating the data.

(3) For the data matrix $X$ in Exercise 2, replace $X$ with $X + F$ where $F$ is a perturbation matrix and repeat the computations. To generate $F$, set each entry $f_{ij}$ to be $10^{-4}$ times a pseudo-random sample from a standard normal distribution. Compare the change in the solution to the estimate given in Theorem 2.9.

(4) Show that if $A$ is real, square, and invertible, then the QR factorization is unique apart from a diagonal scaling by factors of $\pm 1$. That is, if $A = Q_1R_1 = Q_2R_2$ with $Q_1$, $Q_2$ orthogonal and $R_1$, $R_2$ upper triangular, then there is a diagonal matrix $D$ with diagonal entries $\pm 1$ where $Q_2 = Q_1D$ and $DR_2 = R_1$. In particular, show that if the diagonal entries of $R_1$ and $R_2$ are all positive, then $Q_1 = Q_2$ and $R_1 = R_2$.

(5) Show partial uniqueness for "thin" QR factorizations of $A$ an $m \times n$ real matrix with $m > n$ provided $A$ has linearly independent columns. Suppose $A = Q_1R_1 = Q_2R_2$ where $Q_1$ and $Q_2$ are $m \times n$ with orthonormal columns ($Q_1^TQ_1 = Q_2^TQ_2 = I$) and $R_1$, $R_2$ are $n \times n$ upper triangular matrices. Show that there is a diagonal matrix $D$ with diagonal entries $\pm 1$ where $Q_2 = Q_1D$ and $DR_2 = R_1$.

(6) Repeat Exercise 5 for a *complex* $m \times n$ matrix $A$ with some modifications: $A = Q_1 R_1 = Q_2 R_2$ where $Q_1$, $Q_2$ are $m \times n$ and $\overline{Q_1}^T Q_1 = \overline{Q_2}^T Q_2 = I$, while $R_1$, $R_2$ are upper triangular. Show that there is a diagonal matrix $D$ with diagonal entries of the form $e^{i\theta}$, $\theta \in \mathbb{R}$ where $Q_2 = Q_1 D$ and $D R_2 = R_1$.

(7) Show that if $A$ is a real $m \times n$ matrix where $A^T A = L L^T$ (Cholesky factorization) and $A = QR$ (QR factorization) then there is a diagonal matrix $D$ with diagonal entries $\pm 1$ where $R = DL^T$.

(8) QR factorizations can give orthonormal bases for the range and null space of rectangular matrices. Suppose $A$ is a real $m \times n$ matrix with linearly independent columns. Let $A = [Q_1, Q_2] \begin{bmatrix} R_1 \\ 0 \end{bmatrix}$ is the QR factorization of $A$. Show that $\text{range}(A) = \text{range}(Q_1)$ and $\text{null}(A^T) = \text{range}(Q_2)$. [**Hint:** For the second part, you can use Theorem 8.6.]

## 2.3 Sparse Matrices

Sparse matrices are matrices where most entries are zero. This can be exploited using sparse matrix data structures that are substantially different from the simple arrays used for dense or full matrices where every entry is explicitly represented. Sparse matrices often arise in practice related to differential equations and networks.

In this section, we will focus on direct methods rather than iterative methods. Sparse matrices can also be used in iterative methods where the matrix is only used to compute matrix–vector products. If a sparse matrix is only used for computing matrix–vector products, the exact location of the non-zeros is not particularly important. What is important is having an implementation that can efficiently navigate the data structure.

Direct methods, such as LU, Cholesky, and QR factorizations, often result in the creation of new non-zero entries that had originally been zero. This is called *fill-in* and is important for understanding the performance of a sparse factorization method.

### *2.3.1 Tridiagonal Matrices*

The simplest example of a family of sparse matrices that have practical impact are *tridiagonal matrices*. This is a matrix $A$ where $a_{ij} \neq 0$ implies $|i - j| \leq 1$. That is, all non-zeros of a tridiagonal matrix occur on the main diagonal ($i = j$) or the first sub-diagonal ($i = j + 1$) or first superdiagonal ($i = j - 1$). This is part of a more general family of *banded matrices* of bandwidth $b$ where $a_{ij} \neq 0$ implies $|i - j| \leq b$.

**Algorithm 18** Tridiagonal LU factorization with no pivoting

```
1  function tridiagLU(a, b, c)
2    u_1 ← a_1 ; v_1 ← b_1 ; ℓ_1 ← c_1/u_1
3    for k = 2, ..., n − 1
4      u_k ← a_k − ℓ_{k−1} v_{k−1}
5      v_k ← b_k ; ℓ_k ← c_k/u_k
6    end for
7    u_n ← a_n − ℓ_{n−1} v_{n−1}
8    return ℓ, u, v
9  end function
```

A general example of a tridiagonal matrix is shown below:

$$\begin{bmatrix} a_1 & b_1 & & & \\ c_1 & a_2 & b_2 & & \\ & c_2 & a_3 & b_3 & \\ & & c_3 & a_4 & \ddots \\ & & & \ddots & \ddots & b_{n-1} \\ & & & & c_{n-1} & a_n \end{bmatrix}. \tag{2.3.1}$$

Computing the LU factorization without partial pivoting can be done efficiently:

$$\begin{bmatrix} a_1 & b_1 & & & \\ c_1 & a_2 & b_2 & & \\ & c_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & c_{n-1} & a_n \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ \ell_1 & 1 & & & \\ & \ell_2 & 1 & & \\ & & \ddots & \ddots & \\ & & & \ell_{n-1} & 1 \end{bmatrix} \begin{bmatrix} u_1 & v_1 & & & \\ & u_2 & v_2 & & \\ & & u_3 & \ddots & \\ & & & \ddots & v_{n-1} \\ & & & & u_n \end{bmatrix}$$

leads to the equations

$$\begin{aligned} a_1 &= u_1, & b_1 &= v_1, & c_1 &= \ell_1 u_1, \\ a_2 &= \ell_1 v_1 + u_2, & b_2 &= v_2, & c_2 &= \ell_2 u_2, \\ a_3 &= \ell_2 v_2 + u_3, & b_3 &= v_3, & c_3 &= \ell_3 u_3, \\ &\vdots & &\vdots & &\vdots \\ a_{n-1} &= \ell_{n-2} v_{n-2} + u_{n-1}, & b_{n-1} &= v_{n-1}, & c_{n-1} &= \ell_{n-1} u_{m-1}, \\ a_n &= \ell_{n-1} v_{n-1} + u_n. \end{aligned}$$

Solving these equations gives Algorithm 18. This algorithm can also be written to overwrite $\boldsymbol{a}$ with $\boldsymbol{u}$, $\boldsymbol{b}$ with $\boldsymbol{v}$, and $\boldsymbol{c}$ with $\boldsymbol{\ell}$.

**Algorithm 19** Tridiagonal solver using LU factorization

```
1  function tridiagsolve(ℓ, u, v, y)
2    x ← y
3    x_n ← x_n/u_n
4    for k = n − 1, ..., 2, 1
5      x_k ← (y_k − v_k x_{k+1})/u_k
6    end for
7    for k = 2, ..., n
8      x_k ← x_k − ℓ_k x_{k−1}
9    end for
10   return x
11 end function
```

Algorithm 18 requires $3n-1$ flops and $3n-2$ extra memory locations, making it an extremely efficient algorithm. Solving a tridiagonal linear system after the factorization can also be done very efficiently, as shown in Algorithm 19.

If we incorporate partial pivoting, then the algorithms will be somewhat more complex. However, the sparsity of the matrix limits the possibilities for pivoting. For example, the first row could only be swapped with the second row; in general, row $k$ could only be swapped with row $k+1$. These swaps do result in additional non-zero entries. We can show how this happens in an example below; "$*$" denotes an original non-zero, while "$+$" denotes a new non-zero entry created by the algorithm.

$$\begin{bmatrix} * & * & & & \\ * & * & * & & \\ & * & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix} \xrightarrow{\text{swap rows 1 \& 2}} \begin{bmatrix} * & * & + & & \\ * & * & 0 & & \\ & * & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix} \xrightarrow{\text{elimination step}} \begin{bmatrix} * & * & + & & \\ 0 & * & * & & \\ & * & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix}$$

$$\begin{bmatrix} * & * & + & & \\ 0 & * & * & & \\ & * & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix} \xrightarrow{\text{swap rows 2 \& 3}} \begin{bmatrix} * & * & + & & \\ & * & * & + & \\ & * & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix} \xrightarrow{\text{elimination step}} \begin{bmatrix} * & * & + & & \\ 0 & * & * & + & \\ & 0 & * & \ddots & \\ & & \ddots & \ddots & * \\ & & & * & * \end{bmatrix}$$

etc.

The result is an additional second superdiagonal of potential non-zero entries in the $U$ matrix. The $L$ matrix in the LU factorization remains a bidiagonal matrix with entries on the main diagonal and the first sub-diagonal.

QR factorizations of tridiagonal matrices can also be computed, and the resulting sparsity pattern is essentially the same as for LU with partial pivoting: an additional second superdiagonal of non-zero entries is created in the $R$ factor. The $Q$ matrix given explicitly is *not* sparse, but if it is stored in compact form (such as using the WY

form, or just storing the Householder vectors) it can be stored as a sparse matrix with no additional storage of fill-in except for a single vector of length $n$ if the factored matrix is $m \times n$.

Banded matrices can also be factored in a way that preserves their sparsity pattern using either LU factorization without pivoting or Cholesky factorization. LU factorization with partial pivoting and QR factorization both result in fill-in: if the bandwidth of the matrix factored is $b$ then the bandwidth of $U$ in the LU factorization and the $R$ in the QR factorization is increased by $b - 1$. See [105, Sec. 4.3, pp. 152–160] for more details about factoring banded matrices.

### 2.3.2 Data Structures for Sparse Matrices

General sparse matrices require more flexible data structures than we need for dense or full matrices. The notes of G.W. Stewart on sparse matrices [239] provide a good introduction to many of the issues in creating sparse factorization codes. The most common data structures for sparse matrices are *Compressed Sparse Row* (CSR) and *Compressed Sparse Column* (CSC) data structures. These data structures are essentially mirror images of each other; the CSR representation of a matrix $A$ can be considered as the CSC representation of $A^T$, and vice versa. CSC format is used by MATLAB and Julia, while Python/NumPy uses both.

**Example 2.14** To illustrate how these data structures work, consider the following $5 \times 5$ sparse matrix:

$$\begin{bmatrix} 7 & 2 & & & \\ 3 & 1 & & & 6 \\ & & 4 & 1 & \\ 2 & & & 3 & \\ & 1 & 9 & 5 & 8 \end{bmatrix}.$$

The CSR and CSC representations are shown in Figure 2.3.1.

The `rowptr` and `colptr` arrays hold the indexes of the start of the corresponding row and column, respectively. The `colidx` and `rowidx` entries give the column (respectively, row) index for that entry, while the `value` entries give the actual matrix entry.

Algorithm 20 gives pseudo-code for computing $\boldsymbol{y} = A\boldsymbol{x}$ where $A$ is given by either a CSR or CSC data structure. For simplicity, it is assumed that the final entry in both the `rowptr` and `colptr` arrays is the number of non-zeros in the matrix plus one.

The CSR and CSC matrix–vector multiplication algorithms are roughly equally efficient: the CSR algorithm can hold $y_i$ in a register while accesses to the $\boldsymbol{x}$ vector may be widely scattered; the CSC algorithm can hold $x_j$ in a register while accesses to the $\boldsymbol{y}$ vector may be widely scattered.

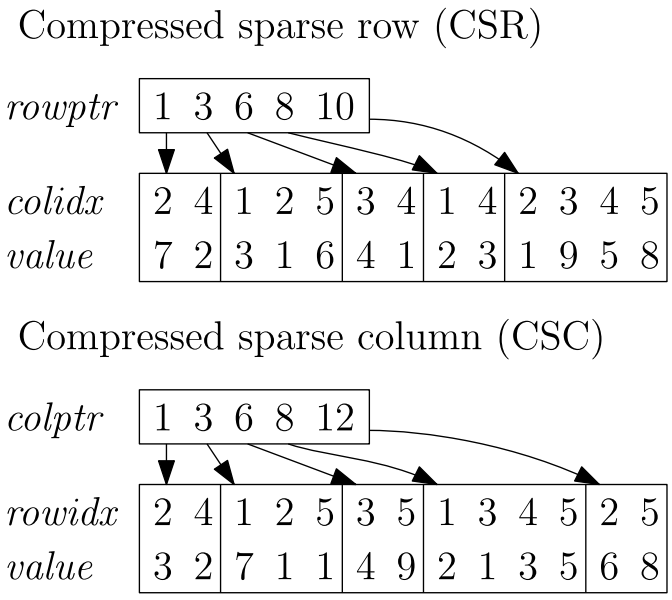



**Fig. 2.3.1** CSR and CSC sparse matrix data structures

**Algorithm 20** Computing $\boldsymbol{y} \leftarrow \boldsymbol{y} + A\boldsymbol{x}$ using CSR and CSC data structures

```
1  function spmultCSR(A, x, y)
2    for i = 1, 2, ..., m
3      for k = A.rowptr_i, ..., A.rowptr_{i+1} − 1
4        j ← A.colidx_k
5        y_i ← y_i + A.value_k · x_j
6      end
7    end
8    return y
9  end
// --------------------
1  function spmultCSC(A, x, y)
2    for j = 1, 2, ..., m
3      for k = A.colptr_j, ..., A.colptr_{j+1} − 1
4        i ← A.rowidx_k
5        y_i ← y_i + (A.value_k) · x_j
6      end
7    end
8    return y
9  end
```

Other ways of representing a sparse matrix include using a set of $(i, j, a_{ij})$ triples, or using a dictionary or hash table to look-up values $a_{ij}$ from a pair $(i, j)$. From the point of view of memory access, CSR and CSC data structures are the most efficient for most matrix operations. Memory access in the matrix algorithm should be

adapted to being along columns for CSC sparse matrices, and along rows for CSR sparse matrices. Dictionary or hash table representations usually are not so efficient regarding memory access as they require pseudo-random memory accesses. However, both the set-of-triples $(i, j, a_{ij})$ and dictionary representations can be much more efficient about inserting new non-zero entries than either CSR or CSC structures. An efficient way to create a sparse matrix in CSC or CSR format is to create a list or array of $(i, j, a_{ij})$ triples, sort the triples lexicographically according to either the $(i, j)$ or $(j, i)$ pairs according to whether CSR or CSC format is desired. Then build the sparse matrix sequentially from the sorted list of triples. Sorting the triples ensures that no new entries has to be inserted between already existing entries, only appended to the end of the *value* and *rowidx* or *colidx* arrays.

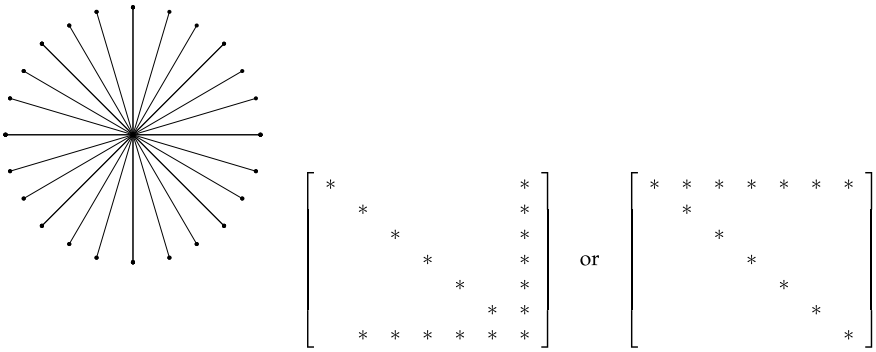
### 2.3.3 Graph Models of Sparse Factorization

In this section, we will focus more on the case of symmetric matrices and Cholesky factorization. We can represent the sparsity structure of a matrix by means of a *graph* or *network*. The nodes or vertices of the graph of an $n \times n$ symmetric matrix $A$ are simply the integers $1, 2, \ldots, n$. There is an edge from $i$ to $j$ (denoted $i \mapsto j$) in the graph of $A$ (denoted $G_A$) if and only if $a_{ij} \neq 0$. For a symmetric matrix, since we have edges in both directions ($i \mapsto j$ and $j \mapsto i$) if $a_{ij} = a_{ji} \neq 0$, so we consider the edges as being undirected (denoted $i \sim j$), although we ignore edges connecting a node to itself. The graph consists of both its vertices and edges, denoted $G = (V, E)$ where $V$ is the set of vertices and $E$ is the set of edges. For any vertex $x \in V$ of $G$, its set of neighbors is $\mathcal{N}(x; G) = \{ y \in V \mid x \sim_G y \}$ where $x \sim_G y$ means that there is an edge connecting $x$ and $y$ in graph $G$.

The graph of a tridiagonal matrix is simply a path: $1 \sim 2 \sim 3 \sim \cdots \sim n-1 \sim n$. The graph of an $n \times n$ diagonal matrix consists of $n$ vertices and no edges. An $n \times n$ full or dense matrix, with all entries non-zero, has a graph called the complete graph where for any nodes $i \neq j$ there is an edge $i \sim j$. This graph is denoted $K_n$. We can think of $K_3$ as a triangle, and $K_4$ as a square plus both diagonals.

The ordering of the rows and columns can be of immense importance regarding the efficiency of using Cholesky factorization for solving systems of equations. Consider the sparsity structure in Figure 2.3.2. The graph on the left is a *spokes graph*, which (depending on the ordering) can represent either of the two arrowhead matrices on the right. The first of these matrices where the "tip" of the arrow is at the bottom-right corner suffers from no fill-in if we use Cholesky factorization, or LU factorization with no pivoting. The second of these matrices, where the "tip" of the arrow is at the top-left corner, results in complete fill-in after just the first stage of the factorization.

We want to use the graph model to understand how to order the rows and columns to minimize fill-in. To do this, consider carrying out the first stage of Cholesky factorization, but with a specified node in the graph listed as being the "first". Recall the recursive formulation of the Cholesky factorization:




(A) Spokes graph (B) Arrowhead matrices

**Fig. 2.3.2** Spokes graph and arrowhead matrix

$$A = \left[\begin{array}{c|c} \alpha & \boldsymbol{a}^T \\ \hline \boldsymbol{a} & \widetilde{A} \end{array}\right] = \left[\begin{array}{c|c} \sqrt{\alpha} & \\ \hline \boldsymbol{\ell} & I \end{array}\right] \left[\begin{array}{c|c} \alpha & \\ \hline & \widetilde{A} - \boldsymbol{a}\boldsymbol{a}^T/\alpha \end{array}\right] \left[\begin{array}{c|c} \sqrt{\alpha} & \boldsymbol{\ell}^T \\ \hline & I \end{array}\right].$$

The graph of $\widetilde{A} - \boldsymbol{a}\boldsymbol{a}^T/\alpha$ with the first node and all the edges from the first node removed, but with a number of nodes added coming from $\boldsymbol{a}\boldsymbol{a}^T/\alpha$. Now $(\boldsymbol{a}\boldsymbol{a}^T/\alpha)_{ij} = a_{i1}a_{j1}/a_{11} \neq 0$ if $a_{i1} \neq 0$ and $a_{j1} \neq 0$. Ignoring the possibility of "accidentally" zeroing out an entry of the matrix (where $a_{ij} = a_{i1}a_{j1}/a_{11}$ coincidentally), we get new edges $i \sim j$ in the graph of $\widetilde{A} - \boldsymbol{a}\boldsymbol{a}^T/\alpha$ where $1 \sim i$ and $1 \sim j$. That is, the graph of $\widetilde{A} - \boldsymbol{a}\boldsymbol{a}^T/\alpha$ is the graph of $\widetilde{A}$ with edges $i \sim j$ added whenever $i \sim 1 \sim j$ in the graph of $A$.

The task now is: given the graph $G_A$ of an $n \times n$ matrix $A$, we want to choose which vertex $x$ of $G_A$ to eliminate. When we eliminate vertex $x$, we remove $x$ and all its edges from $G_A$ and add edges between each pair of neighbors of $x$ in $G_A$. We can describe this operation more formally as

$$G' = (V \setminus \{x\},\ (E \setminus \mathcal{N}(x)) \cup \{\, i \sim j \mid i, j \in \mathcal{N}(x),\ i \neq j \,\}).$$

This gives us the graph of $A' := \widetilde{A} - \boldsymbol{a}\boldsymbol{a}^T/\alpha$, which is used recursively to compute the remainder of the Cholesky factorization. So we want to find an ordering of the vertices of $G_A$ so that these elimination operations add as few edges as possible.

The path graph $1 \sim 2 \sim 3 \sim \cdots \sim n$ can be eliminated in the natural ordering without producing fill-in. At stage $k$, the graph is $k \sim k+1 \sim k+2 \sim \cdots \sim n$. Eliminating vertex $k$ does not result in any fill-in, because the only neighbor of $k$ is $k+1$ and there is already an edge $k \sim k+1$.

There are many graphs with the property that there is a fill-in free elimination ordering. These include banded matrices, for example. The graph of a general banded $n \times n$ matrix with bandwidth $b$ has vertices $1, 2, \ldots, n$ and edges $i \sim j$ if and only if $|i - j| \leq b$. At the first stage of the factorization, vertex 1 is connected to vertices 2, 3, …, $b+1$. Elimination of vertex 1 connects vertices $2, 3, \ldots, b+1$ to each other.

**Algorithm 21** Cuthill–McKee algorithm

```
1   function cuthillmckee(G, x)
2     q ← queue() // create empty queue
3     ℓ ← list() // create empty list
4     add(ℓ, x); add(q, x) // add x to both ℓ and q
5     while q not empty
6       v ← remove(q) // v was first entry in q
7       N ← {y | y ∼ v is an edge of G} sorted by deg(y)
8       for each y ∈ N in order
9         add(ℓ, y); add(q, y)
10      end for
11    end while
12    return ℓ
13  end function
```

But they were already connected to each other! So there are no new edges. Repeating the process, by the time we get to stage $k$ of the factorization, vertex $k$ is connected to vertices $k+1, k+2, \ldots, k+b$. But again, they were already connected to each other, so there are again no new edges. Thus, we can complete the Cholesky factorization, or LU factorization without pivoting, of a banded matrix without incurring any additional fill-in.

We can use these insights to see how to factor an arrowhead matrix by looking at the spokes graph shown in Figure 2.3.2. Since every vertex of the spokes graph, except the "hub" vertex, is equivalent from the point of view of graph theory, the choice of which vertex to eliminate first comes down to whether to eliminate a "spoke" vertex or the "hub" vertex. Eliminating the hub vertex immediately results in edges between all pairs of the other vertices, which is complete fill-in. However, eliminating a spoke vertex does not result in any fill-in since the only neighbor of a spoke vertex is the hub vertex. Thus, a no fill-in ordering will put the hub vertex last. The ordering of the spoke vertices is arbitrary.

#### 2.3.3.1 Heuristics: Reverse Cuthill–McKee

The *Cuthill–McKee ordering* [66, 101] is a method for reducing the *bandwidth* of a sparse matrix: the bandwidth of $A$ is $1 + \max\left\{\, |i-j| \mid a_{ij} \neq 0 \,\right\}$. Tridiagonal matrices, for example, have bandwidth three. An $n \times n$ matrix of bandwidth $b \geq 1$ has an LU factorization that also has bandwidth $b$ provided there is no pivoting. This is useful in limiting the amount of fill-in as each row of a matrix with bandwidth $b$ has no more than $b$ entries in each row or column. The Cuthill–McKee ordering is based on *breadth-first search* [59] for graphs, a method of traversing a graph. Breadth-first search uses a queue, being a data structure where items can be added to the *back* of the queue, which can later be removed from the *front* of the queue. The order of items in the queue is fixed according to the order in which they are added to the queue. The Cuthill–McKee algorithm is shown in Algorithm 21.

While the Cuthill–McKee ordering works well for minimizing the bandwidth, it turns out that reversing the Cuthill–McKee ordering actually is better for reducing fill-in and floating point operation counts. The reason for the benefits of using the reverse Cuthill–McKee ordering over the original Cuthill–McKee ordering is explained in [165].

As an example for comparing the forward and reverse Cuthill–McKee orderings, consider the spokes graphs as illustrated in Figure 2.3.2, where there is a single hub vertex and all other vertices are connected to the hub vertex, but no other vertex or edge. It should be noted that no ordering of a spokes graph gives small bandwidth. The Cuthill–McKee ordering starting with a non-hub vertex always has the hub vertex as the second vertex. All other non-hub vertices follow. The resulting Cholesky factorization using the Cuthill–McKee then completely fills in except for the first row and column. On the other hand, in the reverse Cuthill–McKee ordering, the hub vertex is the *second last* vertex. Applying the Cholesky factorization to the re-ordered matrix using the reverse Cuthill–McKee ordering for the spokes graph results in *no fill-in*.

In fact, Liu and Sherman [165] showed that the reverse Cuthill–McKee ordering is *never* worse than the forward Cuthill–McKee ordering for either the number of floating point operations or the amount of fill-in.

#### 2.3.3.2 Heuristics: Nested Dissection

The main idea behind nested dissection is to split the graph of the matrix into two independent parts with a "separator" set being the "glue" between the parts. The idea is illustrated in Figure 2.3.3. This approach is developed in [101], for example.

This means that the vertices of the original graph $G$ is split $V = V_1 \cup V_2 \cup S$ where $V_1$, $V_2$, and $S$ are pairwise disjoint. Furthermore, no edge $x \sim y$ in $G$ is between vertices $x$ and $y$ where $x \in V_1$ and $y \in V_2$ or vice versa.

The subgraph $G_1$ consists of the vertices $V_1$ and all edges of $G$ between vertices in $V_1$; subgraph $G_2$ consists of the vertices $V_2$ and all edges of $G$ between vertices in $V_2$.

If we have a symmetric sparse matrix $A$ whose graph $G_A$ can be split in this way, if we order the rows and columns of $A$, ordering the rows (and columns) corresponding to $V_1$ first, followed by the rows (and columns) corresponding to $V_2$, followed finally by the rows (and columns) corresponding to $S$, then the matrix will have the structure

$$P^T A P = \left[\begin{array}{cc|c} A_1 & & B_1^T \\ & A_2 & B_2^T \\ \hline B_1 & B_2 & A_S \end{array}\right].$$

We can think of applying Cholesky factorization in a block fashion to this matrix: $P^T A P = L^T L$ where

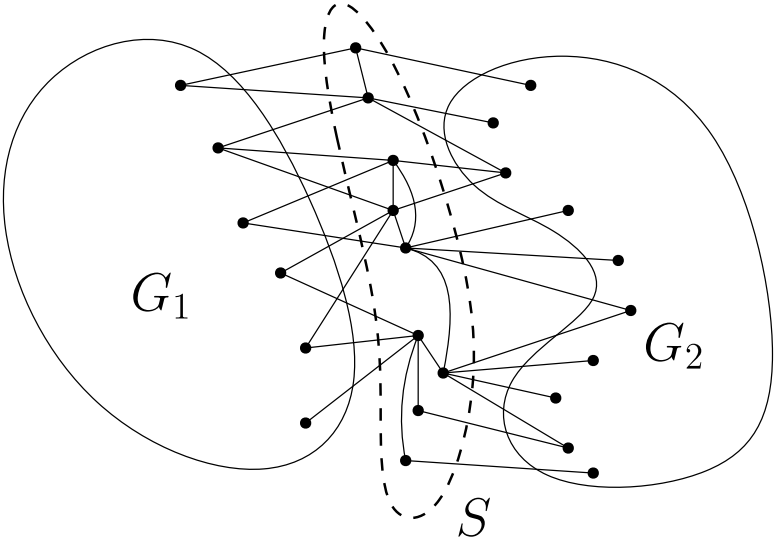



**Fig. 2.3.3** Splitting graph into two parts with a separator set

$$L = \left[\begin{array}{cc|c} L_1 & & \\ & L_2 & \\ \hline M_1 & M_2 & L_S \end{array}\right] \qquad \text{where}$$

$$\begin{aligned} L_1 L_1^T &= A_1, \qquad \text{(Cholesky factorization)} \\ L_2 L_2^T &= A_2, \qquad \text{(Cholesky factorization)} \\ M_1 &= B_1 L_1^{-T}, \qquad \text{(forward substitution)} \\ M_2 &= B_2 L_2^{-T}, \qquad \text{(forward substitution)} \\ L_S L_S^T &= A_S - M_1 M_1^T - M_2 M_2^T \qquad \text{(Cholesky factorization).} \end{aligned}$$

The first two Cholesky factorizations for $A_1$ and $A_2$ can be done independently. Furthermore, we can apply the splitting approach recursively to $A_1$ and $A_2$ by further splitting graphs $G_1$ and $G_2$, which would have their own separator sets. This can be repeated recursively until the number of vertices is too small for this to be helpful. The Cholesky factorization of $A_S - M_1 M_1^T - M_2 M_2^T$ is typically dense. The reason for this is that the graph $G_S$ of $A_S - M_1 M_1^T - M_2 M_2^T$ has an edge $x \sim y$ for $x, y \in S$ if and only if there is a *path* from $x$ to $y$ through either $G_1$ or $G_2$, but not both. If either $G_1$ or $G_2$ is a connected graph (which is usually the case), then $G_S$ will be a complete graph and $L_S$ will be dense.

We can analyze this approach to estimate the number of floating point operations needed to compute the sparse Cholesky factorization, and to estimate the fill-in needed for the Cholesky factors. To get these bounds, we assume that $B_1$ and $B_2$ are dense. We can use the recursive decomposition as follows:

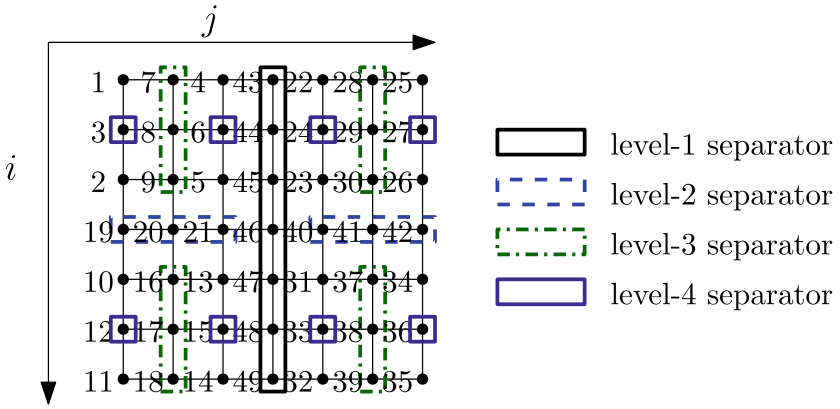



**Fig. 2.3.4** Nested dissection ordering for a $7 \times 7$ rectangular grid

$$\begin{aligned}\text{flops}(G) &= \text{flops}(G_1) + \text{flops}(G_2) \\ &\quad + (\text{fill-in}(G_1) + \text{fill-in}(G_2))\,|S| \\ &\quad + (|V_1| + |V_2|)\,|S|^2 + \frac{1}{3}\,|S|^3\,, \\ \text{fill-in}(G) &= \text{fill-in}(G_1) + \text{fill-in}(G_2) \\ &\quad + (|V_1| + |V_2| + |S|)\,|S|\,.\end{aligned}$$

A good splitting has $|V_1| \approx |V_2| \gg |S|$.

To turn these values into concrete estimates, we need to estimate the size of $|S|$. For planar graphs $G$, we can bound $|S| = \mathcal{O}(\sqrt{n})$ [164].

This is particularly easy to implement on, for example, a two-dimensional rectangular grid. For an $M \times N$ rectangular grid, we take $S$ to be either a vertical line or a horizontal line splitting the grid into two roughly equal parts; whether $S$ is vertical or horizontal depends on whether $M > N$ or $M < N$. If $M = N$ then $S$ can be either vertical or horizontal. An example of a $7 \times 7$ rectangular grid is shown in Figure 2.3.4. The vertical solid box shows the separator set $S$ at the top level, dashed for the separator sets at the next level, then dot-dash, and finally solid again at the lowest level.

Nested dissection often produces near-optimal orderings [1]. For an $N \times N$ rectangular grid, the factorization can be performed in $\mathcal{O}(N^3)$ floating point operations and $\mathcal{O}(N^2 \log N)$ fill-in. By contrast, the standard ordering by row or by column results in $\mathcal{O}(N^4)$ floating point operations and $\mathcal{O}(N^3)$ fill-in. For three dimensions, nested dissection still gives good result for direct methods, but the cost increases more rapidly in $N$ and iterative methods become increasingly advantageous.

### 2.3.4 Unsymmetric Factorizations

Graph models can be developed for unsymmetric factorizations, such as LU factorizations, both with and without partial pivoting and QR factorization. Rather than using undirected graphs with the set of vertices $\{1, 2, 3, \ldots, n\}$, we can use *directed graphs* where there is an edge $i \mapsto j$ if $a_{ij} \neq 0$, or *bipartite graphs* with vertices $\{r_1, r_2, \ldots, r_m\} \cup \{c_1, c_2, \ldots, c_n\}$ with edges $r_i \sim c_j$ if $a_{ij} \neq 0$. A graph is *bipartite* if the vertex set $V = V_1 \cup V_2$ with $V_1 \cap V_2 = \emptyset$ and every edge is undirected but joins a vertex in $V_1$ with a vertex in $V_2$. In a bipartite graph representing an unsymmetric matrix, $V_1$ can be taken to represent the rows, while $V_2$ can be taken to represent the columns.

Here we focus on the QR factorization and the bipartite graph model:

$$V = \{r_1, r_2, \ldots, r_m\} \cup \{c_1, c_2, \ldots, c_n\}$$
$$\text{with edges } r_i \sim_A c_j \text{ if } a_{ij} \neq 0.$$

Let $\mathcal{N}_A(x) = \{\, y \mid x \sim_A y \,\}$ be the set of neighbors of $x$ in $G_A$. If $A = QR$, then let $B = A^T A = R^T Q^T Q R = R^T R$ with $R$ upper triangular. But $B = A^T A$ is symmetric and we can use the undirected graph model of $B$: the vertices of $G_B$ are the column vertices of $A$ ($V_B = \{c_1, c_2, \ldots, c_n\}$), and $c_i \sim_B c_j$ if $b_{ij} \neq 0$. Unless there are "accidental" zeros, $b_{ij} \neq 0$ if $\mathcal{N}(c_i) \cap \mathcal{N}(c_j) \neq \emptyset$. We can then use the elimination model for Cholesky factorization of $B$ to determine how to order the columns of $B$, and thus to order the columns of $A$. The rows of $A$ should then be ordered so that $r_k \sim_A c_k$ to avoid unnecessary fill-in at the diagonal entry $(k, k)$ in the QR factorization of $A$.

The sparsity structure of the LU factorization of $A$ *without pivoting* is a subset of the sparsity structure of the QR factorization of $A$. If we allow for all possible row swaps in partial pivoting, then the sparsity structure of the union of possible sparsity structures of LU factorizations of $A$ is the sparsity structure of the QR factorization. This means that the sparsity structure of the QR factorization can be pre-computed, and this sparsity structure can be used for LU factorization with partial pivoting, without the need for any additional fill-in.

**Exercises.**

(1) Suppose we add a single symmetric entry to the top-right and bottom-left corners of a symmetric tridiagonal matrix. What is the graph of the resulting matrix? How much fill-in will be generated by Cholesky factorization? Assume that the resulting matrix is positive definite so that the Cholesky factorization exists.

(2) The graph of an arrowhead matrix is a spokes graph (Figure 2.3.2). If we have a symmetric matrix that is the sum of a tridiagonal matrix and a matrix $\boldsymbol{u}\boldsymbol{e}_1^T + \boldsymbol{e}_1\boldsymbol{u}^T$ that fills in the first row and column, show that the corresponding graph is a *wheel graph*: a cycle of $n - 1$ vertices together with a "hub vertex"

that is connected to every node in the cycle. Show that there is an ordering that gives no fill-in for this graph.

(3) Show that if the graph of a symmetric matrix is a tree (a connected undirected graph with no cycles) then the matrix can be re-ordered so that Cholesky factorization gives no fill-in.

(4) Show that for a symmetric banded matrix ($a_{ij} \neq 0$ only if $|i - j| \leq b$ where $b$ is the "bandwidth") the Cholesky factorization can be done with no fill-in outside the band. Note that a tridiagonal matrix is the special case of a banded matrix with bandwidth $b = 1$.

(5) The standard discretization of the Laplacian operator in two dimensions is $(-\nabla^2 u)_{ij} = (4u_{i,j} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1})/(\Delta x)^2$. Create an $N^2 \times N^2$ matrix using the natural ordering which assign point $(i, j)$ on the grid to index $N(i-1) + j$ for $1 \leq i, j \leq N$. The corresponding graph is the grid graph that connects $(i, j)$ to $(i \pm 1, j)$ and $(i, j \pm 1)$. Generate these matrices for $N = 5$, 10, 20, 40, 100, 200, 400, 1000. How much fill-in is generated by Cholesky factorization? Apply the reverse Cuthill–McKee reordering algorithm to these matrices. How much fill-in is now generated?

(6) For the matrices in Exercise 5, use the nested dissection algorithm of Section 2.3.3.2 to minimize fill-in. How much fill-in is generated by Cholesky factorization? Plot the amount of fill-in against $N$.

(7) Graphs can generate matrices. Specifically, given an undirected graph $G$ with vertex set $V$ and edge set $E$, we define the *graph Laplacian* $L_G$ to be a $V \times V$ matrix where $(L_G)_{ii}$ is the degree of node $i$ in $G$, with $(L_G)_{ij} = -1$ if $i \sim j$ is an edge in $G$ and $(L_G)_{ij} = 0$ otherwise. Show that:

(a) the graph of $L_G$ is $G$;

(b) $L_G$ is a symmetric positive semi-definite matrix [**Hint:** $z^T L_G z = \frac{1}{2}\sum_{i,j:i\sim j}(z_i - z_j)^2$];

(c) the null space of $L_G$ is the set of vectors that are constant on each connected component of $G$;

(d) $(\alpha I + L_G)^{-1}$ is a matrix with only non-negative entries for any $\alpha > 0$. [**Hint:** $\alpha I + L_G$ is strictly diagonally dominant with positive diagonal entries. Write $\alpha I + L_G = D - F$ where $D$ is the diagonal part of $\alpha I + L_G$, and $F$ has non-negative entries. Use the power series formula for $(I - D^{-1}F)^{-1}$.]

(8) Suppose that $A$ is a symmetric strictly diagonally dominant tridiagonal matrix ($|a_{ii}| > |a_{i,i-1}| + |a_{i,i+1}|$ for all $i$). Show that the magnitude of the entries of $A^{-1}D$ ($D$ is the diagonal part of $A$) decrease exponentially rapidly away from the main diagonal. [**Hint:** Let $\alpha = \max_i \left(|a_{i,i-1}| + |a_{i,i+1}|\right) / |a_{ii}| < 1$. Writing $A = D - F$, $A^{-1}D = (I - D^{-1}F)^{-1} = I + D^{-1}F + (D^{-1}F)^2 + \cdots$ with $\left\|D^{-1}F\right\|_\infty = \alpha < 1$.]

(9) A standard procedure for eigenvalue/vector algorithms for real symmetric matrix $A$ is to put a matrix into a certain sparse form by means of Givens' rotations: $A \leftarrow G_{ij}(c, s)^T A\, G_{ij}(c, s)$ where $G_{ij}(c, s)$ is the identity matrix except that for rows $i$, $j$ and columns $i$, $j$. Here $G_{ij}(c, s)_{ii} = G_{ij}(c, s)_{jj} = c$ while

$G_{ij}(c,s)_{ji} = -G_{ij}(c,s)_{ij} = s$ with $c^2 + s^2 = 1$. Given $(i, j)$ where $a_{ij} \neq 0$ we can find $(c, s)$ so that the $(i, j)$ entry of $G_{ij}(c, s)^T A\, G_{ij}(c, s)$ is set to zero. Show that unless there is "accidental" cancellation, this operation can be represented by the following operation on the graph of $A$: for every $k$ where $i \sim k$, add an edge $j \sim k$; for every $k$ where $j \sim k$, add an edge $i \sim k$; delete the edge $i \sim j$. This graph operation is introduced in [238].

(10) Read about *SuperLU* [75], a high-performance super-nodal method for carrying out LU factorization with partial pivoting. Outline how the algorithm improves performance over naive implementations of sparse LU factorization with partial pivoting.

## 2.4 Iterations

The methods we have seen so far are direct methods: they give answers which would be exactly correct if done in exact arithmetic. But for large systems of equations, if the accuracy requirements are not too stringent, using iterative methods can give good results far more cheaply in terms of time and memory. Some iterative methods were developed in the nineteenth and early twentieth centuries. The 1950s saw the creation of the conjugate gradient method which differed from most of the previous method, by not requiring detailed knowledge of, or manipulation of, the entries of the matrix. Instead, the conjugate gradient method just required the computation of matrix–vector products, short linear combinations, and inner products. In the 1970s through 1990s a number of new methods of this type were developed. These methods are called Krylov subspace methods.

The two approaches of classical and Krylov subspace methods are not exclusive. In fact, classical iterations are very useful as *preconditioners* for Krylov subspace methods.

### *2.4.1 Classical Iterations*

The classical iterations and their variants are based on *matrix splittings*: to solve $A\boldsymbol{x} = \boldsymbol{b}$ for $\boldsymbol{x}$ we split $A = M - N$ where $M$ is invertible. The equation $A\boldsymbol{x} = \boldsymbol{b}$ can then be put into the form $M\boldsymbol{x} = N\boldsymbol{x} + \boldsymbol{b}$. Then we can create the *fixed-point iteration*

$$\boldsymbol{x}_{k+1} \leftarrow M^{-1}(N\boldsymbol{x}_k + \boldsymbol{b}), \qquad k = 0, 1, 2, \ldots. \tag{2.4.1}$$

Classical iterations include the Jacobi or Gauss–Jacobi iteration: choose $M = D$, the diagonal part of $A$, and $N = D - A$. One iteration can be implemented as Algorithm 22. The iteration matrix is $G_J = I - D^{-1}A$. The iteration is

**Algorithm 22** Jacobi iteration for $A\boldsymbol{x} = \boldsymbol{b}$

```
1   function jacobi(A, x, b)
2     for i = 1, 2, ..., n
3       s ← b_i
4       for j = 1, 2, ..., n
5         if j ≠ i
6           s ← s − a_ij x_j
7         end if
8       end for
9       y_i ← s/a_ii
10    end for
11    x ← y
12    return x
13  end function
```

**Algorithm 23** Gauss–Seidel iteration for $A\boldsymbol{x} = \boldsymbol{b}$

```
1   function gauss_seidel(A, x, b)
2     for i = 1, 2, ..., n
3       s ← b_i
4       for j = 1, 2, ..., n
5         if j ≠ i
6           s ← s − a_ij x_j
7         end if
8       end for
9       x_i ← s/a_ii
10    end for
11    return x
12  end function
```

$$\boldsymbol{x}_{k+1} \leftarrow D^{-1}((D - A)\boldsymbol{x}_k + \boldsymbol{b}) \tag{2.4.2}$$

The Jacobi method is more effective for sparse matrices as the number of floating point operations for one iteration of the Jacobi method uses $\leq 2\,nnz(A)$ floating point operations where $nnz(A)$ is the number of non-zero entries of $A$.

Another classical iteration is the Gauss–Seidel iteration. Now we choose $M = D + L$ where $D$ is the diagonal part of $A$, $L$ the strictly lower triangular part of $A$, so that $N$ is the strictly upper triangular part $U$ of $A$. This is often preferred over the Jacobi iteration as it is not necessary to keep a separate vector to store the result, as shown in Algorithm 23. The iteration can be represented in matrix–vector form as

$$\boldsymbol{x}_{k+1} \leftarrow (D + L)^{-1}\left[\boldsymbol{b} - U\boldsymbol{x}_k\right]. \tag{2.4.3}$$

The iteration matrix is $G_{GS} = -(D + L)^{-1}U$. The number of floating point operations for Gauss–Seidel per iteration is the same as for the Jacobi method.

The *successive over-relaxation* (SOR) iteration is a variation of this which has an over-relaxation parameter $\omega \geq 1$, and pseudo-code for one iteration is shown in Algorithm 24. Using $\omega = 1$ is equivalent to the Gauss–Seidel method. The SOR

**Algorithm 24** Successive over-relaxation iteration for $A\boldsymbol{x} = \boldsymbol{b}$

```
1   function sor(A, x, b, ω)
2     for i = 1, 2, ..., n
3       s ← b_i
4       for j = 1, 2, ..., n
5         if j ≠ i
6           s ← s − a_ij x_j
7         end if
8       end
9       x_i ← (1 − ω)x_i + ωs/a_ii
10    end for
11    return x
12  end function
```

iteration can be represented in matrix–vector form as

$$\boldsymbol{x}_{k+1} \leftarrow (D + \omega L)^{-1} \left[\omega \boldsymbol{b} - (\omega U + (\omega - 1)D)\boldsymbol{x}_k\right] \tag{2.4.4}$$

with the iteration matrix $G_{SOR} = -(D + \omega L)^{-1}(\omega U + (\omega - 1)D)$. The number of floating point operations per iteration for the SOR method is $\leq 2\, nnz(A) + 2\, n$ where $A$ is $n \times n$ and $nnz(A)$ is the number of non-zero entries of $A$.

There are block versions of these algorithms where instead of $a_{ij}$ being a scalar, we treat $A_{ij}$ as a block in the matrix $A$, while $\boldsymbol{x} = [\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \ldots, \boldsymbol{x}_n^T]^T$ and $\boldsymbol{b} = [\boldsymbol{b}_1^T, \boldsymbol{b}_2^T, \ldots, \boldsymbol{b}_N^T]^T$ have blocks consistent with the blocks $A_{ij}$. The sums $\boldsymbol{s}$ become block vectors, and division by $a_{ii}$ is replaced by pre-multiplication by $A_{ii}^{-1}$. Many iterative algorithms for solving large linear systems can be understood as block Jacobi, block Gauss–Seidel, or block SOR iterations.

#### 2.4.1.1 Convergence Analysis

The convergence of the iteration (2.4.1) depends on the matrix $G := M^{-1}N$. For an iteration

$$\begin{aligned}
z_{k+1} &\leftarrow G\, z_k + \boldsymbol{y} \qquad \text{we have} \\
z_1 &= G\, z_0 + \boldsymbol{y}, \\
z_2 &= G(G\, z_0 + \boldsymbol{y}) + \boldsymbol{y} = G^2 z_0 + (G + I)\boldsymbol{y}, \\
z_3 &= G(G^2 z_0 + (G + I)\boldsymbol{y}) + \boldsymbol{y} \\
&= G^3 z_0 + (G^2 + G + I)\boldsymbol{y}, \\
&\vdots \\
z_k &= G^k z_0 + (G^{k-1} + \cdots + G + I)\boldsymbol{y}.
\end{aligned}$$

This converges if $\|G\| < 1$ for some matrix norm $\|\cdot\|$, so $\left\|G^k\right\| \leq \|G\|^k \to 0$, and $G^{k-1} + \cdots + G + I \to (I - G)^{-1}$ as $k \to \infty$ (see Lemma 2.2). There is a more precise convergence theorem.

**Theorem 2.15** *The matrix $G$ has the property that $G^k \to 0$ as $k \to \infty$ if and only if $\rho(G) < 1$ where $\rho(G)$ is the spectral radius*

$$\rho(G) = \max\{\,|\lambda| : \lambda \text{ is an eigenvalue of } A\,\}. \tag{2.4.5}$$

*Furthermore, if $\rho(G) < 1$ then $I + G + G^2 + \cdots$ converges and is equal to $(I - G)^{-1}$.*

This result is a consequence of the following theorem:

**Theorem 2.16** *For any square matrix $A$ and $\epsilon > 0$, there is an induced matrix norm $\|\cdot\|_{(\epsilon)}$ where*

$$\|A\|_{(\epsilon)} \leq \rho(A) + \epsilon.$$

Note that for any induced matrix norm and square matrix $A$, $\rho(A) \leq \|A\|$.

Theorem 2.16 implies that if $\rho(G) < 1$ then for $\epsilon = (1 - \rho(G))/2$ there is an induced matrix norm $\|\cdot\|_{(\epsilon)}$ where $\|G\|_{(\epsilon)} \leq \rho(G) + \epsilon = \frac{1}{2}(1 + \rho(G)) < 1$, and so $G^k \to 0$ as $k \to \infty$ and $I + G + G^2 + \cdots = (I - G)^{-1}$ by Lemma 2.2. The proof of Theorem 2.16 we develop here uses the Schur decomposition of Section 2.5.2.

***Proof*** Note that if $A \mapsto \|A\|$ is a matrix norm induced by a suitable vector norm $\boldsymbol{v} \mapsto \|\boldsymbol{v}\|$ then for invertible $X$, $A \mapsto \left\|X\,A\,X^{-1}\right\|$ is the matrix norm induced by the norm $\boldsymbol{v} \mapsto \|X\boldsymbol{v}\|$. To see this, let $\|\boldsymbol{v}\|_X = \|X\boldsymbol{v}\|$. Then

$$\begin{aligned}
\|A\|_X &= \max_{z \neq \mathbf{0}} \frac{\|Az\|_X}{\|z\|_X} = \max_{z \neq \mathbf{0}} \frac{\|XAz\|}{\|Xz\|} \\
&= \max_{\boldsymbol{y} \neq \mathbf{0}} \frac{\left\|XAX^{-1}\boldsymbol{y}\right\|}{\|\boldsymbol{y}\|} \qquad \text{where } Xz = \boldsymbol{y} \\
&= \left\|XAX^{-1}\right\|.
\end{aligned}$$

Now, by the Schur decomposition (Theorem 2.23), there is a unitary matrix $U$ where $\overline{U}^T A\,U = T$ with $T$ upper triangular. Note that $\overline{U}^T = U^{-1}$. Write $T = D + N$ where $D$ is the diagonal part of $T$ so each $d_{jj}$ is an eigenvalue of $A$. The matrix $N$ is strictly upper triangular ($n_{ij} = 0$ for all $i \geq j$). Let $S(\alpha)$ be the diagonal matrix with diagonal entries $s_{jj}(\alpha) = \alpha^j$. Then $\left(S(\alpha)^{-1} N\,S(\alpha)\right)_{ij} = \alpha^{j-i} n_{ij}$ which is zero if $i \geq j$ and goes to zero as $\alpha \downarrow 0$ if $i < j$. That is, $S(\alpha)^{-1} N\,S(\alpha) \to 0$ as $\alpha \downarrow 0$. On the other hand, since $D$ is diagonal and diagonal matrices commute, $S(\alpha)^{-1} D\,S(\alpha) = D$. Choose $\alpha > 0$ sufficiently small so that $\left\|S(\alpha)^{-1} N\,S(\alpha)\right\|_2 < \epsilon$. Let $X = S(\alpha)^{-1} U^{-1}$; then

$$\begin{aligned} XAX^{-1} &= S(\alpha)^{-1}U^{-1}A\,U\,S(\alpha) = S(\alpha)^{-1}T\,S(\alpha) \\ &= S(\alpha)^{-1}T\,S(\alpha) = D + S(\alpha)^{-1}N\,S(\alpha), \text{ and} \\ \left\|XAX^{-1}\right\|_2 &\le \|D\|_2 + \left\|S(\alpha)^{-1}N\,S(\alpha)\right\|_2 < \|D\|_2 + \epsilon. \end{aligned}$$

Recall that since $D$ is diagonal, $\|D\|_2 = \sqrt{\lambda_{\max}(\overline{D}^T D)} = \max_k |d_{kk}| = \rho(D)$, its spectral radius. But since the eigenvalues of $A$ are the eigenvalues of $D$, $\|D\|_2 = \rho(A)$. Thus in the induced matrix norm for the vector norm $\boldsymbol{v} \mapsto \|X\boldsymbol{v}\|_2$,

$$\|A\|_{X,2} = \left\|XAX^{-1}\right\|_2 \le \rho(A) + \epsilon,$$

as we wanted. □

Note that in any induced matrix norm $\rho(A) \le \|A\|$; if $A\boldsymbol{v} = \lambda\boldsymbol{v}$ and $\boldsymbol{v} \neq \boldsymbol{0}$, $|\lambda|\,\|\boldsymbol{v}\| = \|\lambda\boldsymbol{v}\| = \|A\boldsymbol{v}\| \le \|A\|\,\|\boldsymbol{v}\|$ so $|\lambda| \le \|A\|$. Taking the maximum over all eigenvalues $\lambda$ gives $\rho(A) \le \|A\|$.

So the iteration $\boldsymbol{z}_{k+1} \leftarrow G\,\boldsymbol{z}_k + \boldsymbol{y}$ is convergent if and only if $\rho(G) < 1$. If $\rho(G) \ge 1$ there must be an eigenvalue $\lambda$ with $|\lambda| \ge 1$ and an eigenvector $\boldsymbol{v} \neq \boldsymbol{0}$ where $A\boldsymbol{v} = \lambda\boldsymbol{v}$; then if $\boldsymbol{z}^* = G\,\boldsymbol{z}^* + \boldsymbol{y}$ we have $\boldsymbol{z}_{k+1} - \boldsymbol{z}^* = G(\boldsymbol{z}_k - \boldsymbol{z}^*)$. If $\boldsymbol{z}_0 = \boldsymbol{z}^* + \boldsymbol{v}$ then $\boldsymbol{z}_k = \boldsymbol{z}^* + G^k\boldsymbol{v} = \boldsymbol{z}^* + \lambda^k\boldsymbol{v} \not\to \boldsymbol{z}^*$. The iteration does not converge.

Returning to the iteration (2.4.1), we see it is convergent if and only if $\rho(M^{-1}N) < 1$.

Convergence can often be easily shown for certain classes of matrices. Consider first strictly row dominant matrices: $A$ is strictly row dominant if

$$|a_{ii}| > \sum_{j:j\neq i} |a_{ij}| \qquad \text{for all } i.$$

Non-strictly dominant matrices are defined in (2.1.6). We can immediately see that the Jacobi iteration is convergent as

$$\rho(G_J) \le \|G_J\|_\infty = \|-D(L+U)\|_\infty = \max_i \frac{1}{|a_{ii}|} \sum_{j:j\neq i} |a_{ij}| < 1.$$

Gauss–Seidel is also convergent in this case as $\rho(G_{GS}) = \rho(-(D+L)^{-1}U) \ge 1$ implies that there is an eigenvalue $\lambda$ with $|\lambda| \ge 1$. But if $-(D+L)^{-1}U\boldsymbol{v} = \lambda\boldsymbol{v}$ with $\boldsymbol{v} \neq \boldsymbol{0}$, this would mean that $\lambda^{-1}U\boldsymbol{v} = -(D+L)\boldsymbol{v}$ and $D\boldsymbol{v} = -(L + \lambda^{-1}U)\boldsymbol{v}$. Then

$$-\lambda^{-1}\sum_{j:j>i} a_{ij}v_j - \sum_{j:j<i} a_{ij}v_j = a_{ii}v_i.$$

In particular, this is true for the index $i$ where $|v_i| = \max_\ell |v_\ell| = \|\boldsymbol{v}\|_\infty$. Then taking absolute values of both sides,

$$|\lambda|^{-1}\sum_{j:j>i} |a_{ij}|\,|v_j| + \sum_{j:j<i} |a_{ij}|\,|v_j| \ge |a_{ii}|\,|v_i| = |a_{ii}|\,\|\boldsymbol{v}\|_\infty.$$

Dividing both sides by $\|\boldsymbol{v}\|_\infty$ and using $\left|v_j\right| / \|\boldsymbol{v}\|_\infty \leq 1$, we get

$$|a_{ii}| \leq |\lambda|^{-1} \sum_{j:j>i} \left|a_{ij}\right| + \sum_{j:j<i} \left|a_{ij}\right|,$$

which contradicts strict dominance as $|\lambda|^{-1} \leq 1$. Thus there can be no eigenvalue of $G$ of magnitude $\geq 1$ for Gauss–Seidel applied to a strictly row dominant matrix.

While strict diagonal dominance is sufficient for convergence, it is far from necessary. For these methods, if $A$ is symmetric positive definite then the methods converge. The best way to show this is via the Householder–John theorem:

**Theorem 2.17** *If $A$ and $M + \overline{M}^T - A$ are complex Hermitian positive definite, then $\rho(I - M^{-1}A) < 1$.*

***Proof*** Suppose that $(I - M^{-1}A)\boldsymbol{v} = \lambda \boldsymbol{v}$ with $\boldsymbol{v} \neq \boldsymbol{0}$. Then $(1-\lambda)\boldsymbol{v} = M^{-1}A\boldsymbol{v}$ so $(1-\lambda)M\boldsymbol{v} = A\boldsymbol{v}$. Since $A$ is positive definite, $A$ is invertible, so $\lambda \neq 1$. Thus $M\boldsymbol{v} = (1-\lambda)^{-1}A\boldsymbol{v}$. Pre-multiplying by $\overline{\boldsymbol{v}}^T$ gives $\overline{\boldsymbol{v}}^T M\boldsymbol{v} = (1-\lambda)^{-1}\overline{\boldsymbol{v}}^T A\boldsymbol{v}$. Taking conjugate transposes using $\overline{A}^T = A$ gives $\overline{\boldsymbol{v}}^T \overline{M}^T \boldsymbol{v} = (1-\overline{\lambda})^{-1}\overline{\boldsymbol{v}}^T A\boldsymbol{v}$. Then

$$\begin{aligned}\overline{\boldsymbol{v}}^T \left[M + \overline{M}^T - A\right]\boldsymbol{v} &= \overline{\boldsymbol{v}}^T A\boldsymbol{v}\left[\frac{1}{1-\lambda} + \frac{1}{1-\overline{\lambda}} - 1\right] \\ &= \overline{\boldsymbol{v}}^T A\boldsymbol{v}\frac{1-|\lambda|^2}{|1-\lambda|^2}.\end{aligned}$$

Since the left-hand side is positive and $\overline{\boldsymbol{v}}^T A\boldsymbol{v}$ is positive, $1 - |\lambda|^2 > 0$. That is $|\lambda| < 1$. Therefore $\rho(I - M^{-1}A) < 1$. □

This theorem can be used to show convergence of the Jordan and Gauss–Seidel methods.

**Theorem 2.18** *If $A$ is Hermitian positive definite then the Gauss–Seidel iteration converges. If, in addition, the matrix $2D - A$ is positive definite where $D$ is the diagonal part of $A$, then the Jacobi iteration converges.*

***Proof*** Let $D$ be the diagonal part of $A$, $L$ the strictly lower triangular part of $A$, and $U = \overline{L}^T$ the strictly upper triangular part of $A$. For the Gauss–Seidel method, $M = D + L$ and the iteration matrix is $G_{GS} = -(D+L)^{-1}U = I - (D+L)^{-1}A = I - M^{-1}A$. To show that the conditions of Theorem 2.17 hold, we first note that $A$ is Hermitian positive definite. We also need to show that $M + \overline{M}^T - A$ is positive definite. Now $M + \overline{M}^T = D + L + \overline{D}^T + \overline{L}^T = 2D + L + U$ so $M + \overline{M}^T - A = 2D + L + U - (D + L + U) = D$ which is positive definite as each $d_{jj} > 0$ by positive definiteness of $A$. Therefore, we can apply Theorem 2.17 to conclude that $\rho(G_{GS}) < 1$.

For the Jacobi method, $M = D = \overline{D}^T$ so $M + \overline{M}^T - A = 2D - A$ which is positive definite by assumption. As $A$ is also positive definite by assumption, we can

apply Theorem 2.17 to conclude that $\rho(G_J) = \rho(I - D^{-1}A) < 1$ and so the Jacobi method is convergent. □

**Example 2.19** To illustrate how these methods work in practice, consider the problem of solving a discrete approximation to the Poisson equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \qquad (x, y) \in [0, 1] \times [0, 1] \tag{2.4.6}$$

with zero boundary conditions: $u(x, 0) = u(x, 1) = u(0, y) = u(1, y) = 0$ for all $x,\ y \in [0, 1]$. Using the discrete approximations

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) \approx \frac{u(x_{i+1}, y_j) - 2\,u(x_i, y_j) + u(x_{i-1}, y_j)}{(\Delta x)^2},$$

where $x_i = i\,\Delta x$ and $y_j = j\,\Delta x$ and $\Delta x = 1/(N + 1)$, we obtain the equations for the approximate values $u_{i,j} \approx u(x_i, y_j)$:

$$\frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{(\Delta x)^2} = f(x_i, y_j), \tag{2.4.7}$$

taking $u_{i,j} = 0$ for $i = 0$ or $i = N + 1$ or $j = 0$ or $j = N + 1$ for the zero boundary conditions.

This example is used with the right-hand side $f(x, y) = 1$ for all $(x, y)$ to generate the results shown in Figure 2.4.1.

As can be clearly seen from Figure 2.4.1, for this example, the Gauss–Seidel iteration converges faster than the Jacobi iteration. The residual norms for the Jacobi

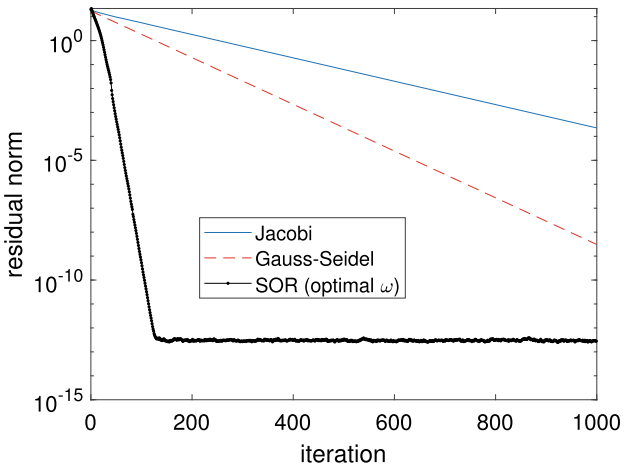



**Fig. 2.4.1** Results for classical iterations

iteration are used to estimate $\rho(G_J)$. For a class of matrices having "Property A" [265] in addition being symmetric positive definite, Gauss–Seidel is twice as fast as the Jacobi method: $\rho(G_{GS}) = \rho(G_J)^2$. Property A is that the graph of the $A - D$ is a bipartite graph, that is, the vertices of the graph of $A - D$ can be split into two disjoint subsets $V = V_1 \cup V_2$ where every edge joins a vertex in $V_1$ with a vertex in $V_2$. If the rows and columns are ordered consistently with this partition of vertices, then we can compute the optimal $\omega$ for the SOR iteration (2.4.4) which is given by [225, p. 112-116]:

$$\omega_{\text{opt}} = \frac{1}{1 + \sqrt{1 - \rho(G_J)^2}}. \tag{2.4.8}$$

This value of $\omega$ is optimal in the sense of minimizing $\rho(G_{SOR})$. Using this value from the estimate of $\rho(G_J)$ gives the results shown in Figure 2.4.1 for the SOR iteration.

## 2.4.2 Conjugate Gradients and Krylov Subspaces

The *conjugate gradient* method was first published by Hestenes and Stiefel [122] in 1952. It applies to symmetric positive-definite linear systems $A\boldsymbol{x} = \boldsymbol{b}$. The derivation of the conjugate gradient algorithm is often framed as minimizing a quadratic convex function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$; this is equivalent to the linear system $A\boldsymbol{x} = \boldsymbol{b}$ provided $A$ is symmetric positive definite.

The conjugate gradient method was initially considered to be a candidate for a general-purpose direct solver for solving symmetric positive-definite linear systems as in exact arithmetic it should solve an $n \times n$ linear system in $n$ conjugate gradient steps. Numerical issues and a certain kind of numerical instability meant that it did not work well in this way. However, as an iterative method, it works very well, often taking far fewer than $n$ steps to give sufficiently accurate solutions. While the performance of the conjugate gradient method is generally good, it can be improved by the use of preconditioners.

Of particular importance for the success of the conjugate gradient method is that the method only requires the ability to compute matrix–vector products $A\boldsymbol{z}$ for given vectors $\boldsymbol{z}$. This means that *functional representations* of $A$ can be used: $A$ is represented by a function $\boldsymbol{x} \mapsto A\boldsymbol{x}$. This is useful, not just for sparse matrices, but for many other matrices that can be represented efficiently, for example, by using a combination of low-rank matrices and discrete Fourier transforms as well as sparse matrices.

### 2.4.2.1 Derivation and Properties of the Conjugate Gradient Method

Minimizing a convex quadratic function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$$

is equivalent to solving the linear system

$$\nabla f(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}.$$

For positive definite $A$, this can be done using a method called the conjugate gradient method that has close ties to optimization. The basic idea is to write

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \alpha_0 \boldsymbol{p}_0 + \alpha_1 \boldsymbol{p}_1 + \cdots + \alpha_{k-1} \boldsymbol{p}_{k-1},$$

where the $\alpha_i$'s are scalars and the $\boldsymbol{p}_i$'s are search directions. Then $f(\boldsymbol{x}_k)$ is a quadratic function of the $\alpha_i$'s, and finding the minimizing $\alpha_i$'s can be done by solving a linear system. But linear systems can be most easily solved when they are *diagonal*. The linear system for the minimizing $\alpha_i$'s is

$$\boldsymbol{p}_i^T \left[ A(\boldsymbol{x}_0 + \sum_{j=0}^{k-1} \alpha_j \boldsymbol{p}_j) - \boldsymbol{b} \right] = 0 \qquad \text{for } i = 0,\ 1,\ 2,\ \ldots,\ k-1.$$

The linear system is $k \times k$ with $(i, j)$ entry given by $\boldsymbol{p}_i^T A \boldsymbol{p}_j$. This matrix is diagonal if

$$\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0 \qquad \text{for all } i \neq j. \tag{2.4.9}$$

This is the condition that $\boldsymbol{p}_i$'s are *A-conjugate,* or just *conjugate* if $A$ is understood.

*If* the $\boldsymbol{p}_i$'s are conjugate (with respect to $A$) then the system of linear equations for the $\alpha_i$'s becomes simply

$$\boldsymbol{p}_i^T A \boldsymbol{p}_i \, \alpha_i = \boldsymbol{p}_i^T (\boldsymbol{b} - A\boldsymbol{x}_0) = \boldsymbol{p}_i^T \left[ \boldsymbol{b} - A(\boldsymbol{x}_0 + \sum_{j=0}^{i-1} \alpha_j \boldsymbol{p}_j) \right].$$

Note that increasing $k$ does not change the value of $\alpha_i$ for $i \leq k$. This means that $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$. Note that $\boldsymbol{x}_k$ minimizes $f(\boldsymbol{x}_0 + \sum_{i=0}^{k-1} \alpha_i \boldsymbol{p}_i)$ over all $\alpha_i$'s, that is, $\boldsymbol{x}_k$ minimizes $f(z)$ over all $z \in \boldsymbol{x}_0 + \text{span}\{\, \boldsymbol{p}_0,\ \ldots,\ \boldsymbol{p}_{k-1} \}$. This is the conjugate gradient minimization property.

Let $\boldsymbol{r}_k = \nabla f(\boldsymbol{x}_k) = A\boldsymbol{x}_k - \boldsymbol{b}$. In optimization, this is clearly the gradient; in linear algebra, it is called the *residual* for $\boldsymbol{x}_k$.

If we had a sequence $\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots$ of conjugate vectors then we could design an iterative algorithm for minimizing $f(\boldsymbol{x})$ as shown in Algorithm 25.

The problem now is to find out how to generate the conjugate $\boldsymbol{p}_i$'s.

At the beginning, any $\boldsymbol{p}_0 \neq \boldsymbol{0}$ by itself is conjugate. So we have a place to start. We can then proceed using mathematical induction. Now let's suppose we have generated $\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k$ which are, so far, all conjugate. We will also show that the residuals $\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_k$ are all orthogonal, and that $\text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_k\} = \text{span}\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k\} = \text{span}\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ A^2\boldsymbol{r}_0,\ \ldots,\ A^k\boldsymbol{r}_0\}$. These will be our

**Algorithm 25** Minimization with given conjugate directions

```
function conjdirns(A, b, x_0, (p_0, p_1, p_2, ...))
  for k ← 0, 1, 2, ..., n
    r_k ← A x_k − b
    α_k ← −p_k^T r_k / p_k^T A p_k
    x_{k+1} ← x_k + α_k p_k
  end for
  return x_n
end function
```

**Algorithm 26** Conjugate gradients algorithm — version 1

```
function conjgrad1(A, b, x_0, ε)
  while ‖A x_k − b‖_2 ≥ ε
    r_k ← A x_k − b
    if k = 0: p_k ← −r_k end if
    α_k ← −p_k^T r_k / p_k^T A p_k
    x_{k+1} ← x_k + α_k p_k
    β_k ← r_{k+1}^T A p_k / p_k^T A p_k
    p_{k+1} ← −r_{k+1} + β_k p_k
    k ← k + 1
  end while
  return x_k
end function
```

induction hypotheses. Note that this last subspace

$$\mathcal{K}_k(A, \boldsymbol{r}_0) = \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ A^2\boldsymbol{r}_0,\ \ldots,\ A^k\boldsymbol{r}_0\right\} \tag{2.4.10}$$

is the *Krylov subspace* generated by the original residual $\boldsymbol{r}_0$. Note that $\boldsymbol{p}_0 = -\boldsymbol{r}_0$ so all the induction hypotheses are true for $k = 0$. Now we want to find $\boldsymbol{p}_{k+1}$ so that $\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k,\ \boldsymbol{p}_{k+1}$ are all conjugate, and the other induction hypotheses are true with $k$ replaced by $k+1$.

How can we generate $\boldsymbol{p}_{k+1}$? We can compute $\boldsymbol{x}_{k+1}$ from $\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k$ so we can compute $\boldsymbol{r}_{k+1}$. We are going to suppose that we have a particular form $\boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$ for $\boldsymbol{p}_{k+1}$. The amazing thing is that this will work. But more on that later. We can find $\beta_k$ by requiring conjugacy between $\boldsymbol{p}_k$ and $\boldsymbol{p}_{k+1}$ using symmetry of $A$ so that $\boldsymbol{p}_k^T A\boldsymbol{r}_{k+1} = \boldsymbol{r}_{k+1}^T A^T \boldsymbol{p}_k = \boldsymbol{r}_{k+1}^T A\boldsymbol{p}_k$:

$$\begin{aligned} 0 = \boldsymbol{p}_k^T A\boldsymbol{p}_{k+1} &= \boldsymbol{p}_k^T A(-\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k) \\ &= -\boldsymbol{r}_{k+1}^T A\boldsymbol{p}_k + \beta_k\, \boldsymbol{p}_k^T A\boldsymbol{p}_k \qquad \text{so} \\ \beta_k &= \boldsymbol{r}_{k+1}^T A\boldsymbol{p}_k / \boldsymbol{p}_k^T A\boldsymbol{p}_k. \end{aligned} \tag{2.4.11}$$

This ensures conjugacy between *two* vectors: $\boldsymbol{p}_k^T A\boldsymbol{p}_{k+1} = 0$. But it is enough, as we will see.

The algorithm with this way of generating the conjugate directions is the conjugate gradient method which is shown in Algorithm 26.

There are several different ways of representing the conjugate gradient method for linear symmetric positive-definite systems of equations; they are all equivalent because of the many relationships between the $\boldsymbol{r}_i$'s and $\boldsymbol{p}_i$'s.

**Theorem 2.20** *In Algorithm 26, for each $k = 0, 1, 2, \ldots$ and $i, j \leq k$,*

$$\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0 \quad \text{for all } i \neq j, \tag{2.4.12}$$

$$\boldsymbol{r}_i^T \boldsymbol{r}_j = 0 \quad \text{for all } i \neq j, \tag{2.4.13}$$

$$\text{span}\{\boldsymbol{r}_0, \boldsymbol{r}_1, \ldots, \boldsymbol{r}_k\} = \text{span}\left\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\right\} = \text{span}\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, \boldsymbol{r}_0\}. \tag{2.4.14}$$

***Proof*** We use proof by induction on $k$.

**Base case:** For $k = 0$, we have (2.4.12) and (2.4.13) holding trivially, while (2.4.14) holds because $\boldsymbol{p}_0 = -\boldsymbol{r}_0$.

**Induction step:** Suppose (2.4.12, 2.4.13, 2.4.14) hold for $k = m$. We show that these hold for $k = m + 1$.

First note that $\boldsymbol{r}_{m+1} \in \text{span}\left\{\boldsymbol{p}_m,\ \boldsymbol{p}_{m+1}\right\} \subseteq \text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_{m+1}\right\}$. On the other hand,

$$\begin{aligned}\boldsymbol{p}_{m+1} &= -\boldsymbol{r}_{m+1} + \beta_m \boldsymbol{p}_m \in -\boldsymbol{r}_{m+1} + \beta_m \text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_m\} \\ &\subseteq \text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_{m+1}\}.\end{aligned}$$

So $\text{span}\left\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_m,\ \boldsymbol{r}_{m+1}\right\} = \text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_m,\ \boldsymbol{p}_{m+1}\right\}$. Also $\boldsymbol{r}_{m+1}^T \boldsymbol{p}_j = 0$ for all $j \leq m$ from our optimality result.

That is, $\boldsymbol{r}_{m+1}$ is orthogonal to $\text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_m\right\} = \text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_m\}$; this means that $\boldsymbol{r}_{m+1}^T \boldsymbol{r}_j = 0$ for $j \leq m$. Note that $\boldsymbol{r}_{m+1} = \boldsymbol{r}_m + \alpha_m A \boldsymbol{p}_m$ (since $\boldsymbol{x}_{m+1} = \boldsymbol{x}_m + \alpha_m \boldsymbol{p}_m$). Then

$$\begin{aligned}\boldsymbol{r}_{m+1} &\in \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^m \boldsymbol{r}_0\right\} + \alpha_m A\, \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^m \boldsymbol{r}_0\right\} \\ &\subseteq \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^m \boldsymbol{r}_0,\ A^{m+1}\boldsymbol{r}_0\right\}.\end{aligned}$$

On the other hand, $A^{m+1}\boldsymbol{r}_0 = A\left(A^m \boldsymbol{r}_0\right)$ and $A^m \boldsymbol{r}_0 \in \text{span}\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^m \boldsymbol{r}_0\} = \text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_m\right\}$, so we can write $A^m \boldsymbol{r}_0 = \sum_{j=0}^m \gamma_j \boldsymbol{p}_j$, and $A^{m+1}\boldsymbol{r}_0 = \sum_{j=0}^m \gamma_j A \boldsymbol{p}_j = \sum_{j=0}^m \gamma_j \alpha_j^{-1}(\boldsymbol{r}_{j+1} - \boldsymbol{r}_j) \in \text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_{m+1}\}$. (Note that $\alpha_j \neq 0$ since $\alpha_j = 0$ implies that $\boldsymbol{r}_{j+1} = \boldsymbol{r}_j$; orthogonality of the residuals then implies that $0 = \boldsymbol{r}_{j+1}^T \boldsymbol{r}_j = \boldsymbol{r}_j^T \boldsymbol{r}_j$ and $\boldsymbol{r}_j = \boldsymbol{0}$, so the algorithm must have already terminated.) Thus $\text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_{m+1}\} = \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^{m+1}\boldsymbol{r}_0\right\}$.

Now we show $\boldsymbol{p}_i^T A \boldsymbol{p}_{m+1} = 0$ for all $i < m$. For $i < m$, $\boldsymbol{p}_i \in \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^i \boldsymbol{r}_0\right\}$, so $A\boldsymbol{p}_i \in \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^i \boldsymbol{r}_0,\ A^{i+1}\boldsymbol{r}_0\right\} = \text{span}\{\boldsymbol{r}_0,\ \boldsymbol{r}_1,\ \ldots,\ \boldsymbol{r}_{i+1}\}$. Note that $\boldsymbol{p}_i^T A \boldsymbol{p}_{m+1} = \boldsymbol{p}_{m+1}^T A \boldsymbol{p}_i$ (since $A$ symmetric). Then $\boldsymbol{p}_{m+1}^T A \boldsymbol{p}_i = -\boldsymbol{r}_{m+1}^T A \boldsymbol{p}_i + \beta_m \boldsymbol{p}_m^T A \boldsymbol{p}_i$ by the formula for $\boldsymbol{p}_{m+1}$. But since $i < m$, $\boldsymbol{r}_{m+1}^T A \boldsymbol{p}_i = 0$ as $\boldsymbol{r}_{m+1}^T \boldsymbol{r}_j = 0$ for all $j \leq m$, and $\boldsymbol{p}_m^T A \boldsymbol{p}_i = 0$ for $i < m$ by the induction hypothesis. Thus $\boldsymbol{p}_{m+1}^T A \boldsymbol{p}_i = 0$ provided $i + 1 \leq m$, that is, for

**Algorithm 27** Conjugate gradients algorithm—version 2

```
1   function conjgrad2(A, b, x_0, ε)
2     k ← 0;  r_0 ← Ax_0 − b;  p_0 ← −r_0
3     while ‖r_k‖_2 ≥ ε
4       q_k ← Ap_k
5       α_k ← −p_k^T r_k / p_k^T q_k
6       x_{k+1} ← x_k + α_k p_k
7       r_{k+1} ← r_k + α_k q_k
8       β_k ← r_{k+1}^T r_{k+1} / r_k^T r_k
9       p_{k+1} ← −r_{k+1} + β_k p_k
10      k ← k + 1
11    end while
12    return x_k
13  end function
```

$i < m$ we have $\boldsymbol{p}_i^T A \boldsymbol{p}_{m+1} = 0$. As we have already shown that $\boldsymbol{p}_m^T A \boldsymbol{p}_{m+1}$ we see that $\boldsymbol{p}_0$, $\boldsymbol{p}_1$, …, $\boldsymbol{p}_m$, $\boldsymbol{p}_{m+1}$ are conjugate, as we wanted. Thus, all the induction hypotheses have been shown to be true for $k$ replaced by $m+1$.

Thus, by the principle of induction, the result is proven for $k = 0, 1, 2, \ldots$. □

#### 2.4.2.2 Reformulating the Algorithm

The formula for $\beta_{k+1} = \boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k \,/\, \boldsymbol{p}_k^T A \boldsymbol{p}_k$ can be modified using the properties that have been discovered, such as the orthogonality of the $\boldsymbol{r}_k$'s, and $\boldsymbol{r}_{k+1} = A(\boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k) - \boldsymbol{b} = \boldsymbol{r}_k + \alpha_k A \boldsymbol{p}_k$. So $A\boldsymbol{p}_k = (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)/\alpha_k$. Therefore,

$$\begin{aligned} \boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k &= \boldsymbol{r}_{k+1}^T (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)/\alpha_k \qquad \text{and} \\ \boldsymbol{p}_k^T A \boldsymbol{p}_k &= \boldsymbol{p}_k^T (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)/\alpha_k = -\boldsymbol{p}_k^T \boldsymbol{r}_k/\alpha_k \end{aligned}$$

since $\boldsymbol{r}_{k+1}$ is orthogonal to $\boldsymbol{p}_k$. But $\boldsymbol{p}_k = -\boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1}$ so $\boldsymbol{p}_k^T \boldsymbol{r}_k = (-\boldsymbol{r}_k + \beta_{k-1}\boldsymbol{p}_{k-1})^T \boldsymbol{r}_k = -\boldsymbol{r}_k^T \boldsymbol{r}_k$ as $\boldsymbol{r}_k^T \boldsymbol{p}_{k-1} = 0$. By orthogonality of the $\boldsymbol{r}_j$'s, $\boldsymbol{r}_{k+1}^T \boldsymbol{r}_k = 0$. So $\boldsymbol{r}_{k+1}^T A \boldsymbol{p}_k / \boldsymbol{p}_k^T A \boldsymbol{p}_k = \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k$.

Using the update $\boldsymbol{r}_{k+1} \leftarrow \boldsymbol{r}_k + \alpha_k A \boldsymbol{p}_k$ instead of using $\boldsymbol{r}_{k+1} \leftarrow A\boldsymbol{x}_{k+1} - \boldsymbol{b}$, this new formula for $\beta_{k+1}$, and $\boldsymbol{q}_k = A\boldsymbol{p}_k$ gives Algorithm 27. This makes clear that there is only one matrix–vector multiplication per loop. The matrix $A$ can be represented by a function, and this function is only called once per loop. Apart from the representation of the matrix $A$, the only vectors needed to be stored are $\boldsymbol{x}_k$, $\boldsymbol{r}_k$, $\boldsymbol{p}_k$, and $\boldsymbol{q}_k$. That is, to solve an $n \times n$ linear system, apart from the memory needed for a functional representation of $A$, the amount of memory needed for conjugate gradients is $\approx 4n$ floating point numbers. This means that extremely large problems can be solved using conjugate gradients.

### 2.4.2.3 Rate of Convergence

One of the important properties of the conjugate gradient algorithm is that $\boldsymbol{x}_{k+1}$ minimizes $f(\boldsymbol{x}) := \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x}$ over all $\boldsymbol{x} \in \boldsymbol{x}_0 + \text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k\right\}$. The other fact is that $\text{span}\left\{\boldsymbol{p}_0,\ \boldsymbol{p}_1,\ \ldots,\ \boldsymbol{p}_k\right\} = \text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^k\boldsymbol{r}_0\right\}$. Then $\boldsymbol{x}_k - \boldsymbol{x}_0$ is a linear combination of vectors $A^j\boldsymbol{r}_0$. Such a space $\text{span}\left\{\boldsymbol{r}_0,\ A\boldsymbol{r}_0,\ \ldots,\ A^k\boldsymbol{r}_0\right\}$ is called a *Krylov subspace*. This means that we can represent $\boldsymbol{x}_{k+1} - \boldsymbol{x}_0$ in terms of polynomials:

$$\boldsymbol{x}_{k+1} - \boldsymbol{x}_0 = \sum_{j=0}^{k} \gamma_j A^j \boldsymbol{r}_0 = \left(\sum_{j=0}^{k} \gamma_j A^j\right)\boldsymbol{r}_0 = g(A)\,\boldsymbol{r}_0,$$

where $g$ is a polynomial of degree $\leq k$. If $\boldsymbol{x}^*$ is the minimizer of $f$ (that is, $A\boldsymbol{x}^* = \boldsymbol{b}$), then $f(\boldsymbol{x}) - f(\boldsymbol{x}^*) = \frac{1}{2}\left\|\boldsymbol{x} - \boldsymbol{x}^*\right\|_A^2$ where $\|\boldsymbol{u}\|_A = \sqrt{\boldsymbol{u}^T A\boldsymbol{u}}$ is the norm generated by $A$.

We can study this better by using eigenvalues and eigenvectors of $A$:

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i, \qquad \|\boldsymbol{v}_i\|_2 = 1$$

with $\boldsymbol{v}_i^T\boldsymbol{v}_j = 0$ if $i \neq j$. Note that if $g$ is a polynomial, then $g(A)\boldsymbol{v}_i = g(\lambda_i)\,\boldsymbol{v}_i$. Let $\boldsymbol{x}_0 - \boldsymbol{x}^* = \sum_{j=1}^n c_j\boldsymbol{v}_j$. Then $\boldsymbol{r}_0 = A\boldsymbol{x}_0 - \boldsymbol{b} = A(\boldsymbol{x}_0 - \boldsymbol{x}^*) = \sum_{j=1}^n \lambda_j c_j \boldsymbol{v}_j$. Also, with $\boldsymbol{x}_{k+1} - \boldsymbol{x}_0 = g(A)\boldsymbol{r}_0 = g(A)A(\boldsymbol{x}_0 - \boldsymbol{x}^*) = \sum_{j=1}^n g(\lambda_j)\lambda_j c_j \boldsymbol{v}_j$ so that $\boldsymbol{x}_{k+1} - \boldsymbol{x}^* = \sum_{j=1}^n (1 + \lambda_j g(\lambda_j))c_j\boldsymbol{v}_j$, and

$$\begin{aligned}
\left\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\right\|_A^2 &= \sum_{j=1}^{n} \lambda_j \left[1 + \lambda_j g(\lambda_j)\right]^2 c_j^2 \\
&\leq \left(\max_j \left|1 + \lambda_j g(\lambda_j)\right|\right)^2 \sum_{j=1}^{n} \lambda_j\, c_j^2 \\
&= \left(\max_j \left|1 + \lambda_j g(\lambda_j)\right|\right)^2 \left\|\boldsymbol{x}_0 - \boldsymbol{x}^*\right\|_A^2 .
\end{aligned}$$

The polynomial $g$ of degree $\leq k$ is chosen to minimize $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|_A^2$. So we look for polynomials $g$ that make $\max_{\lambda\in\sigma(A)} |1 + \lambda g(\lambda)|$ small where $\lambda$ ranges over the set of all eigenvalues $\sigma(A)$ of $A$. If we write $q(\lambda) = 1 + \lambda g(\lambda)$ we note that $q$ is a polynomial of degree $\leq k + 1$ with $q(0) = 1$ (that is, the constant term in $q$ is 1). On the other hand, for any such $q$, $g(\lambda) = (q(\lambda) - 1)/\lambda$ is a polynomial of degree $\leq k$.

If $A$ has only a few distinct eigenvalues $\lambda_1,\ \lambda_2,\ \ldots,\ \lambda_r$, we can set

$$q(\lambda) = \frac{(\lambda - \lambda_1)(\lambda - \lambda_2)\cdots(\lambda - \lambda_r)}{(-\lambda_1)(-\lambda_2)\cdots(-\lambda_r)},$$

and we can see that $q(\lambda_j) = 0$ for all eigenvalues of $A$, and $\boldsymbol{x}_{r+1} = \boldsymbol{x}^*$. This would mean exact convergence in $r + 1$ steps. In practice, it rarely happens that a matrix has only a few exactly repeated eigenvalues. However, it may happen that the eigenvalues fall into a few small clusters. Then the choice of $q(\lambda)$ given above would make $\max_{\lambda\in\sigma(A)} |1 + \lambda g(\lambda)|$ small.

If we know nothing of the eigenvalues of $A$ except for the extreme eigenvalues $0 < \lambda_1$ (smallest) and $\lambda_n$ (largest), then we can try to minimize $\max_{\lambda\in[\lambda_1,\lambda_n]} |q(\lambda)|$ with the constraint that $q(0) = 1$. This leads to *Chebyshev polynomials* (4.6.3):

$$T_j(\cos\theta) = \cos(j\theta), \qquad j = 0,\ 1,\ 2,\ 3,\ \ldots.$$

It is not immediately obvious that these are, in fact, polynomials. However, it is easy to check that $T_0(x) = 1$, $T_1(x) = x$, and using the trigonometric addition formulas we can show that

$$T_{k+1}(x) = 2x\,T_k(x) - T_{k-1}(x), \qquad k = 1,\ 2,\ 3,\ \ldots$$

The nice property of these polynomials is that $\left|T_j(x)\right| \le 1$ whenever $|x| \le 1$. The optimal $q$ is given by

$$q(\lambda) = \frac{T_{k+1}(1 - 2(\lambda - \lambda_1)/(\lambda_n - \lambda_1))}{T_{k+1}((\lambda_n + \lambda_1)/(\lambda_n - \lambda_1))}.$$

Then for $\lambda_1 \le \lambda \le \lambda_n$, $|T_{k+1}(\lambda)| \le 1$. Thus $\max_{\lambda\in[\lambda_1,\lambda_n]} |q(\lambda)| = 1/T_{k+1}((\lambda_n + \lambda_1)/(\lambda_n - \lambda_1))$. Note that the formula $T_j(\cos\theta) = \cos(j\theta)$ will not work for $T_j(x)$ with $|x| > 1$. However, there is another formula that works here: $T_j(\cosh u) = \cosh(ju)$. For $\lambda_n/\lambda_1 \gg 1$, $(\lambda_n + \lambda_1)/(\lambda_n - \lambda_1) = 1 + 2\lambda_1/(\lambda_n - \lambda_1) \approx 1 + 2\lambda_1/\lambda_n$. Setting $(\lambda_n + \lambda_1)/(\lambda_n - \lambda_1) = \cosh u$ gives $u \approx 2\sqrt{\lambda_1/\lambda_n}$, and $T_{k+1}((\lambda_n + \lambda_1)/(\lambda_n - \lambda_1)) \approx \cosh(2(k+1)\sqrt{\lambda_1/\lambda_n}) \approx \frac{1}{2}(1 + \sqrt{\lambda_1/\lambda_n})^{2(k+1)}$. This means that the number of iterations needed to achieve a certain error tolerance $\epsilon$ is $\mathcal{O}(\log(1/\epsilon)\kappa_2(A)^{1/2})$ since for a symmetric positive-definite matrix $\kappa_2(A) = \lambda_n(A)/\lambda_1(A)$.

#### 2.4.2.4 Preconditioned Conjugate Gradient Algorithm

Because the rate of convergence depends on the condition number $\kappa_2(A)$, it is desirable to reformulate the system of equations so as to reduce this condition number. We could try to reformulate the system of equations $A\boldsymbol{x} = \boldsymbol{b}$ as

$$M^{-1}A\boldsymbol{x} = M^{-1}\boldsymbol{b}. \tag{2.4.15}$$

But, unless we are extremely lucky, we do not expect $M^{-1}A$ to be symmetric even if its condition number is much better. However, it is *self-adjoint* with respect to

**Algorithm 28** Preconditioned conjugate gradients

```
function pcconjgrad(A, M^{-1}, b, x_0, ε)
  k ← 0; r_0 ← Ax_0 − b; z_0 ← M^{-1}r_0; p_0 ← −z_0
  while ‖r_k‖_2 ≥ ε
    q_k ← Ap_k
    α_k ← −z_k^T r_k / p_k^T q_k
    x_{k+1} ← x_k + α_k p_k
    r_{k+1} ← r_k + α_k q_k
    z_{k+1} ← M^{-1} r_{k+1}
    β_k ← z_{k+1}^T r_{k+1} / z_k^T r_k
    p_{k+1} ← −z_{k+1} + β_k p_k
    k ← k + 1
  end while
  return x_k
end function
```

the inner product $(\boldsymbol{u},\ \boldsymbol{v})_M = \boldsymbol{u}^T M\boldsymbol{v}$. This defines an inner product provided $M$ is symmetric and positive definite. To see that it is self-adjoint note that

$$(\boldsymbol{u},\ M^{-1}A\boldsymbol{v})_M = \boldsymbol{u}^T MM^{-1}A\boldsymbol{v} = \boldsymbol{u}^T A\boldsymbol{v}, \qquad \text{and}$$
$$(M^{-1}A\boldsymbol{u},\ \boldsymbol{v})_M = (M^{-1}A\boldsymbol{u})^T M\boldsymbol{v} = \boldsymbol{u}^T A^T M^{-T} M\boldsymbol{v} = \boldsymbol{u}^T A\boldsymbol{v}$$

by symmetry of $A$ and $M$.

We aim to choose $M$ where

- $M$ is symmetric and positive definite,
- $M\boldsymbol{z} = \boldsymbol{y}$ can be solved for $\boldsymbol{z}$ easily,
- the function $\boldsymbol{z} \mapsto M^{-1}\boldsymbol{z}$ can be efficiently implemented,
- $\kappa_2(M^{-1}A)$ is much less than $\kappa_2(A)$.

We transform the code (setting $\boldsymbol{z}_j = M^{-1}\boldsymbol{r}_j = M^{-1}(A\boldsymbol{x}_j - \boldsymbol{b})$) from Algorithm 27, replacing $A$ with $M^{-1}A$ and $\boldsymbol{u}^T\boldsymbol{v}$ with $(\boldsymbol{u}, \boldsymbol{v})_M = \boldsymbol{u}^T M\boldsymbol{v}$. Note that this means that $\boldsymbol{p}^T A\boldsymbol{q}$ is replaced by $\boldsymbol{p}^T M\, M^{-1}A\boldsymbol{q} = \boldsymbol{p}^T A\boldsymbol{q}$. This gives Algorithm 28.

In Algorithm 28, we can use functional representation for both $A$ and $M^{-1}$ with one function evaluation for each of $A$ and $M^{-1}$ per iteration. Note that Algorithm 28 with $M = \alpha I$ for any $\alpha > 0$ is equivalent to Algorithm 27.

Much work has gone into determining how to create a good preconditioner, especially for specific families of linear systems. Readily implemented preconditioners for conjugate gradients include the symmetrized Gauss–Seidel preconditioner $M_{SGS}^{-1} = (D - U)^{-1}D(D - L)^{-1}$ where $D$ is the diagonal of $A$, $L$ is the strictly lower triangular part of $A$, and $U$ is the strictly upper triangular part of $A$. A similar symmetrized SOR preconditioner is given by $M_{SSOR}^{-1} = (D - \omega U)^{-1}D(D - \omega L)^{-1}$. The exact choice of the parameter $\omega$ is less crucial for its use as a preconditioner than for using the SOR method directly. Another possibility is to use incomplete sparse

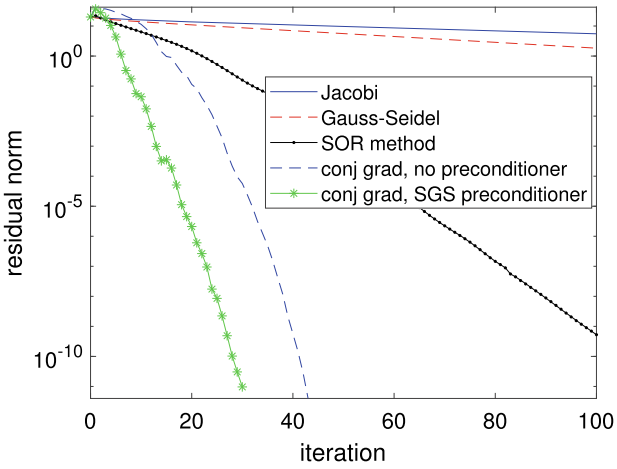



**Fig. 2.4.2** Comparison of classical iterative methods with conjugate gradients with and without preconditioning.

matrix factorizations. For other problems, like large systems that come from partial differential equations, specialized methods such as *multigrid methods* are very good.

If we compare the classical methods shown in Figure 2.4.1 with the results for conjugate gradients, with and without preconditioning, we can see the results in Figure 2.4.2. The preconditioner used is the symmetrized Gauss–Seidel (SGS) preconditioner mentioned above.

### *2.4.3 Non-symmetric Krylov Subspace Methods*

Krylov subspaces are natural vector spaces to consider for iterative methods. We suppose that we start with a vector $\boldsymbol{x}_0 \neq \mathbf{0}$, and the operations we are allowed to perform are linear combinations, inner products, and matrix–vector products $\boldsymbol{x} \mapsto A\boldsymbol{x}$. Of these, we will assume that computing matrix–vector products is the most computationally intensive operation. The question naturally arises: What is the set of vectors we can obtain with linear combinations and no more than $r$ matrix–vector products? The answer is $\mathcal{K}_r(A, \boldsymbol{x}_0) = \operatorname{span}\{\boldsymbol{x}_0, A\boldsymbol{x}_0, \ldots, A^r\boldsymbol{x}_0\}$, which is the Krylov subspace generated by $A$ and $\boldsymbol{x}_0$ of dimension $r + 1$. We will see how we can use this idea to develop methods for solving equations and also finding eigenvalues and eigenvectors (see Section 2.5.5).

### 2.4.3.1 The Lanczos and Arnoldi Iterations

The Arnoldi iteration [9] is a way of generating an orthonormal basis for Krylov subspaces $\mathcal{K}_r(A, \boldsymbol{x}_0) = \text{span}\{\boldsymbol{x}_0, A\boldsymbol{x}_0, \ldots, A^r\boldsymbol{x}_0\}$. While it is normally understood as part of a recipe for computing estimates of eigenvalues and eigenvectors, it is a cornerstone of many algorithms for solving large systems of equations.

The basis of the idea of the Arnoldi iteration is fairly simple: given a previously constructed orthonormal basis $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ for $\mathcal{K}_k(A, \boldsymbol{v}_1)$, we use the Gram–Schmidt process (2.2.11) to compute a new element $\boldsymbol{v}_{k+1}$ of an orthonormal basis by orthogonalizing $A\boldsymbol{v}_k$ against $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$ and normalizing:

$$\begin{aligned}\boldsymbol{w}_{k+1} &= A\boldsymbol{v}_k - ((A\boldsymbol{v}_k)^T\boldsymbol{v}_1)\boldsymbol{v}_1 - \cdots - ((A\boldsymbol{v}_k)^T\boldsymbol{v}_k)\boldsymbol{v}_k\\ \boldsymbol{v}_{k+1} &= \boldsymbol{w}_{k+1}/\left\|\boldsymbol{w}_{k+1}\right\|_2.\end{aligned}$$

As noted in Section 2.2.2.2, the original Gram–Schmidt process has numerical instabilities that can be avoided by using the modified Gram–Schmidt process. Setting $h_{\ell k} = (A^T\boldsymbol{v}_k)^T\boldsymbol{v}_\ell$ for $\ell \le k$ and $h_{k+1,k} = \|\boldsymbol{w}_{k+1}\|_2$ we see that

$$A\boldsymbol{v}_k = \sum_{\ell=1}^{k+1} h_{\ell k}\boldsymbol{v}_\ell.$$

Setting $h_{\ell k} = 0$ for $\ell > k+1$. Following the Gram–Schmidt reconstruction (2.2.12), we can write

$$\begin{aligned}A\left[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\right] &= \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k, \boldsymbol{v}_{k+1}\right]\begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,k}\\ h_{21} & h_{22} & \cdots & h_{2,k}\\ 0 & h_{32} & \ddots & \vdots\\ 0 & 0 & \ddots & h_{k,k}\\ 0 & 0 & \cdots & h_{k+1,k}\end{bmatrix}\\ &= \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\right]\begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,k}\\ h_{21} & h_{22} & \cdots & h_{2,k}\\ 0 & \ddots & \ddots & \vdots\\ 0 & \cdots & h_{k-1,k} & h_{k,k}\end{bmatrix} + h_{k+1,k}\boldsymbol{v}_{k+1}\boldsymbol{e}_k^T.\end{aligned}$$

If we put

$$V_k = \left[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\right] \text{ and } H_k = \begin{bmatrix} h_{11} & h_{12} & \cdots & h_{1,k}\\ h_{21} & h_{22} & \cdots & h_{2,k}\\ 0 & \ddots & \ddots & \vdots\\ 0 & \cdots & h_{k-1,k} & h_{k,k}\end{bmatrix}, \quad \text{then}$$

$$A\,V_k = V_k H_k + h_{k+1,k}\boldsymbol{v}_{k+1}\boldsymbol{e}_k^T.$$

**Algorithm 29** Arnoldi iteration

```
1   function arnoldi(A, v_1, m)
2     v_1 ← v_1 / ||v_1||_2
3     for k = 1, 2, ..., m
4       w_k ← A v_k
5       for j = 1, 2, ..., k
6         h_jk ← w_k^T v_j
7         w_k ← w_k − h_jk v_j
8       end for
9       h_{k+1,k} ← ||w_k||_2
10      v_{k+1} ← w_k / h_{k+1,k}
11    end for
12    V_m = [v_1, v_2, ..., v_m];  H_m = [ h_11  h_12  ...  h_1,m
                                           h_21  h_22  ...  h_2,m
                                            0     .     .     .
                                            0    ...  h_m-1,m  h_m,m ]
13    return (V_m, H_m, h_{m+1,m}, v_{m+1})
14  end function
```

The matrix $H_k$ is zero below the first sub-diagonal ($h_{ij} = 0$ if $i > j + 1$); matrices with this structure are called *Hessenberg matrices* (see Section 2.5.3.4). Since $V_k$ is a matrix of orthonormal columns $V_k^T V_k = I$, we have

$$\begin{aligned} V_k^T A\, V_k &= V_k^T (V_k H_k + h_{k+1,k} \boldsymbol{v}_{k+1} \boldsymbol{e}_k^T) \\ &= V_k^T V_k H_k + h_{k+1,k} V_k^T \boldsymbol{v}_{k+1} \boldsymbol{e}_k^T = H_k \end{aligned}$$

as $V_k^T \boldsymbol{v}_{k+1} = \boldsymbol{0}$. The Arnoldi iteration is shown in Algorithm 29.

If the matrix $A$ is symmetric, then there are very important computational advantages. Since $H_k = V_k^T A V_k$, if $A$ is symmetric then $H_k^T = (V_k^T A V_k)^T = V_k^T A^T V_k = V_k^T A V_k = H_k$ is also symmetric. As $H_k$ is symmetric Hessenberg, $H_k$ is symmetric tridiagonal. This means (at least in exact arithmetic) that the Gram–Schmidt process stops after orthogonalizing $\boldsymbol{w}_k$ against $\boldsymbol{v}_k$ and $\boldsymbol{v}_{k-1}$. Using these observations gives the *Lanczos iteration* [152], as shown in Algorithm 30.

As with the Arnoldi iteration, the Lanczos iteration has $A\, V_m = V_m T_m + \beta_m \boldsymbol{v}_m \boldsymbol{e}_m^T$.

Both the Arnoldi and Lanczos iterations can break down if we get $h_{k+1,k} = 0$ or $\beta_k = 0$. In either of these cases, $\boldsymbol{w}_k = \boldsymbol{0}$ in line 9 of Algorithm 29 or line 7 of Algorithm 30 after the Gram–Schmidt process. This means that $A\boldsymbol{v}_k \in \text{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\} = \text{span}\left\{\boldsymbol{v}_1, A\boldsymbol{v}_1, \ldots, A^{k-1}\boldsymbol{v}_1\right\}$. This means that $\text{range}(V_k) = \text{span}\left\{\boldsymbol{v}_1, A\boldsymbol{v}_1, \ldots, A^{k-1}\boldsymbol{v}_1\right\}$ is an *invariant subspace*: $A\,\text{range}(V_k) \subseteq \text{range}(V_k)$. This is actually good for both eigenvalue problems and solving linear systems: $\text{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ must then contain an exact eigenvector. If $\boldsymbol{v}_1 = \boldsymbol{b}$ for solving $A\boldsymbol{x} = \boldsymbol{b}$, exact breakdown means that $\text{span}\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k\}$ must contain an exact solution of $A\boldsymbol{x} = \boldsymbol{b}$.

The bigger problem for these methods is when $h_{k+1,k} \approx 0$ or $\beta_k \approx 0$. Then round-off errors being amplified in the steps $\boldsymbol{v}_{k+1} \leftarrow \boldsymbol{w}_k / h_{k+1,k}$ and $\boldsymbol{v}_k \leftarrow \boldsymbol{w}_k / \beta_k$. This

**Algorithm 30** Lanczos iteration

```
1   function lanczos(A, v_1, m)
2     v_1 ← v_1/‖v_1‖_2;  β_0 ← 0;  v_0 ← 0
3     for k = 1, 2, ..., m
4       w_k ← A v_k
5       α_k ← w_k^T v_k
6       w_k ← w_k − α_k v_k − β_{k−1} v_{k−1}
7       β_k ← ‖w_k‖_2
8       v_{k+1} ← w_k/β_k
9     end for
10    V_m = [v_1, v_2, ..., v_m];  T_m = [ α_1  β_1                    ]
                                         [ β_1  α_2   ⋱               ]
                                         [  0   ⋱     ⋱    β_{m−1}    ]
                                         [  0         β_{m−1}  α_m    ]
11    return (V_m, T_m, β_m, v_{m+1})
12  end function
```

results in gradual loss of orthogonality of the vectors $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$. This can be repaired by re-applying the Gram–Schmidt process to $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, or by using a sequential version of the Householder QR factorization (see Section 2.2.2.4).

There is one more iteration that we should mention: *Lanczos biorthogonalization*. This does *not* create orthonormal bases or even orthogonal bases. However, it does create two sets of vectors $\{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_m\}$ and $\{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_m\}$, where

$$\boldsymbol{v}_k^T \boldsymbol{w}_\ell = \begin{cases} 1, & \text{if } k = \ell, \\ 0, & \text{if } k \neq \ell. \end{cases} \tag{2.4.16}$$

The method of Lanczos biorthogonalization is equivalent to the Lanczos iteration (Algorithm 30) if $A$ is symmetric and $\boldsymbol{v}_1 = \boldsymbol{w}_1$, in which case $\boldsymbol{v}_j = \boldsymbol{w}_j$ for $j = 1, 2, \ldots$. Another aspect that it has in common with the Lanczos iteration is that it involves short sets of operations, unlike the Arnoldi iteration (Algorithm 29). As a result, Lanczos biorthogonalization only requires $\mathcal{O}(n)$ memory. Lanczos biorthogonalization is shown in Algorithm 31.

Lanczos biorthogonalization is the basis of the QMR method (see Section 2.4.3.4). The choice of $\gamma_j$ on line 7 is somewhat arbitrary. The important point is that $\beta_j \gamma_j = \widetilde{\boldsymbol{v}}_{j+1}^T \widetilde{\boldsymbol{w}}_{j+1}$, so as to ensure the biorthogonality property (2.4.16). Note that the $\alpha_j$'s, $\beta_j$'s and $\gamma_j$'s form a non-symmetric tridiagonal matrix

$$T_m = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \gamma_1 & \alpha_2 & \beta_2 & & \\ & \gamma_2 & \alpha_3 & \ddots & \\ & & \ddots & \ddots & \beta_{m-1} \\ & & & \gamma_{m-1} & \alpha_m \end{bmatrix}.$$

**Algorithm 31** Lanczos biorthogonalization

```
1   function lanczosbiorthog(A, A^T, v_1, w_1, m)
2     v_0 ← 0;  w_0 ← 0;  β_0 ← 0;  γ_0 ← 0
3     for j = 1, 2, ..., m
4       α_j ← w_j^T A v_j
5       ṽ_{j+1} ← A v_j − α_j v_j − β_j v_{j−1}
6       w̃_{j+1} ← A^T w_j − α_j w_j − γ_j w_{j−1}
7       γ_j ← |ṽ_{j+1}^T w̃_{j+1}|^{1/2}
8       if γ_{j+1} = 0: return end if // failure
9       β_j ← (ṽ_{j+1}^T w̃_{j+1})/γ_j
10      v_{j+1} ← ṽ_{j+1}/γ_j
11      w_{j+1} ← w̃_{j+1}/β_j
12    end for
13    return (α_1, ..., α_m), (β_1, ..., β_{m−1}), (γ_1, ..., γ_{m−1}),
             (v_1, ..., v_m), (w_1, ..., w_m)
14  end function
```

The main properties for Algorithm 31 are

$$\begin{aligned} A\,V_m &= V_m T_m + \gamma_m \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T, \\ A^T W_m &= W_m T_m^T + \beta_m \boldsymbol{w}_{m+1} \boldsymbol{e}_m^T, \\ W_m^T A V_m &= T_m. \end{aligned}$$

Lanczos biorthogonalization can break down with $\widetilde{\boldsymbol{v}}_j^T \widetilde{\boldsymbol{w}}_j = 0$. If this happens with either $\widetilde{\boldsymbol{v}}_j = \boldsymbol{0}$ or $\widetilde{\boldsymbol{w}}_j = \boldsymbol{0}$, then we can happily terminate the algorithm as we have at least one invariant subspace. However, if $\widetilde{\boldsymbol{v}}_j^T \widetilde{\boldsymbol{w}}_j = 0$ and both $\widetilde{\boldsymbol{v}}_j \neq \boldsymbol{0}$ and $\widetilde{\boldsymbol{w}}_j \neq \boldsymbol{0}$ we have a serious breakdown of the method. There are ways of repairing this problem using what are known as *look-ahead Lanczos* algorithms [198].

### 2.4.3.2 GMRES

The Generalized Minimum RESidual (GMRES) method [224] is based on the Arnoldi iteration. The basic idea is to minimize $\|A\boldsymbol{v} - \boldsymbol{b}\|_2$ over all $\boldsymbol{v} \in \mathcal{K}_m(A, \boldsymbol{b})$, the Krylov subspace generated by $\boldsymbol{b}$. We can write $\boldsymbol{v} = V_m \boldsymbol{y}$ for some $\boldsymbol{y} \in \mathbb{R}^m$. Since $AV_m = V_m H_m + h_{m+1,m} \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T$,

$$\begin{aligned} A\boldsymbol{v} = A V_m \boldsymbol{y} &= V_m H_m \boldsymbol{y} + h_{m+1,m} \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T \boldsymbol{y} \\ &= [V_m,\ \boldsymbol{v}_{m+1}] \begin{bmatrix} H_m \\ h_{m+1,m} \boldsymbol{e}_m^T \end{bmatrix} \boldsymbol{y} = V_{m+1} \widetilde{H}_m \boldsymbol{y} \quad \text{where} \\ \widetilde{H}_m &= \begin{bmatrix} H_m \\ h_{m+1,m} \boldsymbol{e}_m^T \end{bmatrix}. \end{aligned}$$

Note that $\boldsymbol{v}_1$ (the first column of $V_m$) is $\boldsymbol{b}/\|\boldsymbol{b}\|_2$, so $\boldsymbol{b} = V_m(\|\boldsymbol{b}\|_2\,\boldsymbol{e}_1)$. Since $[V_m,\ \boldsymbol{v}_{m+1}]$ has orthonormal columns, $\|[V_m,\ \boldsymbol{v}_{m+1}]\boldsymbol{z}\|_2 = \|\boldsymbol{z}\|_2$. We therefore wish to minimize

$$\left\|\widetilde{H}_m\boldsymbol{y} - \|\boldsymbol{b}\|_2\,\boldsymbol{e}_1\right\|_2 \qquad \text{over } \boldsymbol{y} \in \mathbb{R}^m.$$

This can be accomplished using the QR factorization (see Section 2.2.2). In fact, the QR factorization can be computed especially efficiently because $H_m$ is a Hessenberg matrix by using Givens' rotations (see Section 2.2.2.5).

For the error analysis of the GMRES algorithm, we note that $\boldsymbol{v} = V_m\boldsymbol{y}$ minimizes $\|A\boldsymbol{v} - \boldsymbol{b}\|_2$ over $\mathcal{K}_m(A, \boldsymbol{b})$ and so we can apply the perturbation theorem for linear systems (Theorem 2.1) to bound the error in the solution in terms of the condition number $\kappa_2(A)$. This condition number can be estimated by the least squares condition number $\kappa_2(\widetilde{H}_m)$ (see (2.2.5)).

### 2.4.3.3 Least Squares Based Methods

Since conjugate gradients can be applied to symmetric positive-definite linear systems, they can be applied to the normal equations (2.2.3) $A^TA\boldsymbol{x} = A^T\boldsymbol{b}$ for least squares problems $\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2$. However, the rate of convergence is controlled by $\kappa_2(A^TA) = \kappa_2(A)^2$, which means that the number of iterations is $\mathcal{O}(\kappa_2(A)\ \log(1/\epsilon))$ for a relative error of $\epsilon$, instead of $\mathcal{O}(\kappa_2(A)^{1/2}\log(1/\epsilon))$ as we would get if $A$ were symmetric positive definite. To avoid this squaring of the condition number, one approach is to apply the Lanczos iteration (Algorithm 30) to the symmetric but indefinite matrix

$$B = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \quad \text{with starting vector} \quad \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix}.$$

Assuming that $\|\boldsymbol{u}_1\|_2 = 1$, this gives a sequence of vectors forming an orthonormal basis of the form

$$\boldsymbol{v}_1 = \begin{bmatrix} \boldsymbol{u}_1 \\ \boldsymbol{0} \end{bmatrix},\ \boldsymbol{v}_2 = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{w}_1 \end{bmatrix},\ \boldsymbol{v}_3 = \begin{bmatrix} \boldsymbol{u}_2 \\ \boldsymbol{0} \end{bmatrix},\ \boldsymbol{v}_4 = \begin{bmatrix} \boldsymbol{0} \\ \boldsymbol{w}_2 \end{bmatrix},\ \ldots.$$

This means that in the Lanczos iteration $\alpha_j = \boldsymbol{v}_j^T B\boldsymbol{v}_j = 0$ for all $j$, and that $\beta_{2j-1} = \boldsymbol{v}_{2j}^T B\boldsymbol{v}_{2j-1} = \boldsymbol{w}_j^T A^T\boldsymbol{u}_j = \boldsymbol{u}_j^T A\boldsymbol{w}_j$ while $\beta_{2j} = \boldsymbol{v}_{2j+1}B\boldsymbol{v}_{2j} = \boldsymbol{u}_{j+1}^T A\boldsymbol{w}_j$. Let $\gamma_j = \beta_{2j}$ and $\delta_j = \beta_{2j-1}$. Then the tridiagonal matrix

$$T_{2m} = \begin{bmatrix} 0 & \delta_1 & & & & \\ \delta_1 & 0 & \gamma_1 & & & \\ & \gamma_1 & 0 & \delta_2 & & \\ & & \delta_2 & 0 & \ddots & \\ & & & \ddots & \ddots & \gamma_{m-1} \\ & & & & \gamma_{m-1} & 0 \end{bmatrix}.$$

Permuting the rows and columns to put the odd numbered rows and columns before the even numbered rows and columns, represented by permutation matrix $P_m$, gives

$$P_m^T T_{2m} P_m = \begin{bmatrix} & B_m \\ B_m^T & \end{bmatrix} \quad \text{where } B_m = \begin{bmatrix} \delta_1 & & & & \\ \gamma_1 & \delta_2 & & & \\ & \ddots & \ddots & & \\ & & \gamma_{m-2} & \delta_{m-1} & \\ & & & \gamma_{m-1} & \end{bmatrix}. \tag{2.4.17}$$

This algorithm is called *Lanczos bidiagonalization*. If $V_{2m} = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{2m}]$ then

$$V_{2m} P_m = \begin{bmatrix} U_m & \\ & W_m \end{bmatrix},$$

where $U_m = [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m]$ and $W_m = [\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m]$ which are matrices with orthonormal columns. Using the relationship

$$\begin{aligned} B_m V_{2m} &= V_{2m} T_{2m} + \delta_m \boldsymbol{v}_{2m} \boldsymbol{e}_{2m}^T \qquad \text{we get} \\ A^T U_m &= B_m^T W_m \qquad \text{and} \\ A\, W_m &= B_m U_m + \delta_m \boldsymbol{u}_{m+1} \boldsymbol{e}_m^T. \end{aligned}$$

Note that $B_m$ is $m \times (m-1)$. The *LSQR* algorithm of Paige and Saunders [195] uses Lanczos bidiagonalization to obtain the least squares problem $\min_{\boldsymbol{y}} \|B_m \boldsymbol{y} - \|\boldsymbol{b}\|_2 \boldsymbol{e}_1\|_2$, which is solved by means of a QR factorization using Givens' rotations and gives the approximate solution of the least squares problem $\|A\boldsymbol{x} - \boldsymbol{b}\|_2$ as $\boldsymbol{x} = W_m \boldsymbol{y}$.

#### 2.4.3.4 Quasi-Minimal Residual Method

The *Quasi-Minimal Residual (QMR)* method [98] is based on Lanczos biorthogonalization (Algorithm 31) [225, Alg. 7.4, pp. 212–214]. The tridiagonal matrix $T_m$ produced by the method is used to create a least squares problem

$$\min_{\boldsymbol{y}} \left\| \begin{bmatrix} T_m \\ \gamma_m \boldsymbol{e}_m^T \end{bmatrix} \boldsymbol{y} - \|\boldsymbol{b}\|_2 \boldsymbol{e}_1 \right\|_2,$$

and then the solution is $\boldsymbol{x} = V_m \boldsymbol{y}$. The main trick is to build up $\boldsymbol{y}$ and $\boldsymbol{x}$ as the Lanczos biorthogonalization proceeds, so that it is not necessary to store the $\boldsymbol{v}_j$ and $\boldsymbol{w}_j$ vectors. The implementation uses a QR factorization of the $\begin{bmatrix} T_m \\ \gamma_m \boldsymbol{e}_m^T \end{bmatrix}$ matrix using Givens' rotations. If the $j$th Givens rotation uses the $(c_j = \cos\theta_j, s_j = \sin\theta_j)$ pair

to represent the $2 \times 2$ rotation matrix, Saad [225] shows that the computed solution $\boldsymbol{x}_m$ has residual bounded by

$$\|A\boldsymbol{x}_m - \boldsymbol{b}\|_2 \leq \|V_m\|_2 \left( \prod_{j=1}^{m} |s_j| \right) \|\boldsymbol{b}\|_2 .$$

The value of $\|V_m\|_2$ can be estimated during computation by using $\|V_m\|_2 \leq \|V_m\|_F$ where $\|\cdot\|_F$ is the Frobenius norm.

Other non-symmetric Krylov and related methods have been developed. Of note are the *Conjugate Gradient Squared (CGS)* method of Sonneveld [236] and a *transpose-free QMR* method [97].

As always with these Krylov subspace-type solvers, creating good preconditioners is enormously beneficial. And almost always, finding them is a problem-dependent task.

**Exercises.**

(1) The standard discretization of the Laplacian operator on a region $\Omega \subset \mathbb{R}^2$ on a grid of points $(x_i, y_j) \in \Omega$ with $x_i = x_0 + i\,h$ and $y_j = y_0 + j\,h$ is given by

$$-\nabla^2 u(x_i, y_j) \approx \frac{4u_{ij} - u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1}}{h^2},$$

where $u_{i,j} \approx u(x_i, y_j)$. To be specific, we consider the discretization of the problem where $\Omega = \left\{ (x, y) \mid x^2 + y^2 < 1 \right\}$ with boundary conditions $u(x, y) = g(x, y)$ on $\partial\Omega$. We deal with the boundary conditions by setting $u_{ij} = g(x_i, y_j)$ if $(x_i, y_j) \notin \Omega$.

Let $A_h \boldsymbol{u}_h = \boldsymbol{f}_h$ be the linear system representing the discretization of the Poisson equation

$$\begin{aligned} -\nabla^2 u &= f(x, y) \qquad \text{in } \Omega, \\ u(x, y) &= 0 \qquad \text{for } (x, y) \in \partial\Omega, \end{aligned}$$

where $\boldsymbol{u}_h$ is the vector consisting of $u_{ij}$ for $(x_i, y_j) \in \Omega$.

Use conjugate gradients to solve $A_h \boldsymbol{u}_h = \boldsymbol{f}_h$ for $h = 1/N$ with $N = 5,\ 10,\ 20,\ 40,\ 100$. Use the stopping criterion $\left\| A_h \boldsymbol{u}_h - \boldsymbol{f}_h \right\|_2 / \left\| \boldsymbol{f}_h \right\|_2 < \epsilon$ for a suitable value of $\epsilon$. To be specific, put $\epsilon = 10^{-5}$. Report how the number of iterations needed to achieve this stopping criterion changes with $N$.

(2) Solve the equations in Exercise 1 using GMRES for $N = 10,\ 100$. Compare the rate of convergence with the conjugate gradient method. Since convergence tends to be exponential in the number of iterations, it can be useful to plot the

scaled residual norm $\|A_h \boldsymbol{u}_{h,n} - \boldsymbol{f}_h\|_2 / \|\boldsymbol{f}_h\|_2$ against the iteration count $n$ where $\boldsymbol{u}_{h,n}$ is the $n$th candidate solution from the method.

(3) Solve the equations in Exercise 1 using the Jacobi and Gauss–Seidel iterations for $N = 10,\ 100$. Compare the rate of convergence with the conjugate gradient method.

(4) The *Symmetrized SOR* or SSOR iteration is a method for solving $A\boldsymbol{x} = \boldsymbol{b}$ for a symmetric positive-definite matrix $A$. Write $A = D - L - U$ where $D$ is the diagonal part of $A$ and $L = U^T$ is strictly lower triangular. SSOR is a two-step method based on the SOR iteration (2.4.4), first going forward through the matrix, then backward through the matrix:

$$\boldsymbol{x}_{2k+1} = (D + \omega L)^{-1}(\omega \boldsymbol{b} - [\omega U + (\omega - 1)D]\boldsymbol{x}_k), \tag{2.4.18}$$

$$\boldsymbol{x}_{2k+2} = (D + \omega U)^{-1}(\omega \boldsymbol{b} - [\omega L + (\omega - 1)D]\boldsymbol{x}_k). \tag{2.4.19}$$

Show that the iteration matrix $B_{\text{SSOR}}$ where

$$\boldsymbol{x}_{2k+2} = B_{\text{SSOR}}\boldsymbol{x}_{2k} + \boldsymbol{c} \quad \text{is}$$

$$B_{\text{SSOR}} = \omega(2 - \omega)(D + \omega U)^{-1} D (D + \omega L)^{-1}. \tag{2.4.20}$$

Show that $B_{\text{SSOR}}$ is symmetric and positive definite provided $0 < \omega < 2$.

(5) Implement the SSOR iteration matrix (2.4.20) as a function

```
function ssoritermx(A, ω, x)
   ...
end function
```

Use this with a suitable value of $\omega$ as a preconditioner for the conjugate gradient method applied to the problem of Exercise 1.

(6) The Gauss–Seidel, SOR, and SSOR iterations for solving $A\boldsymbol{x} = \boldsymbol{b}$ are all essentially sequential if implemented in the obvious way, because solving $(D + \omega L)\boldsymbol{z} = \boldsymbol{y}$ for $\boldsymbol{z}$ using forward substitution is inherently sequential. An approach to speed things up comes from graph theory. The idea is to color the vertices of the graph of $A$ so that any pair of adjacent vertices have different colors. Show that if $G = (V, E)$, the graph of $A$ $(n \times n)$ has a coloring $c\colon V \to \{1, 2, \ldots, m\}$ like this ($i \sim j$ implies $c(i) \neq c(j)$), then solving $(D + \omega L)\boldsymbol{z} = \boldsymbol{y}$ for $\boldsymbol{z}$ can be performed in $m$ parallel steps.

(7) Show that the graph of $A$ in Exercise 1, being a grid graph, can be colored by two colors so that no two adjacent vertices have the same color.

(8) Polynomial preconditioners for a matrix $A$ $(n \times n)$ have the form $B = p(A)$ where $p$ is a suitably chosen polynomial. These can be desirable over conjugate gradients for parallel computation since they do not require the computation of inner products $\boldsymbol{u}^T\boldsymbol{v}$, which requires global communication. Show that the spectral radius of $I - BA$ is $\rho(I - BA) = \max\{\,|1 - \lambda\, p(\lambda)| \mid \lambda \text{ is an eigenvalue of } A\,\}$. If $A$ is also symmetric, show that $\|I - BA\|_2 = \rho(I - BA)$.

(9) If the eigenvalues $\lambda$ of $A$ are all real and lie in the interval $[\alpha, \beta]$ with $0 < \alpha$, then we can use

$$q(\lambda) = \frac{T_{m+1}(2\frac{\lambda - \alpha}{\beta - \alpha} - 1)}{T_{m+1}(2\frac{0 - \alpha}{\beta - \alpha} - 1)},$$
$$p(\lambda) = \frac{1 - q(\lambda)}{\lambda},$$

where $T_{m+1}(s)$ is the Chebyshev polynomial of degree $m + 1$ (see (4.6.3) of Section 4.6.2). Check that $p(\lambda)$ is actually a polynomial. Obtain a bound on the spectral radius of $I - BA$ in terms of $\beta/\alpha$. This bound must be strictly less than one to be useful! Note that if $A$ is also symmetric, then $\beta/\alpha \geq \kappa_2(A)$.

(10) The LSQR algorithm [195] is an iterative method for solving least squares problems. Implement it if you do not have an implementation in your favorite language. For $m = 1000$ and $n = 100$, create an $m \times n$ matrix $A$ with entries sampled from a standard normal distribution. Also create two $\boldsymbol{b} \in \mathbb{R}^m$ vectors: (1) $\boldsymbol{b}_1$ has entries samples from a standard normal distribution; (2) for $\boldsymbol{b}_2$ first create $\widehat{\boldsymbol{x}} \in \mathbb{R}^n$ with entries sampled from a standard normal distribution and set $\boldsymbol{b}_2 = A\widehat{\boldsymbol{x}}$. For each $\boldsymbol{b}$ vector plot $\|A\boldsymbol{x}_k - \boldsymbol{b}\|_2 / \|\boldsymbol{b}\|_2$ and $\left\|A^T(A\boldsymbol{x}_k - \boldsymbol{b})\right\|_2 / (\|A\|_2 \|\boldsymbol{b}\|_2)$ against iteration count $k$ for the LSQR algorithm. Note that for $\boldsymbol{b}_1$ with very high probability $\|A\boldsymbol{x}^* - \boldsymbol{b}_1\|_2 / \|\boldsymbol{b}_1\|_2 \approx 1$, while for $\boldsymbol{b}_2$, $\|A\boldsymbol{x}^* - \boldsymbol{b}_2\|_2 = 0$.

## 2.5 Eigenvalues and Eigenvectors

The eigenvalue/eigenvector problem is: Given a square matrix $A$, find $\lambda$ and $\boldsymbol{v}$ where

$$A\boldsymbol{v} = \lambda\boldsymbol{v}, \qquad \boldsymbol{v} \neq \boldsymbol{0}.$$

Computing eigenvalues and eigenvectors forms the third part of numerical linear algebra. This is also the foundation for methods for computing the *Singular Value Decomposition* (SVD). Eigenvalues and eigenvectors, in general, are valuable for understanding, for example, dynamical systems and stability, while computing eigenvalues of symmetric matrices is important in optimization and statistics.

### *2.5.1 The Power Method & Google*

The *power method* is the simplest method for computing eigenvalues and eigenvectors. It is shown in Algorithm 32.

**Algorithm 32** Power method

```
function eigenpower(A, x_0, ε)
  x_0 ← x_0/‖x_0‖; k ← 0; λ̂_0 ← 0
  while ‖Ax_k − λ̂_k x_k‖ > ε‖x_k‖
    y_k ← A x_k
    λ̂_{k+1} ← eigest(x_k, y_k)
    x_{k+1} ← y_k/‖y_k‖
    k ← k + 1
  end while
  return (x_k, λ̂_k)
end function
```

The function `eigest` used in Algorithm 32 can be chosen in different ways. One of the simplest is to simply use the ratio $(\boldsymbol{y}_k)_j/(\boldsymbol{x}_k)_j$; if $A\boldsymbol{x}_k = \lambda\boldsymbol{x}_k$ then $(\boldsymbol{y}_k)_j/(\boldsymbol{x}_k)_j = \lambda$ for any choice of $j$. To avoid amplification of errors, we use the largest denominator:

$$eigest(\boldsymbol{x}, \boldsymbol{y}) = \frac{y_j}{x_j} \quad \text{where} \quad |x_j| = \max_{\ell} |x_\ell| \,. \tag{2.5.1}$$

An alternative that is most useful for real symmetric or complex Hermitian matrices is

$$eigest(\boldsymbol{x}, \boldsymbol{y}) = \frac{\overline{\boldsymbol{x}}^T \boldsymbol{y}}{\overline{\boldsymbol{x}}^T \boldsymbol{x}}. \tag{2.5.2}$$

**Example 2.21** As an example we will use

$$A = \begin{bmatrix} +1 & +2 & 0 & 0 & -1 \\ -3 & +1 & -1 & 0 & +1 \\ -1 & -1 & +2 & +1 & +1 \\ -3 & +3 & +2 & +1 & -1 \\ -2 & -3 & -2 & +2 & -2 \end{bmatrix}, \qquad \boldsymbol{x}_0 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ 1 \\ 1 \end{bmatrix}. \tag{2.5.3}$$

We measure the error by the ratio $\left\| A\boldsymbol{x} - \widehat{\lambda}\boldsymbol{x} \right\|_2 / \|\boldsymbol{x}\|_2$. Note that if $\boldsymbol{z} = A\boldsymbol{x} - \widehat{\lambda}\boldsymbol{x}$, then $A - \boldsymbol{z}\overline{\boldsymbol{x}}^T / \|\boldsymbol{x}\|_2^2$ has $\widehat{\lambda}$ as an exact eigenvalue and $\boldsymbol{x}$ as an exact eigenvector. The perturbation $E = \boldsymbol{z}\overline{\boldsymbol{x}}^T / \|\boldsymbol{x}\|_2^2$ has 2-norm $\|E\|_2 = \|z\|_2 / \|\boldsymbol{x}\|_2$.

After 50 iterations, the eigenvalue estimate was $\approx 3.2654125$ which differed from the MATLAB-computed maximum eigenvalue by $\approx 3.48 \times 10^{-5}$. The progress of the convergence is shown in Figure 2.5.1. The dashed line in Figure 2.5.1 is given by $4 \times 0.805^k$.

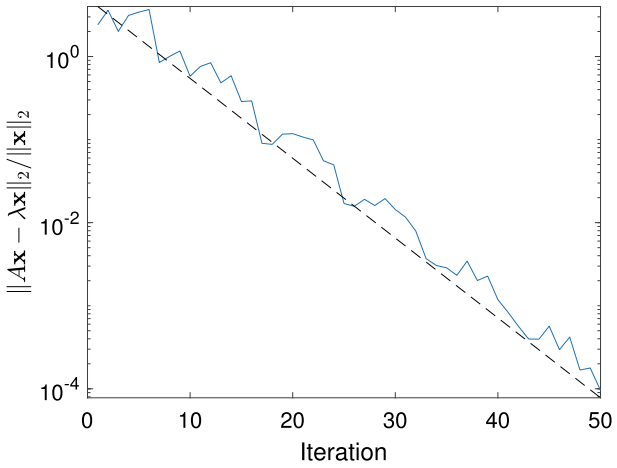



**Fig. 2.5.1** Convergence of an example of the power method

### 2.5.1.1 Google PageRank

An example of the practical use of the power method is the original *Google PageRank* algorithm to rank web pages matching a given search criterion. This algorithm is based on a *Markov chain* (see Section 7.4.2.1). The Markov chain behind the PageRank algorithm models an indiscriminate web surfer who randomly picks links to follow from a given web page with equal probability. If there are no links, simply pick any of the very large number of web pages in existence. Web pages with high equilibrium probability of being viewed in this Markov chain are taken as being more important: there must be more links to these web pages, and many of these links are from other web pages that are likely to be important. Thus, the equilibrium probabilities give a basis for ranking web pages without having to somehow determine the quality or meaning of their content. The Google PageRank algorithm modifies this Markov chain by having a probability $\alpha > 0$ of simply picking a web page uniformly from all of the possible web pages.

If $\boldsymbol{p}_t$ is the probability vector with $(\boldsymbol{p}_t)_i$ the probability that web page $i$ is being viewed after $t$ steps, then

$$\boldsymbol{p}_{t+1} = P\,\boldsymbol{p}_t,$$

where $P$ is the matrix of transition probabilities: $P_{ij}$ is the probability that web page $i$ is viewed at time $t+1$ given that web page $j$ is viewed at time $t$. These values are all between zero and one. Furthermore, the total probability must remain one: $\sum_{i=1}^{N}(\boldsymbol{p}_t)_i = 1$, that is, $\boldsymbol{e}^T\boldsymbol{p}_t = 1$ for all $t$, where $\boldsymbol{e} = [1, 1, 1, \ldots, 1]^T$ is the vector of all ones. Then

$$\boldsymbol{e}^T\boldsymbol{p}_{t+1} = \boldsymbol{e}^T P\,\boldsymbol{p}_t = \boldsymbol{e}^T\boldsymbol{p}_t \qquad \text{for all probability vectors } \boldsymbol{p}_t.$$

That is, $\boldsymbol{e}^T P = \boldsymbol{e}^T$.

If $P_0$ is matrix of transition probabilities for picking links described above, the PageRank algorithm modifies this by incorporating a probability $\alpha > 0$ of simply picking a random web page instead of picking from the links in the current page. The matrix of transition probabilities then becomes

$$P = \alpha \boldsymbol{e}\boldsymbol{e}^T/N + (1-\alpha)P_0. \tag{2.5.4}$$

This means that each entry $P_{ij} \geq \alpha/N$.

The set of probability vectors is $\Sigma = \left\{ \boldsymbol{p} \mid \boldsymbol{e}^T \boldsymbol{p} = 1,\ \boldsymbol{p} \geq \boldsymbol{0} \right\}$ where the inequality "$\geq$" is understood componentwise. This set is a convex, bounded set in $\mathbb{R}^N$ where $N$ is the number of web pages. The matrix defines a function $\boldsymbol{p} \mapsto P\,\boldsymbol{p}$ that is $\Sigma \to \Sigma$. Brouwer's fixed-point theorem [95] shows that there is a fixed point $\boldsymbol{p}^*$ where $P\,\boldsymbol{p}^* = \boldsymbol{p}^*$. Then $\boldsymbol{p}^*$ is an eigenvector of $P$ with eigenvalue one. Furthermore, there is only one probability vector with this property since every entry in $P$ is strictly positive:

$$\begin{aligned}
\|P(\boldsymbol{p}-\boldsymbol{q})\|_1 &= \sum_{i=1}^{N} \left| \sum_{j=1}^{N} P_{ij}(p_j - q_j) \right| \\
&\leq \sum_{i=1}^{N} \sum_{j=1}^{N} P_{ij} \left| p_j - q_j \right| = \sum_{j=1}^{N} \sum_{i=1}^{N} P_{ij} \left| p_j - q_j \right| \\
&= \sum_{j=1}^{N} \left| p_j - q_j \right| = \|\boldsymbol{p}-\boldsymbol{q}\|_1 .
\end{aligned}$$

Equality can only occur if $\left| \sum_{j=1}^{N} P_{ij}(p_j - q_j) \right| = \sum_{j=1}^{N} P_{ij} \left| p_j - q_j \right|$ for all $i$. Since all $P_{ij} > 0$, this would mean that $p_j - q_j$ have the same sign for all $j$. Since $\sum_{j=1}^{N} p_j = \sum_{j=1}^{N} q_j = 1$ for probability vectors, this means that $p_j - q_j = 0$ for all $j$. Thus, the equilibrium probability vector $\boldsymbol{p}^*$ is unique.

We can apply the power method

$$\boldsymbol{p}_{t+1} = P\,\boldsymbol{p}_t$$

to this matrix. Since $P$ is the sum of a sparse matrix (the matrix of links between web pages) plus a low-rank matrix $\alpha \boldsymbol{e}\boldsymbol{e}^T/N$ plus the matrix with column $\boldsymbol{e}/N$ for every web page with no out-going links, the matrix–vector product $P\,\boldsymbol{p}$ can be computed efficiently. The number of floating point operations needed is $\mathcal{O}(N+L)$ for each step where $N$ is the number of web pages and $L$ is the number of links rather than $\mathcal{O}(N^2)$.

To see how quickly the method converges for the transition probability matrix (2.5.4),

$$\begin{aligned}\left\|\boldsymbol{p}_{t+1}-\boldsymbol{p}^*\right\|_1 &= \left\|P(\boldsymbol{p}_t-\boldsymbol{p}^*)\right\|_1\\ &= \left\|\frac{\alpha}{N}\boldsymbol{e}\boldsymbol{e}^T(\boldsymbol{p}_t-\boldsymbol{p}^*)+(1-\alpha)P_0(\boldsymbol{p}_t-\boldsymbol{p}^*)\right\|_1\\ &= \left\|\frac{\alpha}{N}\boldsymbol{e}(\boldsymbol{e}^T\boldsymbol{p}_t-\boldsymbol{e}^T\boldsymbol{p}^*)+(1-\alpha)P_0(\boldsymbol{p}_t-\boldsymbol{p}^*)\right\|_1\\ &= \left\|(1-\alpha)P_0(\boldsymbol{p}_t-\boldsymbol{p}^*)\right\|_1 \quad (\text{since } \boldsymbol{e}^T\boldsymbol{p}_t=\boldsymbol{e}^T\boldsymbol{p}^*=1)\\ &\le (1-\alpha)\left\|\boldsymbol{p}_t-\boldsymbol{p}^*\right\|_1.\end{aligned}$$

So the power method in this case converges and converges at rate controlled by $1-\alpha$. The number of iterations needed to give a specified accuracy $\epsilon$ is $\mathcal{O}(\log(\epsilon)/\log(1-\alpha))$. Google reportedly uses $\alpha \approx 0.15$, so that they can guarantee a certain level of accuracy with a modest number of iterations.

#### 2.5.1.2 Convergence of the Power Method

While Google's PageRank method is guaranteed convergence because of its special structure, we need to investigate the convergence of the power method in general.

We first show that in Algorithm 32,

$$\boldsymbol{x}_k = \frac{A^k\boldsymbol{x}_0}{\left\|A^k\boldsymbol{x}_0\right\|}, \tag{2.5.5}$$

where $\|\cdot\|$ is the vector norm used in Algorithm 32. Note that if (2.5.5) holds, then

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \frac{A\boldsymbol{x}_k}{\|A\boldsymbol{x}_k\|} = \frac{A(A^k\boldsymbol{x}_0/\left\|A^k\boldsymbol{x}_0\right\|)}{\left\|A(A^k\boldsymbol{x}_0/\left\|A^k\boldsymbol{x}_0\right\|)\right\|}\\ &= \frac{A^{k+1}\boldsymbol{x}_0/\left\|A^k\boldsymbol{x}_0\right\|}{\left\|A^{k+1}\boldsymbol{x}_0\right\|/\left\|A^k\boldsymbol{x}_0\right\|} = \frac{A^{k+1}\boldsymbol{x}_0}{\left\|A^{k+1}\boldsymbol{x}_0\right\|},\end{aligned}$$

so that (2.5.5) holds for $k$ replaced by $k+1$. Thus, by induction, (2.5.5) holds for $k=0,1,2,\ldots$.

We suppose that our $n\times n$ matrix $A$ has a basis of eigenvectors $\boldsymbol{v}_1,\boldsymbol{v}_2,\ldots,\boldsymbol{v}_n$ with $A\boldsymbol{v}_j=\lambda_j\boldsymbol{v}_j$. We assume that

$$|\lambda_1| > |\lambda_2| \ge |\lambda_3| \ge \cdots \ge |\lambda_n|. \tag{2.5.6}$$

This makes $\lambda_1$ the dominant eigenvalue. Since $\{\boldsymbol{v}_1,\boldsymbol{v}_2,\ldots,\boldsymbol{v}_n\}$ is a basis for $\mathbb{R}^n$ (or $\mathbb{C}^n$ if appropriate), we can write

$$\boldsymbol{x}_0 = c_1\boldsymbol{v}_1+c_2\boldsymbol{v}_2+\cdots+c_n\boldsymbol{v}_n.$$

We further assume that $c_1 \neq 0$. Since $A^k \boldsymbol{v}_j = \lambda_j^k \boldsymbol{v}_j$,

$$A^k \boldsymbol{x}_0 = c_1 \lambda_1^k \boldsymbol{v}_1 + c_2 \lambda_2^k \boldsymbol{v}_2 + \cdots + c_n \lambda_n^k \boldsymbol{v}_n.$$

Then

$$\boldsymbol{x}_k = \frac{A^k \boldsymbol{x}_0}{\left\| A^k \boldsymbol{x}_0 \right\|} = \frac{c_1 \lambda_1^k \boldsymbol{v}_1 + c_2 \lambda_2^k \boldsymbol{v}_2 + \cdots + c_n \lambda_n^k \boldsymbol{v}_n}{\left\| c_1 \lambda_1^k \boldsymbol{v}_1 + c_2 \lambda_2^k \boldsymbol{v}_2 + \cdots + c_n \lambda_n^k \boldsymbol{v}_n \right\|}.$$

Note that

$$\begin{aligned} & c_1 \lambda_1^k \boldsymbol{v}_1 + c_2 \lambda_2^k \boldsymbol{v}_2 + \cdots + c_n \lambda_n^k \boldsymbol{v}_n \\ & \quad = c_1 \lambda_1^k \left[ \boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j \lambda_j^k}{c_1 \lambda_1^k} \boldsymbol{v}_j \right], \qquad \text{so} \\ \boldsymbol{x}_k &= \frac{c_1 \lambda_1^k \left[ \boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j \lambda_j^k}{c_1 \lambda_1^k} \boldsymbol{v}_j \right]}{\left\| c_1 \lambda_1^k \left[ \boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j \lambda_j^k}{c_1 \lambda_1^k} \boldsymbol{v}_j \right] \right\|} \\ & = \frac{c_1 \lambda_1^k}{\left| c_1 \lambda_1^k \right|} \frac{\boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j}{c_1} \left( \frac{\lambda_j}{\lambda_1} \right)^k \boldsymbol{v}_j}{\left\| \boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j}{c_1} \left( \frac{\lambda_j}{\lambda_1} \right)^k \boldsymbol{v}_j \right\|}. \end{aligned}$$

We assumed (2.5.6), and so $\left| \lambda_j / \lambda_1 \right| < 1$ for $j = 2, 3, \ldots, n$. Then $\left| \lambda_j / \lambda_1 \right|^k \to 0$ as $k \to \infty$ for $j \geq 2$. Therefore

$$\frac{\left| c_1 \lambda_1^k \right|}{c_1 \lambda_1^k} \boldsymbol{x}_k = \frac{\boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j}{c_1} \left( \frac{\lambda_j}{\lambda_1} \right)^k \boldsymbol{v}_j}{\left\| \boldsymbol{v}_1 + \sum_{j=2}^{n} \frac{c_j}{c_1} \left( \frac{\lambda_j}{\lambda_1} \right)^k \boldsymbol{v}_j \right\|} \to \frac{\boldsymbol{v}_1}{\left\| \boldsymbol{v}_1 \right\|} \quad \text{as } k \to \infty.$$

Note that this result does *not* claim that $\boldsymbol{x}_k$ converges, just that span $\{\boldsymbol{x}_k\}$ converges to span $\{\boldsymbol{v}_1\}$ in the sense that the angle between these subspaces goes to zero.

A better way to describe the angle $\theta = \angle(\text{span}\,\{\boldsymbol{x}\}, \text{span}\,\{\boldsymbol{y}\})$, the angle between span $\{\boldsymbol{x}\}$ and span $\{\boldsymbol{y}\}$, is to note that

$$\begin{aligned} \sin\theta &= \min_s \frac{\left\| \boldsymbol{x} - s\boldsymbol{y} \right\|_2}{\left\| \boldsymbol{x} \right\|_2} = \left[ 1 - \frac{\left| \overline{\boldsymbol{x}}^T \boldsymbol{y} \right|^2}{\left\| \boldsymbol{x} \right\|_2^2 \left\| \boldsymbol{y} \right\|_2^2} \right]^{1/2} \\ &= \left\| \left[ I - \frac{\boldsymbol{y} \overline{\boldsymbol{y}}^T}{\left\| \boldsymbol{y} \right\|_2^2} \right] \frac{\boldsymbol{x}}{\left\| \boldsymbol{x} \right\|_2} \right\|_2; \end{aligned}$$

the minimizing $s = \overline{\boldsymbol{y}}^T \boldsymbol{x} / \|\boldsymbol{y}\|_2^2$.

We should be concerned with the rate of convergence as well as the fact of convergence. We note that

$$\frac{\boldsymbol{v}_1 + \sum_{j=2}^n \frac{c_j}{c_1} \left(\frac{\lambda_j}{\lambda_1}\right)^k \boldsymbol{v}_j}{\left\| \boldsymbol{v}_1 + \sum_{j=2}^n \frac{c_j}{c_1} \left(\frac{\lambda_j}{\lambda_1}\right)^k \boldsymbol{v}_j \right\|} - \frac{\boldsymbol{v}_1}{\|\boldsymbol{v}_1\|} = \mathcal{O}\left( \left| \frac{\lambda_2}{\lambda_1} \right|^k \right) \qquad \text{as } k \to \infty.$$

This means that if $|\lambda_2| \approx |\lambda_1|$ then convergence can be very slow. For our computational example (2.5.3), $\lambda_2 \approx 0.90230 \pm 2.468507i$ and $|\lambda_2/\lambda_1| \approx 0.80488$, which gives the approximate slope of the residual norm in Figure 2.5.1.

Note that if $|\lambda_2| = |\lambda_1|$, then there is typically no convergence at all. This latter situation is not uncommon: for real matrices, this occurs if $\lambda_1$ is complex, so $\lambda_2 = \overline{\lambda_1}$ and the two largest eigenvalues have the same magnitude. Perhaps this should not be surprising: if $A$ is real and $\boldsymbol{x}_0$ is real, so are all the iterates $\boldsymbol{x}_k$. We cannot expect convergence to a complex eigenvector.

Also note that convergence is to the dominant eigenvalue and its eigenvector. Although we assume $c_1 \neq 0$, this is not a strong assumption. Roundoff error makes it unlikely that we get $c_1 = 0$, or that this is maintained. Even if we begin with $c_1 \approx 0$, this will be amplified.

If we wish to find other eigenvalues, or accelerate convergence, or deal with a complex conjugate pair of dominant eigenvalues, we need other methods.

#### 2.5.1.3 Variants: Inverse Iteration

We can start by giving a target $\mu$ for the eigenvalues: we seek to find the eigenvalue closest to $\mu$ and its eigenvector. While $A - \mu I$ has eigenvalues $\lambda_j - \mu$, if $\lambda_j \approx \mu$ then $\lambda_j - \mu$ is small, and so components in the direction of $\boldsymbol{v}_j$ will be reduced, not amplified. So we use $(A - \mu I)^{-1}$ which has eigenvalues $1/(\lambda_j - \mu)$. Now if $\lambda_j \approx \mu$, $1/(\lambda_j - \mu)$ is large. If $\mu$ is closer to $\lambda_j$ than any other eigenvalue of $A$, then $1/(\lambda_j - \mu)$ is the dominant eigenvalue of $(A - \mu I)^{-1}$. Applying the power method to $(A - \mu I)^{-1}$ should give us convergence of the eigenvalue estimates to $1/(\lambda_j - \mu)$ and the eigenvector to span $\{\boldsymbol{v}_j\}$. If $\widehat{\lambda}$ is the estimate of the eigenvalue of $(A - \mu I)^{-1}$, then the estimate of $\lambda_j$ should be $\mu + 1/\widehat{\lambda}$. This gives us the *inverse power method* in Algorithm 33.

The error then is expected to be $\mathcal{O}((|\lambda_\ell - \mu| / |\lambda_j - \mu|)^k)$ as $k \to \infty$ where $\lambda_\ell$ is the second closest eigenvalue to $\mu$.

For our example (2.5.3), taking $\mu = \frac{1}{2} + 2i$ so as to target $\lambda_2$, the convergence is shown in Figure 2.5.2. The slope of this plot indicates an error $\approx C\, r^k$ with $r \approx 0.3908$, while $|\lambda_\ell - \mu| / |\lambda_j - \mu| \approx 0.3910$.

But this can be accelerated even more. Since we want $\mu$ to be close to the target eigenvalue for fast convergence, we can use the estimated eigenvalue to give an

**Algorithm 33** Inverse (shifted) power method

```
function inveigpower(A, x_0, μ, ε)
  x_0 ← x_0/‖x_0‖;  k ← 0;  λ̂_0 ← 0
  while ‖Ax_k − λ̂_k x_k‖ > ε‖x_k‖
    y_k ← (A − μI)^{-1} x_k
    λ̂_{k+1} ← μ + 1/eigest(x_k, y_k)
    x_{k+1} ← y_k/‖y_k‖
    k ← k + 1
  end while
  return (x_k, λ̂_k)
end function
```

**Algorithm 34** Inverse iteration

```
function inviter(A, x_0, μ_0, ε)
  x_0 ← x_0/‖x_0‖;  k ← 0;  λ̂_0 ← 0
  while ‖Ax_k − μ_k x_k‖ > ε‖x_k‖
    y_k ← (A − μ_k I)^{-1} x_k
    μ_{k+1} ← μ_k + 1/eigest(x_k, y_k)
    x_{k+1} ← y_k/‖y_k‖
    k ← k + 1
  end while
  return (x_k, μ_k)
end function
```

even better approximation to the target eigenvalue. This gives Algorithm 34 which is called *inverse iteration*.

Results for Algorithm 34 are shown for our example (2.5.3) in Figure 2.5.2 using $\mu_0 = \frac{1}{2} + 2i$. As can be seen from Figure 2.5.2, the convergence of inverse iteration

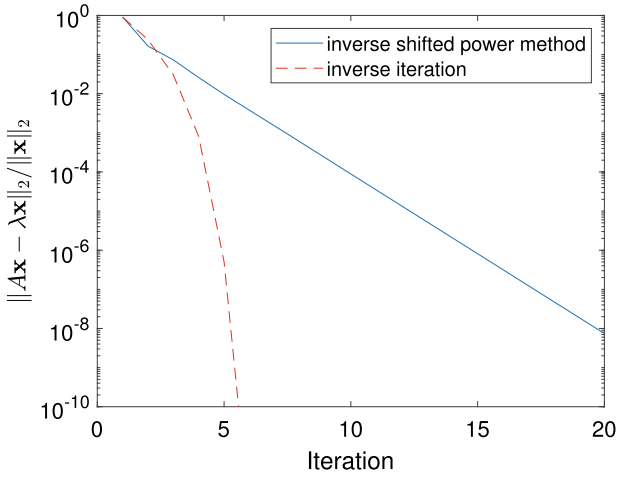



**Fig. 2.5.2** Convergence of inverse (shifted) power method example

is quite rapid. In fact, it can be shown that for inverse iteration where $\mu_k \to \lambda$ as $k \to \infty$ then $|\mu_{k+1} - \lambda| = \mathcal{O}(|\mu_k - \lambda|^2)$ showing quadratic convergence. If $A$ is symmetric and the "symmetric" eigenvalue estimate (2.5.2) is used, inverse iteration has cubic convergence: $|\mu_{k+1} - \lambda| = \mathcal{O}(|\mu_k - \lambda|^3)$ as $k \to \infty$.

### 2.5.2 Schur Decomposition

If a square matrix $A$ does not have a basis of eigenvectors, we often need a way to find a complete eigen-decomposition of $A$. The algebraic answer is the *Jordan canonical form* (JCF):

**Theorem 2.22** *If $A$ is an $n \times n$ real or complex matrix then there is a complex invertible matrix $X$ where*

$$X^{-1} A X = \begin{bmatrix} J(n_1, \lambda_1) & & & & \\ & J(n_2, \lambda_2) & & & \\ & & J(n_3, \lambda_3) & & \\ & & & \ddots & \\ & & & & J(n_r, \lambda_r) \end{bmatrix} \quad \textit{where}$$

$$J(m, \lambda) = \begin{bmatrix} \lambda & 1 & & \\ & \lambda & 1 & \\ & & \lambda & \ddots \\ & & & \ddots & 1 \\ & & & & \lambda \end{bmatrix} \quad (m \times m).$$

We do not give a proof of this theorem. This is a difficult theorem to apply in practice numerically: the matrix $X$ can have arbitrarily bad condition number. Take, for example,

$$A_\eta = \left[\begin{array}{c|c} 1 & 1 \\ \hline 0 & 1+\eta \end{array}\right].$$

This matrix $A_\eta$ has distinct eigenvalues one and $1 + \eta$ for $\eta \neq 0$. Thus, apart from scale factors, the eigenvectors are $\boldsymbol{v}_1 = [1,\ 0]^T$ and $\boldsymbol{v}_2 = [1,\ \eta]^T$. Let $X_\eta = [\boldsymbol{v}_1,\ s\boldsymbol{v}_2]$ for some scale factor $s \neq 0$. Note that the condition number is independent of the overall scaling of a matrix: $\kappa(\alpha X_\eta) = \kappa(X_\eta)$ for any $\alpha \neq 0$. Then

$$X_\eta = \left[\begin{array}{c|c} 1 & s \\ \hline 0 & s\eta \end{array}\right], \qquad X_\eta^{-1} = \left[\begin{array}{c|c} 1 & -1/\eta \\ \hline 0 & 1/(s\eta) \end{array}\right] \qquad \text{so}$$

$$\kappa_\infty(X_\eta) = \|X_\eta\|_\infty \left\|X_\eta^{-1}\right\|_\infty = \max(1 + |s|\,,\ |s\eta|)\ \max(1 + \frac{1}{|\eta|},\ \frac{1}{|s\eta|})$$

$$\geq 1 + \frac{1}{|\eta|}.$$

Indeed, a thorough investigation of the numerical issues in numerically computing the JCF by Demmel in his PhD thesis [76] in 1983 showed that it can be extremely challenging to numerically compute the JCF of a given matrix.

Instead, for numerical computation, a much better alternative is the *Schur decomposition*:

**Theorem 2.23** *For any square real or complex matrix $A$ there is a unitary matrix $U$ ($U^{-1} = \overline{U}^T$), where*

$$\overline{U}^T A\, U = T, \qquad T \text{ is upper triangular.} \tag{2.5.7}$$

Note that the diagonal entries of $T$ are the eigenvalues of $A$.

***Proof*** We prove this by induction on $n$ where $A$ is an $n \times n$ matrix.

**Base case:** $n = 1$. In this case $A$ is $1 \times 1$ so $A = [a_{11}]$ and we can take $U = [1]$ and $T = [a_{11}]$.

**Induction step:** Suppose (2.5.7) holds whenever $A$ is $k \times k$; we now wish to show that (2.5.7) holds whenever $A$ is $(k+1) \times (k+1)$.

We start by showing that $A$ has a (real or complex) eigenvalue. The *characteristic polynomial* $p_A(z) = \det(A - zI)$ is a polynomial with real or complex coefficients, and so by the Fundamental Theorem of Algebra [134, p. 309] there is a real or complex zero of this polynomial: $p_A(\lambda) = 0 = \det(A - \lambda I)$. Then $A - \lambda I$ is an invertible matrix with real or complex coefficients, and so there is a real or complex vector $\boldsymbol{v} \neq \boldsymbol{0}$ where $(A - \lambda I)\boldsymbol{v} = \boldsymbol{0}$. Note that if $A$ is a real matrix and $\lambda$ is also real, then $\boldsymbol{v}$ can be chosen to be a real vector. We can normalize $\boldsymbol{v}$ so that $\|\boldsymbol{v}\|_2^2 = \overline{\boldsymbol{v}}^T \boldsymbol{v} = \sum_{j=1}^{k+1} |v_j|^2 = 1$.

By means of the QR factorization of $\boldsymbol{v}$, there is a unitary matrix $Q = [\boldsymbol{q},\ Q_2]$ where $\boldsymbol{v} = [\boldsymbol{q},\ Q_2] \begin{bmatrix} r \\ \boldsymbol{0} \end{bmatrix} = r\boldsymbol{q}$. Since $\|\boldsymbol{v}\|_2 = \|\boldsymbol{q}\|_2 = 1$, $|r| = 1$ and so $A\boldsymbol{q} = \lambda\boldsymbol{q}$ with $\boldsymbol{q} \neq \boldsymbol{0}$. Then

$$\begin{aligned} \overline{Q}^T A\, Q &= \begin{bmatrix} \overline{\boldsymbol{q}}^T \\ \overline{Q_2}^T \end{bmatrix} A \begin{bmatrix} \boldsymbol{q} & Q_2 \end{bmatrix} = \begin{bmatrix} \overline{\boldsymbol{q}}^T A\boldsymbol{q} & \overline{\boldsymbol{q}}^T A\, Q_2 \\ \overline{Q_2}^T A\boldsymbol{q} & \overline{Q_2}^T A\, Q_2 \end{bmatrix} \\ &= \begin{bmatrix} \overline{\boldsymbol{q}}^T \lambda\boldsymbol{q} & \overline{\boldsymbol{q}}^T A\, Q_2 \\ \overline{Q_2}^T \lambda\boldsymbol{q} & \overline{Q_2}^T A\, Q_2 \end{bmatrix} = \begin{bmatrix} \lambda & \overline{\boldsymbol{q}}^T A\, Q_2 \\ \boldsymbol{0} & \overline{Q_2}^T A\, Q_2 \end{bmatrix} \end{aligned}$$

since $\overline{Q_2}^T \boldsymbol{q} = \boldsymbol{0}$ by orthogonality of the columns of $Q_2$ with $\boldsymbol{q}$. By the induction hypothesis, there is $k \times k$ unitary matrix $\widetilde{U}$ where $\overline{\widetilde{U}}^T \overline{Q_2}^T A\, Q_2\, \widetilde{U} = \widetilde{T}$, an upper triangular matrix. Putting

$$U = Q \begin{bmatrix} 1 & \\ & \widetilde{U} \end{bmatrix} \qquad \text{which is unitary, we get}$$

$$\overline{U}^T A\, U = \begin{bmatrix} 1 & \\ & \overline{\widetilde{U}}^T \end{bmatrix} \overline{Q}^T A\, Q \begin{bmatrix} 1 & \\ & \widetilde{U} \end{bmatrix} = \left[\begin{array}{c|c} \lambda & \overline{\boldsymbol{q}}^T A\, Q_2\, \widetilde{U} \\ \hline \mathbf{0} & \overline{\widetilde{U}}^T \overline{Q_2}^T A\, Q_2\, \widetilde{U} \end{array}\right]$$

$$= \left[\begin{array}{c|c} \lambda & \overline{\boldsymbol{q}}^T A\, Q_2\, \widetilde{U} \\ \hline \mathbf{0} & \widetilde{T} \end{array}\right] = T \qquad \text{which is upper triangular}$$

as we wanted.

Therefore, by the principle of induction, (2.5.7) holds for $A$ is $n \times n$ for $n = 1, 2, 3, \ldots$. □

It should be noted that the Schur decomposition gives a true eigen-decomposition in many cases. First, if $A$ is a Hermitian matrix ($\overline{A}^T = A$) or real symmetric ($A^T = A$ and real) then $U$ unitary and

$$\overline{U}^T A\, U = T \quad \text{upper triangular, implies}$$
$$\overline{T}^T = \overline{\overline{U}^T A\, U}^T = \overline{U}^T \overline{A}^T \overline{\overline{U}^T}^T = \overline{U}^T A\, U = T.$$

Then $T$ must be diagonal with real diagonal entries.

More generally, if $A$ is *normal* in the sense that $\overline{A}^T A = A\, \overline{A}^T$, then

$$\overline{A}^T A = \overline{U\, T\, \overline{U}^T}^T U\, T\, \overline{U}^T = \overline{\overline{U}^T}^T \overline{T}^T \overline{U}^T U\, T\, \overline{U}^T = U\, \overline{T}^T T\, \overline{U}^T \quad \text{while}$$
$$A\, \overline{A}^T = U\, T\, \overline{U}^T \overline{U\, T\, \overline{U}^T}^T = U\, T\, \overline{U}^T U\, \overline{T}^T \overline{U}^T = U\, T\, \overline{T}^T \overline{U}^T.$$

Equating these and re-arranging gives $\overline{T}^T T = T\, \overline{T}^T$. We will see that this implies $T$ is diagonal . Write $T = \left[\begin{array}{c|c} \tau & \boldsymbol{t}^T \\ \hline & \widetilde{T} \end{array}\right]$. Then $\overline{T}^T T = T\, \overline{T}^T$ implies that

$$\left[\begin{array}{c|c} \overline{\tau} & \\ \hline \overline{\boldsymbol{t}} & \overline{\widetilde{T}}^T \end{array}\right] \left[\begin{array}{c|c} \tau & \boldsymbol{t}^T \\ \hline & \widetilde{T} \end{array}\right] = \left[\begin{array}{c|c} |\tau|^2 & \overline{\tau} \boldsymbol{t}^T \\ \hline \tau \overline{\boldsymbol{t}} & \overline{\widetilde{T}}^T \widetilde{T} + \overline{\boldsymbol{t}} \boldsymbol{t}^T \end{array}\right] \quad \text{and}$$

$$\left[\begin{array}{c|c} \tau & \boldsymbol{t}^T \\ \hline & \widetilde{T} \end{array}\right] \left[\begin{array}{c|c} \overline{\tau} & \\ \hline \overline{\boldsymbol{t}} & \overline{\widetilde{T}}^T \end{array}\right] = \left[\begin{array}{c|c} |\tau|^2 + \|\boldsymbol{t}\|_2^2 & \boldsymbol{t}^T \overline{\widetilde{T}}^T \\ \hline \widetilde{T} \overline{\boldsymbol{t}} & \widetilde{T}\, \overline{\widetilde{T}}^T \end{array}\right] \quad \text{are equal.}$$

Then $\|\boldsymbol{t}\|_2^2 = 0$ and so $\boldsymbol{t} = \mathbf{0}$. Applying the argument inductively to $\widetilde{T}$ shows that $T$ must be diagonal in this case, and we have a complete orthogonal eigen-decomposition of the matrix $A$.

### 2.5.2.1 Continuity and Perturbation Theorems for Eigenvalues

The set of eigenvalues, or *spectrum*,

$$\sigma(A) = \{\, \lambda \mid \lambda \text{ is an eigenvalue of } A \,\} \tag{2.5.8}$$

is continuous in the entries of $A$ in the sense that for any $\epsilon > 0$ there is a $\delta = \delta(A) > 0$ where for any $E$ with $\|E\|_\infty < \delta$ implies that every $\lambda \in \sigma(A)$ is within distance $\epsilon$ of some $\mu \in \sigma(A+E)$, and every $\mu \in \sigma(A+E)$ is within distance $\epsilon$ of some $\lambda \in \sigma(A)$.

**Theorem 2.24** *For every $n$, the spectrum $\sigma(A)$ of $n \times n$ matrices is continuous in the sense described above.*

***Proof*** (Outline) This proof requires elements of complex analysis (see, for example, [2]). Suppose $f$ is a function that is analytic inside and on a simple non-self-intersecting curve $C$ going once counter-clockwise around a region $R$ (with $C$ the boundary of $R$), then the number of zeros of $f$ counting multiplicity is $(2\pi i)^{-1} \oint_C \left[ f'(z)/f(z) \right] dz$ provided $f$ has no zeros on $C$. Now suppose $g$ is another function that is also analytic inside and on $C$ where $\left| g'(z)/g(z) - f'(z)/f(z) \right| < 2\pi \,\text{length}(C)$ for all $z$ on $C$. Since the value of $(2\pi i)^{-1} \oint_C \left[ g'(z)/g(z) \right] dz$ must also be an integer and differs from $(2\pi i)^{-1} \oint_C \left[ f'(z)/f(z) \right] dz$ by less than one, the two integrals must have the same value. Thus $f$ and $g$ have the same number of zeros counting multiplicity in $R$.

Note that $f'(z) = \text{trace}\left[ (zI - A)^{-1} \right] \det(zI - A)$ and so $f'(z)/f(z) = \text{trace}\left[ (zI - A)^{-1} \right]$. Also, by the formula for the induced $\infty$-norm $|\text{trace}\,[B]| \le n \, \|B\|_\infty$.

If $\lambda \in \sigma(A)$, let $C_{\lambda,\epsilon}$ be the circle of radius $\epsilon$ centered at $\lambda$ going once counter-clockwise around $\lambda$. Without loss of generality, suppose that $\epsilon$ is less than half the distance from $\lambda$ to the nearest different eigenvalue of $A$. Choose

$$\delta = \frac{1}{2 \, \max \left\{ 2n \left\| (zI - A)^{-1} \right\|_\infty^2, \left\| (zI - A)^{-1} \right\|_\infty \mid z \in C_{\lambda,\epsilon} \right\}}.$$

Then provided $\|E\|_\infty < \delta$ we have

$$\begin{aligned}
&\left| \text{trace}\left[ (zI - A - E)^{-1} \right] - \text{trace}\left[ (zI - A)^{-1} \right] \right| \\
&\quad \le n \left\| (zI - A - E)^{-1} - (zI - A)^{-1} \right\|_\infty \\
&\quad \le n \, \|E\|_\infty \left\| (zI - A - E)^{-1} \right\|_\infty \left\| (zI - A)^{-1} \right\|_\infty \\
&\quad \le n \, \|E\|_\infty \frac{\left\| (zI - A)^{-1} \right\|_\infty^2}{1 - \|E\|_\infty \left\| (zI - A)^{-1} \right\|_\infty} < \frac{1}{2}.
\end{aligned}$$

Thus the number of eigenvalues of $A + E$ in $C_{\lambda,\epsilon}$ is the same as the number of eigenvalues of $A$ in $C_{\lambda,\epsilon}$ counting multiplicity. So every eigenvalue of $A$ must be

within $\epsilon$ of some eigenvalue of $A + E$. Since the number of eigenvalues of $A + E$ counting multiplicity is also $n$, every eigenvalue of $A + E$ must be within $\epsilon$ of some eigenvalue of $A$. □

Perhaps the most celebrated eigenvalue "perturbation" theorem is the Gershgorin theorem for perturbations of a diagonal matrix:

**Theorem 2.25** *(Gershgorin's theorem) If $A$ is a complex $n \times n$ matrix, then every eigenvalue of $A$ is in the union of disks*

$$D_j = \left\{ z \in \mathbb{C} \mid \left| z - a_{jj} \right| \leq \sum_{k:k \neq j} \left| a_{jk} \right| \right\}, \qquad j = 1, 2, \ldots, n,$$

*in the complex plane. Furthermore, if a subset of $m$ of these disks do not intersect any other of these disks, then the union of these $m$ disks contains exactly $m$ eigenvalues counting multiplicities.*

***Proof*** For the first part of the theorem, suppose $A\boldsymbol{v} = \lambda \boldsymbol{v}$ with $\boldsymbol{v} \neq \boldsymbol{0}$. Let $\ell$ be an index where $|v_\ell| = \max_j \left| v_j \right| = \|\boldsymbol{v}\|_\infty$. Then

$$|(a_{\ell\ell} - \lambda) v_\ell| = \left| \sum_{k:k \neq \ell} a_{\ell k} v_k \right| \leq \sum_{k:k \neq \ell} |a_{\ell k}| \, |v_k| \leq \left( \sum_{k:k \neq \ell} |a_{\ell k}| \right) |v_\ell| \, .$$

Dividing by $|v_\ell|$ gives the bound $|a_{\ell\ell} - \lambda| \leq \sum_{k:k \neq \ell} |a_{\ell k}|$. Thus $\lambda \in D_\ell$. Repeating this for every eigenvalue shows that every eigenvalue of $A$ is in the union $\bigcup_{j=1}^{n} D_j$.

For the second part of the theorem, let $\widehat{A}(r)$ be the matrix given by $\widehat{a}_{jj}(r) = a_{jj}$ while if $j \neq k$ we set $\widehat{a}_{jk}(r) = r\, a_{jk}$. Clearly $\widehat{A}(0)$ is the diagonal part of $A$ while $\widehat{A}(1) = A$ is the original matrix. Let $\widehat{D}_j(r) = \left\{ z \in \mathbb{C} \mid \left| z - a_{jj} \right| \leq r \sum_{k:k \neq j} \left| a_{jk} \right| \right\}$, which are the Gershgorin disks for $\widehat{A}(r)$. If $r < r'$ then $\widehat{D}_j(r) \subset \widehat{D}_j(r')$. Suppose $\left\{ D_{j_1}, D_{j_2}, \ldots, D_{j_m} \right\}$ is a collection of Gershgorin disks that do not intersect any other Gershgorin disk of $A$. Then $\left\{ \widehat{D}_{j_1}(r), \widehat{D}_{j_2}(r), \ldots, \widehat{D}_{j_m}(r) \right\}$ is a collection of Gershgorin disks that do not intersect any other Gershgorin disk of $\widehat{A}(r)$. If $0 < \epsilon < \min_{k,\ell} |a_{kk} - a_{\ell\ell}|$, then the number of eigenvalues of $\widehat{A}(\epsilon)$ in $\widehat{D}_{j_1}(\epsilon) \cup \widehat{D}_{j_2}(\epsilon) \cup \cdots \cup \widehat{D}_{j_m}(\epsilon)$ is exactly $m$. Because of the continuity of the spectrum (Theorem 2.24), $\widehat{D}_{j_1}(r) \cup \widehat{D}_{j_2}(r) \cup \cdots \cup \widehat{D}_{j_m}(r)$ has exactly $m$ eigenvalues counting multiplicities for all $0 \leq r \leq 1$. Setting $r = 1$ gives the conclusion we wish. □

A related perturbation result is also straightforward to prove:

**Theorem 2.26** *(Bauer–Fike perturbation theorem) If $A$ is an $n \times n$ complex matrix and $X^{-1}AX = diag(\lambda_1, \lambda_2, \ldots, \lambda_n)$ then for any $p \geq 1$, and $\mu \in \sigma(A + E)$ we have the bound*

$$\min_{\lambda : \lambda \in \sigma(A)} |\lambda - \mu| \leq \kappa_p(X) \, \|E\|_p \, .$$

This relates the bound on the perturbation of the eigenvalues in terms of the condition number of a matrix of eigenvectors $X$.

***Proof*** First note that $\sigma(A) = \{\lambda_1, \lambda_2, \ldots, \lambda_n\}$. Let $D = \text{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$, and $F = X^{-1}EX$. Since $A + E - \mu I$ is *not* invertible, neither is $X^{-1}(A + E - \mu I)X = D + F - \mu I$. Therefore,

$$\begin{aligned} 1 &\leq \left\| (D - \mu I)^{-1} F \right\|_p \leq \left\| (D - \mu I)^{-1} \right\|_p \|F\|_p \\ &\leq \left\| (D - \mu I)^{-1} \right\|_p \left\| X^{-1} \right\|_p \|X\|_p \, \|E\|_p \\ &\leq \frac{1}{\min_{\lambda \in \sigma(A)} |\lambda - \mu|} \kappa_p(X) \, \|E\|_p \, . \end{aligned}$$

Re-arranging gives $\min_{\lambda:\lambda\in\sigma(A)} |\lambda - \mu| \leq \kappa_p(X) \, \|E\|_p$. □

A perturbation theorem for eigenvalues of symmetric or complex Hermitian matrices is the Wielandt–Hoffman theorem which is the subject of Exercise 3.

Eigenvectors can change discontinuously for repeated eigenvalues, even for symmetric and complex Hermitian matrices, so any perturbation theorem must take this into account.

**Theorem 2.27** *Suppose $A$ $(n \times n)$ has a simple eigenvalue $\lambda$ with normalized eigenvector $\boldsymbol{v}$ and $A^T$ has a corresponding eigenvector $\boldsymbol{z}$ $(A^T\boldsymbol{z} = \lambda\boldsymbol{z})$. Then $A + E$ has an eigenvector $\boldsymbol{w}$ where*

$$\|\boldsymbol{w} - \boldsymbol{v}\|_2 \leq \left\| \begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix}^+ \right\|_2 \left( 1 + \frac{1}{|\cos\theta|} \right) \|E\|_2 + \mathcal{O}\left( \|E\|_2^2 \right) \qquad \text{as } \|E\|_2 \to 0.$$

*Note that $\theta$ is the angle between $\boldsymbol{v}$ and $\boldsymbol{z}$.*

***Proof*** Let $A(s) = A + s\widehat{E}$ with $\widehat{E} = E/\|E\|_2$, and consider the smooth equations $A(s)\boldsymbol{v}(s) = \lambda(s)\boldsymbol{v}(s)$ and $\|\boldsymbol{v}(s)\|_2^2 = 1$. Under our assumptions have locally unique and smooth solutions. Differentiating these equations with respect to $s$ and substituting $s = 0$ gives

$$\begin{aligned} \widehat{E}\boldsymbol{v} + A\boldsymbol{u} &= \mu\boldsymbol{v} + \lambda\boldsymbol{u} \qquad \text{and} \\ \boldsymbol{v}^T\boldsymbol{u} &= 0, \end{aligned}$$

where $\boldsymbol{u} = d\boldsymbol{v}/ds(0)$ and $\mu = d\lambda/ds(0)$. Re-arranging the first of these equations gives $(A - \lambda I)\boldsymbol{u} = (\mu I - \widehat{E})\boldsymbol{v}$. Pre-multiplying by $\boldsymbol{z}^T$ gives $0 = \boldsymbol{z}^T(\mu I - \widehat{E})\boldsymbol{v}$. Solving for $\mu$ gives $\mu = \boldsymbol{z}^T\widehat{E}\boldsymbol{v}/\boldsymbol{z}^T\boldsymbol{v}$. Therefore, $(A - \lambda I)\boldsymbol{u} = [\boldsymbol{v}\boldsymbol{z}^T/(\boldsymbol{z}^T\boldsymbol{v}) - I]\widehat{E}\,\boldsymbol{v}$. Note that the right-hand side is orthogonal to $\boldsymbol{z}$ since $\boldsymbol{z}^T[\boldsymbol{v}\boldsymbol{z}^T/(\boldsymbol{z}^T\boldsymbol{v}) - I] = (\boldsymbol{z}^T\boldsymbol{v})\boldsymbol{z}^T/(\boldsymbol{z}^T\boldsymbol{v}) - \boldsymbol{z}^T = \boldsymbol{0}$. The null space of $A^T - \lambda I$ is span $\{\boldsymbol{z}\}$ and $\boldsymbol{v} \neq \boldsymbol{0}$, so the null space of $\begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix}^T$ is span $\left\{ \begin{bmatrix} \boldsymbol{z} \\ 0 \end{bmatrix} \right\}$. By Theorem 8.6, the range of $\begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix}$ is the orthogonal complement of span $\left\{ \begin{bmatrix} \boldsymbol{z} \\ 0 \end{bmatrix} \right\}$. Therefore,

$$\begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix} \boldsymbol{u} = \begin{bmatrix} [\boldsymbol{v}\boldsymbol{z}^T/(\boldsymbol{z}^T\boldsymbol{v}) - I]\widehat{E}\,\boldsymbol{v} \\ 0 \end{bmatrix}$$

has a solution $\boldsymbol{u}$, which is given by

$$\boldsymbol{u} = \begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix}^+ \begin{bmatrix} [\boldsymbol{v}\boldsymbol{z}^T/(\boldsymbol{z}^T\boldsymbol{v}) - I]\widehat{E}\,\boldsymbol{v} \\ 0 \end{bmatrix}.$$

The 2-norm of $\boldsymbol{u}$ is bounded by

$$\|\boldsymbol{u}\|_2 \leq \left\| \begin{bmatrix} A - \lambda I \\ \boldsymbol{v}^T \end{bmatrix}^+ \right\|_2 \left( \left\| \frac{\boldsymbol{v}\boldsymbol{z}^T}{\boldsymbol{z}^T\boldsymbol{v}} \right\|_2 + 1 \right) \left\| \widehat{E} \right\|_2 \|\boldsymbol{v}\|_2 .$$

Note that $\left\| \widehat{E} \right\|_2 = 1$, $\|\boldsymbol{v}\|_2 = 1$, and $\left\| \boldsymbol{v}\boldsymbol{z}^T \right\|_2 / \left| \boldsymbol{z}^T\boldsymbol{v} \right| = \|\boldsymbol{v}\|_2 \|\boldsymbol{z}\|_2 / (|\cos\theta| \|\boldsymbol{v}\|_2 \|\boldsymbol{z}\|_2) = 1/|\cos\theta|$. We then obtain the desired bounds with $\boldsymbol{w} = \boldsymbol{v} + s\boldsymbol{u} + \mathcal{O}(s^2)$ using $s = \|E\|_2$. □

Note that if $A$ is symmetric then we can take $\boldsymbol{z} = \boldsymbol{v}$, and note that $\left\| \boldsymbol{v}\boldsymbol{v}^T/(\boldsymbol{v}^T\boldsymbol{v}) - I \right\|_2 \leq 1$, to obtain the improved bounds

$$\|\boldsymbol{w} - \boldsymbol{v}\|_2 \leq \frac{\|E\|_2}{\min_{\mu \in \sigma(A),\, \mu \neq \lambda} |\lambda - \mu|} + \mathcal{O}(\|E\|_2^2).$$

Even for real symmetric matrices, the eigenvectors of nearly repeated eigenvalues are not stable. Consider the perturbation of

$$A_\epsilon = \left[\begin{array}{c|c} 1 & \\ \hline & 1+\epsilon \end{array}\right] \quad \text{to} \quad A'_\epsilon = \left[\begin{array}{c|c} 1 & \epsilon \\ \hline \epsilon & 1+\epsilon \end{array}\right].$$

This perturbation has size $\mathcal{O}(\epsilon)$, which is the same order as the separation of the eigenvalues. The eigenvalues of $A'_\epsilon$ are

$$\lambda_\pm = 1 + \epsilon\frac{1 \pm \sqrt{5}}{2},$$

while for an eigenvector $\boldsymbol{v} = [v_1,\ v_2]^T$ of $A'_\epsilon$ with eigenvalue $\lambda_-$,

$$v_2/v_1 = \frac{1 - \sqrt{5}}{2},$$

while for any eigenvector $\boldsymbol{v} = [v_1,\ v_2]^T$ of $A_\epsilon$ with eigenvalue 1, $v_2/v_1 = 0$. In these cases, if $A = A_\epsilon$ and $E = A'_\epsilon - A_\epsilon$ we have both $\min_{\mu \in \sigma(A),\, \mu \neq \lambda} |\lambda - \mu| = \epsilon$ and $\|E\|_2 = \epsilon$.

**Algorithm 35** Orthogonal iteration

```
1   function orthogiteration(A, U^(0), m)
2     for k = 0, 1, 2, ..., m − 1
3       V^(k) ← A U^(k)
4       V^(k) = U^(k+1) R^(k+1) (QR factorization)
5     end for
6     return U^(k+1)
7   end function
```

### 2.5.3 The QR Algorithm

While the power method and the related methods in Section 2.5.1 can be effective at finding a single eigenvalue and its eigenvector, when we want to find a complete eigen-decomposition, we need something better. That better algorithm is widely understood to be the QR algorithm, developed independently in 1961 by John G. Francis and by Vera Kublanovskaya [94, 148].

#### 2.5.3.1 Orthogonal Iteration

The starting point for these algorithms is *orthogonal iteration*: For an $n \times n$ matrix $A$, start with an $n \times n$ orthogonal matrix $U^{(0)}$. The algorithm is shown in Algorithm 35.

Note that $A\,U^{(k)} = U^{(k+1)}\,R^{(k+1)}$ so

$$\begin{aligned} A^k\,U^{(0)} &= A^{k-1}\,A\,U^{(0)} = A^{k-1}\,U^{(1)}\,R^{(1)} \\ &= A^{k-2}\,U^{(2)}\,R^{(2)}\,R^{(1)} \\ &\;\;\vdots \\ &= U^{(k)}\,R^{(k)}\cdots R^{(2)}\,R^{(1)}. \end{aligned}$$

Since the product of upper triangular matrices is upper triangular we can write $A^k\,U^{(0)} = U^{(k)}\,R^{[k]}$ where $R^{[k]} = R^{(k)}\cdots R^{(2)}\,R^{(1)}$. Setting $U^{(j)} = [\boldsymbol{u}^{(j)},\ \widetilde{U}^{(j)}]$ and partitioning $R^{(k)}$ consistently gives

$$\begin{aligned} U^{(j)}R^{(j)} &= \left[\,\boldsymbol{u}^{(j)},\ \widetilde{U}^{(j)}\,\right]\left[\begin{array}{c|c} \rho^{(j)} & \boldsymbol{r}^{(j)\,T} \\ \hline & \widetilde{R}^{(j)} \end{array}\right] = \left[\,\rho^{(j)}\boldsymbol{u}^{(j)},\ \widetilde{U}^{(j)}\widetilde{R}^{(j)} + \boldsymbol{u}^{(j)}\boldsymbol{r}^{(j)\,T}\,\right] \\ A\,U^{(j-1)} &= A\left[\,\boldsymbol{u}^{(j-1)},\ \widetilde{U}^{(j-1)}\,\right] = \left[\,A\,\boldsymbol{u}^{(j-1)},\ A\,\widetilde{U}^{(j-1)}\,\right] \end{aligned}$$

so the first column of $U^{(j)}$, $\rho^{[j]}\boldsymbol{u}^{(j)} = A\,\boldsymbol{u}^{(j-1)}$. So the first column of $U^{(j)}$ acts as if the power method is being applied to it. Since $U^{(j)}$ is orthogonal its inverse is equal to its transpose. Note that

$$(RS)^{-T} = ((RS)^{-1})^T = (S^{-1}R^{-1})^T = R^{-T}S^{-T}.$$

So

$$A^{-T}U^{(j-1)} = U^{(j)}(R^{(j)})^{-T}.$$

But here, $(R^{(j)})^{-T}$ is lower triangular. Writing $U^{(j)} = [\,\widehat{U}^{(j)},\ \boldsymbol{u}_n^{(j)}\,]$ and partitioning $(R^{(j)})^{-T} = L^{(j)}$ consistently we get

$$\left[\,\widehat{U}^{(j)}\ \boldsymbol{u}_n^{(j)}\,\right]\left[\begin{array}{c|c}\widehat{L}^{(j)} & \\ \hline \boldsymbol{\ell}^{(j)\,T} & \lambda^{(j)}\end{array}\right] = \left[\,\widehat{U}^{(j)}\widehat{L}^{(j)} + \boldsymbol{u}_n^{(j)}\boldsymbol{\ell}^{(j)\,T},\ \boldsymbol{u}_n^{(j)}\lambda^{(j)}\,\right]$$
$$= \left[\,A^{-T}\,\widehat{U}^{(j-1)},\ A^{-T}\,\boldsymbol{u}_n^{(j-1)}\,\right].$$

This means that the *last* column $\boldsymbol{u}_n^{(j)}$ of $U^{(j)}$ essentially has the power method for $A^{-T}$ applied to it. So $\boldsymbol{u}_n^{(j)}$ should converge to an eigenvector for the *smallest* eigenvalue of $A^T$.

After the first column of $U^{(j)}$ has converged to $\alpha^{(j)}\boldsymbol{v}_1$, with $\boldsymbol{v}_1$ the unit eigenvector for the dominant eigenvalue of $A$, the second column of $U^{(j)}$ can be thought of as undergoing the iteration $\boldsymbol{u}_2^{(j)} = \pm P_1\,A\,\boldsymbol{u}_2^{(j)} / \left\| P_1\,A\,\boldsymbol{u}_2^{(j)} \right\|_2$ where $P_1$ is the orthogonal projection $P_1 = I - \boldsymbol{v}_1\boldsymbol{v}_1^T$. This again works like the power method using the matrix $PA$. Assuming the power method for this matrix converges, the limit is an eigenvector $\boldsymbol{v}_2$ of $PA$: $PA\boldsymbol{v}_2 = \lambda_2\boldsymbol{v}_2$. Then $A\boldsymbol{v}_2 - \boldsymbol{v}_1\boldsymbol{v}_1^T A\boldsymbol{v}_2 = \lambda_2\boldsymbol{v}_2$ and so $A\boldsymbol{v}_2 = \mu_{1,2}\boldsymbol{v}_1 + \lambda_2\boldsymbol{v}_2$. That is,

$$A\,[\boldsymbol{v}_1,\ \boldsymbol{v}_2] = [\boldsymbol{v}_1,\ \boldsymbol{v}_2]\begin{bmatrix}\lambda_1 & \mu_{1,2}\\ & \lambda_2\end{bmatrix}.$$

After convergence of the first two columns of $U^{(j)}$, the third column essentially undergoes the iteration $\boldsymbol{u}_3^{(j)} = \pm P_2 A\boldsymbol{u}_3^{(j)} / \left\| P_2\,A\,\boldsymbol{u}_3^{(j)} \right\|_2$ where $P_2 = I - [\boldsymbol{v}_1,\ \boldsymbol{v}_2]\,[\boldsymbol{v}_1,\ \boldsymbol{v}_2]^T$, noting that $\boldsymbol{v}_2$ is orthogonal to $\boldsymbol{v}_1$. The limiting eigenvector (assuming convergence) is then $\boldsymbol{v}_3$ where $P_2 A\boldsymbol{v}_3 = \lambda_3\boldsymbol{v}_3$ which means that $A\boldsymbol{v}_3 = \mu_{1,3}\boldsymbol{v}_1 + \mu_{2,3}\boldsymbol{v}_2 + \lambda_3\boldsymbol{v}_3$, that is,

$$A\,[\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3] = [\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3]\begin{bmatrix}\lambda_1 & \mu_{1,2} & \mu_{1,3}\\ & \lambda_2 & \mu_{2,3}\\ & & \lambda_3\end{bmatrix}.$$

Continuing in this way we can justify the claim that each column $\boldsymbol{u}_\ell^{(j)}$ approaches span $\{\boldsymbol{v}_\ell\}$ as $j \to \infty$ where the $\boldsymbol{v}_\ell$'s are orthonormal and

$$A\,[\boldsymbol{v}_1,\ \boldsymbol{v}_2, \ldots,\ \boldsymbol{v}_n] = [\boldsymbol{v}_1,\ \boldsymbol{v}_2, \ldots,\ \boldsymbol{v}_n]\ T \qquad \text{with } T \text{ upper triangular,}$$

at least provided $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$. We should be careful about this line of reasoning as it gives the impression that the convergence is sequential, first for the first column, then for the second column, and so on, when in fact it is simultaneous.

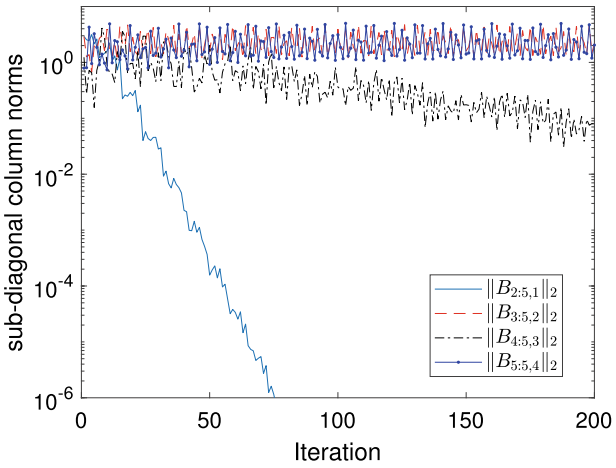



**Fig. 2.5.3** Norms of sub-diagonal columns for orthogonal iteration

**Table 2.5.1** Eigenvalues of $A$ from (2.5.3)

| $k$ | Re $\lambda_k$ | Im $\lambda_k$ | $\lvert\lambda_k\rvert$ |
|---|---|---|---|
| 1 | +3.26537768 | 0 | 3.2653777 |
| 2 | +0.90230447 | +2.468506906 | 2.6282465 |
| 3 | +0.90230447 | −2.468506906 | 2.6282465 |
| 4 | −1.03499331 | +2.371325340 | 2.5873529 |
| 5 | −1.03499331 | −2.371325340 | 2.5873529 |

The quantities that are most important are the ratios $|\lambda_{r+1}| \,/\, |\lambda_r|$. The smaller these are, the faster the convergence.

For the example matrix in (2.5.3), the norms of the sub-diagonal columns of $U^{(j)\,T} A\, U^{(j)}$ are shown in Figure 2.5.3. To understand why the method behaves this way, we need to know the eigenvalues and their magnitudes, which are shown in Table 2.5.1.

Note that the eigenvalue magnitudes are $|\lambda_1| > |\lambda_2| = |\lambda_3| > |\lambda_4| = |\lambda_5|$. So we do not expect $b_{3,2}^{(j)}$ or $b_{5,4}^{(j)}$ to go to zero where $B^{(j)} = U^{(j)\,T} A\, U^{(j)}$. Furthermore, the rate at which $b_{2,1}^{(j)} \to 0$ as $j \to \infty$ is controlled by $|\lambda_2| \,/\, |\lambda_1| \approx 0.805$, while the rate at which $b_{4,3}^{(j)} \to 0$ as $j \to \infty$ is controlled by $|\lambda_4| \,/\, |\lambda_3| \approx 0.9844$. This clearly indicates much faster convergence for $b_{2,1}^{(j)}$ (and other entries below the (2, 1) entry) than for $b_{4,3}^{(j)}$.

Justification for an asymptotic convergence rate of $|\lambda_{r+1}| \,/\, |\lambda_r|$ in the entries can be found, for example, in [105, Sec. 7.3.2, pp. 332–334].

**Algorithm 36** The basic QR algorithm

```
1   function QRalgorithm(A_0, m)
2     U_0 ← I
2     for j = 0, 1, ..., m − 1
3       A_j = Q_j R_j  // QR factorization
4       A_{j+1} ← R_j Q_j ;  U_{j+1} ← U_j Q_j
5     end for
6     return (A_m, U_m)
7   end
```

#### 2.5.3.2 The Basic QR Algorithm

The basic QR algorithm is shown in Algorithm 36.

At first it can seem that the QR algorithm has nothing to do with finding eigenvalues and eigenvectors. But the connections become apparent with some analysis. Since $A_j = Q_j R_j$ from the QR factorization, we have $R_j = \overline{Q_j}^T A_j$ so $A_{j+1} = R_j Q_j = \overline{Q_j}^T A_j Q_j$ so that $A_{j+1}$ is unitarily similar to $A_j$. Thus $A_{j+1}$ and $A_j$ have the same eigenvalues with the same multiplicities. If $\overline{U_j}^T A_0 U_j = A_j$ then

$$A_{j+1} = \overline{Q_j}^T A_j Q_j = \overline{Q_j}^T \overline{U_j}^T A_0 U_j Q_j = \overline{U_{j+1}}^T A_0 U_{j+1}.$$

Since $U_0 = I$, mathematical induction lets us conclude that $\overline{U_j}^T A_0 U_j = A_j$ for $j = 0, 1, 2, \ldots$. This is closely related to orthogonal iteration: $A U^{(j)} = U^{(j+1)} R^{(j+1)}$ (QR factorization). In fact, we will now show that $U^{(j)}$ from orthogonal iteration is, under a suitable restriction on the QR factorization used, identical to the $U_j$ matrix obtained from the QR algorithm for $j = 0, 1, 2, \ldots$.

**Lemma 2.28** *Suppose the QR factorization algorithm is chosen so that the R matrix has only non-negative entries. Then provided $A = A_0$ is invertible, we have $U^{(j)} = U_j$ for $j = 0, 1, 2, \ldots$ where $U^{(j)}$ is the matrix obtained from orthogonal iteration in Algorithm 35 with $U^{(0)} = I$, and $U_j$ the matrix obtained from the QR algorithm in Algorithm 36.*

***Proof*** First note that $U^{(0)} = U_0 = I$. This is the base case for our induction proof.

Suppose that $U^{(j)} = U_j$ and $A = A_0$. Then $A_j = Q_j R_j$. On the other hand,

$$A_j = \overline{U_j}^T A U_j = \overline{U^{(j)}}^T A U^{(j)} = \overline{U^{(j)}}^T U^{(j+1)} R^{(j+1)}, \quad \text{so}$$
$$Q_j R_j = \overline{U^{(j)}}^T U^{(j+1)} R^{(j+1)}.$$

By uniqueness of QR factorizations for invertible matrices, there is a diagonal matrix $D_j$ with diagonal entries $e^{i\theta}$, where $\overline{U^{(j)}}^T U^{(j+1)} = Q_j D_j$ and $\overline{D_j} R_j = R^{(j+1)}$. To simplify the analysis, we can assume that the QR factorization is implemented so that the diagonal entries of the "$R$" matrix of the QR factorization are positive. Then

$D_j = I$. Also, $\overline{U^{(j)}}^T U^{(j+1)} = Q_j$ and $R_j = R^{(j+1)}$. The first equation means that $U^{(j+1)} = U^{(j)} Q_j = U_j Q_j = U_{j+1}$. This completes the induction step.

By mathematical induction, $U^{(j)} = U_j$ and $R^{(j+1)} = R_j$ for $j = 0, 1, 2, \ldots$. □

Lemma 2.28 also implies that the convergence rate of the basic QR algorithm is the convergence rate of orthogonal iteration. As we have seen, this can be very slow. If we view orthogonal iteration as a generalization of the power method, then we can see that shifting the eigenvalues can be a very effective strategy. It can also be applied to the QR algorithm. We will see how in the next section.

#### 2.5.3.3 Shifting

Shifting can be applied to orthogonal iteration and the QR algorithm by replacing $A$ with $A - \mu I$ and $A_j$ with $A_j - \mu I$ for computing the QR factorization. The shifted orthogonal iteration is $(A - \mu I)U^{(j)} = U^{(j+1)} R^{(j+1)}$ and the shifted QR algorithm step is $A_j - \mu I = Q_j R_j$ and $A_{j+1} \leftarrow R_j Q_j + \mu I$. The matrix $\mu I$ must be added back in to restore $A_{j+1}$ to be similar to $A_j$:

$$\begin{aligned} R_j &= \overline{Q_j}^T (A_j - \mu I) = \overline{Q_j}^T A_j - \mu \overline{Q_j}^T, \text{ and so} \\ A_{j+1} &= R_j Q_j + \mu I = (\overline{Q_j}^T A_j - \mu \overline{Q_j}^T) Q_j + \mu I \\ &= \overline{Q_j}^T A_j Q_j - \mu I + \mu I = \overline{Q_j}^T A_j Q_j. \end{aligned}$$

As with the basic QR algorithm, we suppose that $U_j = U^{(j)}$. We want to show that $U_{j+1} = U^{(j+1)}$. Since $A_j - \mu I = Q_j R_j$,

$$\begin{aligned} A_j - \mu I &= \overline{U_j}^T (A - \mu I) U_j = \overline{U^{(j)}}^T (A - \mu I) U^{(j)} = \overline{U^{(j)}}^T U^{(j+1)} R^{(j+1)}, \quad \text{so} \\ Q_j R_j &= \overline{U^{(j)}}^T U^{(j+1)} R^{(j+1)}. \end{aligned}$$

Again assuming that the "$R$" in the QR factorization has positive diagonal entries, we have $R_j = R^{(j+1)}$ and $U^{(j+1)} = U^{(j)} Q_j = U_j Q_j = U_{j+1}$, as we wanted.

The real advantage of shifting is not what happens to the first column of $U^{(j)}$ where $\boldsymbol{u}_1^{(j+1)} = (A - \mu I)\boldsymbol{u}_1^{(j)} / \left\| (A - \mu I)\boldsymbol{u}_1^{(j)} \right\|_2$. Rather, it is the *last* column where

$$\boldsymbol{u}_n^{(j+1)} = \frac{(A - \mu I)^{-T} \boldsymbol{u}_n^{(j)}}{\left\| (A - \mu I)^{-T} \boldsymbol{u}_n^{(j)} \right\|_2}$$

that features the fastest convergence: if $\mu$ is close to an eigenvalue of $A$ then the component in the direction of the corresponding eigenvector is greatly amplified. Estimates of an eigenvalue can be obtained from $A_j$ by using the bottom-right entry $(A_j)_{n,n} = \boldsymbol{u}_n^{(j)\,T} A \boldsymbol{u}_n^{(j)}$. This gives superlinear convergence of the eigenvalue estimate

**Algorithm 37** The QR algorithm with shifts

```
1   function QRshift(A_0, m)
2     U_0 ← I ;  μ ← 0
2     for  j = 0, 1, ..., m − 1
3       A_j − μI = Q_j R_j  // QR factorization
4       A_{j+1} ← R_j Q_j + μI ;  U_{j+1} ← U_j Q_j
5       μ ← smallest eigenvalue of [ (A_j)_{n−1,n−1}  (A_j)_{n−1,n} ]
                                   [ (A_j)_{n,n−1}    (A_j)_{n,n}   ]
5     end for
6     return (A_m, U_m)
7   end function
```

as for inverse iteration (Algorithm 33). If the matrix $A$ is real, this method will still give only real shifts, and will not make explicit any complex eigenvalues. A better strategy is to use one of the eigenvalues of the bottom-right $2 \times 2$ submatrix

$$\begin{bmatrix} (A_j)_{n-1,n-1} & (A_j)_{n-1,n} \\ (A_j)_{n,n-1} & (A_j)_{n,n} \end{bmatrix}.$$

When the ratio between the largest and smallest eigenvalues $|\lambda_1| / |\lambda_n|$ is large, then it is important to use small shifts. Initially, zero shifts may be initially desirable so that the bottom-right entries are small before we use them to compute shifts. A pseudo-code for a QR algorithm with shifting is shown in Algorithm 37. Results for this algorithm applied to our test matrix (2.5.3) are shown in Figure 2.5.4, showing the convergence of selected sub-diagonal entries.

As can be seen in Figure 2.5.4, after some initial "dithering", the $a_{54}$ entry goes to zero quite rapidly. If we continued the graph, then the size of $a_{54}$ goes well below unit roundoff; in fact, in 20 iterations its value is $\approx 2.16 \times 10^{-227}$. The $a_{43}$ entry also

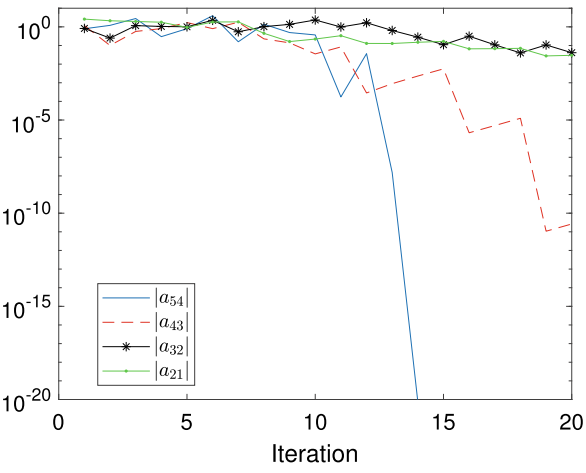



**Fig. 2.5.4** Results for QR algorithm with shifts applied to (2.5.3)

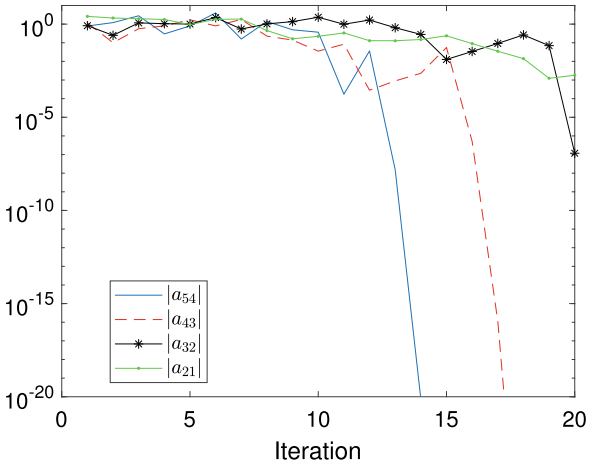



**Fig. 2.5.5** Results for the QR algorithm with shifts and deflation for (2.5.3)

decreases in size dramatically in 20 iterations, but $a_{32}$ and $a_{21}$ are reduced moderately (by about a factor of 10). It is the targeting of the eigenvalue of the last $2 \times 2$ matrix that gives us this dramatic reduction in $a_{54}$.

Since convergence of the bottom-right entry $(A_j)_{n,n}$ to an eigenvalue is fast, once this eigenvalue estimate is sufficiently accurate, how can we accelerate the convergence of the other eigenvalue estimates? The answer is *deflation*. This is where some very small entries are set to zero, and the connection between this eigenvalue and the rest of the matrix is severed. For example, when the entries $(A_j)_{n,k}$ for $k < n$ of the last row are no more than perhaps unit roundoff $\mathbf{u}$ times $(A_j)_{n,n}$, then we can set $(A_j)_{n,k} \leftarrow 0$ for $k = 1, 2, \ldots, n-1$. If we do this, then $(A_j)_{n,n}$ is an eigenvalue of $A_j$ and it is decoupled from the rest of the matrix.

We can then start processing the top-left $(n-1) \times (n-1)$ submatrix of $A_j$ and start targeting its bottom-right $2 \times 2$ submatrix. As we do this, we should remember that the last column of $A_j$ will be modified with each iteration, even though the orthogonal matrices we apply to it will be determined by the top-left $(n-1) \times (n-1)$ submatrix of $A_j$. Results of this shift-deflate-and-retarget strategy are shown in Figure 2.5.5. Note that the matrix was completely deflated in 21 iterations. This method is sequential in the sense that it can only target one eigenvalue for accelerated convergence at a time. Nevertheless, the matrices $A_j$ converge rapidly once it starts to approach moderately accurate estimates.

#### 2.5.3.4 Speeding QR Iterations

Shifting and deflating make the method converge rapidly in terms of the number of iterations. But we can also make each iteration fast. The cost of a QR factorization is $\mathcal{O}(n^3)$ for an $n \times n$ matrix. However, with a single $\mathcal{O}(n^3)$ pre-processing step, we can do each subsequent QR factorization in $\mathcal{O}(n^2)$ operations. Furthermore, for

**Algorithm 38** Constructing Hessenberg matrix $H$ that is unitarily similar to $A$

```
1   function mkhessenberg(A)
2     Q ← I
3     a'_1 ← [a_12, a_13, ..., a_1n]^T
4     v'_1 ← housholder(a'_1); v_1 ← [0, v'^T_1]^T
5     γ_1 ← 2/(v'^T_1 v'_1); P_1 ← I − γ_1 v_1 v_1^T
6     A ← conj(P_1)^T A P_1
7     (H', Q') ← mkhessenberg(A_2:n,2:n)
8     A ← [1, 0; 0, conj(Q')^T] A [1, 0; 0, Q']; Q ← P_1 [1, 0; 0, Q']
9     return (A, Q)
10  end function
```

matrices of special structure, such as real symmetric or complex Hermitian matrices, this pre-processing step means that each subsequent QR factorization can be done in only $\mathcal{O}(n)$ operations.

Since the total number of iterations, assuming that we get good convergence for each targeted eigenvalue, is $\mathcal{O}(n)$, we get a total of $\mathcal{O}(n^3)$ operations for the Schur decomposition of a general $n \times n$ matrix.

To get rapid QR factorizations, we need to pre-process the matrix $A$ into *Hessenberg form*. That is, we want to find a real orthogonal or complex unitary matrix $Q$ so that $H = \overline{Q}^T A\, Q$ is Hessenberg, that is, where $h_{ij} = 0$ if $i > j + 1$. In other words, in $H$, all non-zero entries occur on or above the first sub-diagonal. If $A$ is real symmetric or complex Hermitian, then $H$ must also have this property, and so $H$ must also be tridiagonal. We can compute $H$ using Householder reflectors $P = I - 2\boldsymbol{v}\overline{\boldsymbol{v}}^T/(\overline{\boldsymbol{v}}^T\boldsymbol{v})$ as are used to compute QR factorizations (see Section 2.2.2.4). We must apply these orthogonal matrices in a symmetric fashion to $A$, $\overline{P}^T A\, P = PAP$ as $\overline{P}^T = P$. We choose $\boldsymbol{v}$ so that if $\boldsymbol{a} = \begin{bmatrix} \alpha \\ \boldsymbol{a}' \end{bmatrix}$ is the first column of $A$, $\boldsymbol{v} = \begin{bmatrix} 0 \\ \boldsymbol{v}' \end{bmatrix}$ where $(I - 2\boldsymbol{v}'\overline{\boldsymbol{v}'}^T/(\overline{\boldsymbol{v}'}^T\boldsymbol{v}'))\boldsymbol{a}' = \gamma \left\|\boldsymbol{a}'\right\|_2 \boldsymbol{e}_1$ with $|\gamma| = 1$. Applying $\overline{P}^T A$ will zero every entry in the first column of $A$ except for the first two entries. The first entry will be unchanged, but the second will be $\gamma \left\|\boldsymbol{a}'\right\|_2$. Applying the Householder reflector symmetrically to the columns $\overline{P}^T A\, P$, we note that the first column of $\overline{P}^T A$ is unchanged, preserving the zeros that had just been introduced. Repeating this process recursively to the bottom-right $(n-1) \times (n-1)$ submatrix of $\overline{P}^T A\, P$ will give us a Hessenberg matrix. This is Algorithm 38.

The real advantage of this pre-processing step is how it expedites the QR factorization. The QR factorization of a Hessenberg matrix can be computed by means of Givens' rotations (see Section 2.2.2.5) as shown in Algorithm 39.

The $Q$ matrix in the QR factorization of a Hessenberg matrix is also Hessenberg. This can be seen from the fact that the $Q$ matrix is a product of Givens rotations:

**Algorithm 39** QR factorization of a Hessenberg matrix

```
1   function QRhessenberg(H)
2     Q ← I
3     for k = 1, 2, ..., n − 1
4       (c, s) ← givens(h_kk, h_{k+1,k})
5       Q_{1:n,k:k+1} ← Q_{1:n,k:k+1} [ c | +s ; −s | c ]
6       H_{k:k+1,1:n} ← [ c | −s ; +s | c ] H_{k:k+1,1:n}
7     end for
8     return (H, Q)
9   end
```

$$\begin{bmatrix} c_1 & s_1 & & & \\ -s_1 & c_1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & c_2 & s_2 & & \\ & -s_2 & c_2 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & c_3 & s_3 & \\ & & -s_3 & c_3 & \\ & & & & 1 \end{bmatrix} \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & c_4 & s_4 \\ & & & -s_4 & c_4 \end{bmatrix}$$
$$= \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & & * & * & * \\ & & & * & * \end{bmatrix}.$$

Since $Q$ is Hessenberg and $R$ is upper triangular, $RQ$ is also Hessenberg: $(RQ)_{ij} = \sum_k r_{ik} q_{kj}$ and the sum is taken over all indexes $k$ where $i \leq k \leq j+1$; there can only be a non-zero sum when $i \leq j+1$, showing that $RQ$ is Hessenberg.

We can use the Hessenberg structure to create *chasing algorithms* that compute the QR factorization and the $RQ$ product simultaneously. If $G_1, G_2, \ldots, G_{n-1}$ are the Givens rotation matrices used in the QR factorization of Algorithm 39, then $R = G_{n-1} \cdots G_2 G_1 H$ so $Q = \overline{G_{n-1} \cdots G_2 G_1}^T = \overline{G}_1^T \overline{G}_2^T \cdots \overline{G_{n-1}}^T$. From this, $RQ = G_{n-1} \cdots G_2 G_1 H \overline{G}_1^T \overline{G}_2^T \cdots \overline{G_{n-1}}^T$ which is another Hessenberg matrix $K$. The matrix $G_1 H \overline{G}_1^T$ is *almost* Hessenberg: $(G_1 H \overline{G}_1^T)_{3,1}$ may be non-zero. Since $G_2$ differs from the identity matrix only at entries $(i, j)$ with $i, j \in \{2, 3\}$, it is possible to "patch" this violation of being Hessenberg with $G_2$ by zeroing out the $(3, 1)$ entry using the $(3, 2)$ entry. The resulting matrix $G_2 G_1 H \overline{G}_1^T \overline{G}_2^T$ is again *almost* Hessenberg because $(G_2 G_1 H \overline{G}_1^T \overline{G}_2^T)_{4,2}$ might be non-zero. This time this can be "patched" by using $G_3$ to zero the $(4, 2)$ entry using the $(4, 3)$ entry. Continuing in this way we "chase" the non-Hessenberg entry finally out of the matrix using $G_{n-1}$ to zero out the $(n, n-2)$ entry using the $(n, n-1)$ entry. Figure 2.5.6 illustrates the overall process.

This gives us a new Hessenberg matrix, but is it really (essentially) the same as what we would get using the QR factorization? The answer is *yes*, due to the Implicit

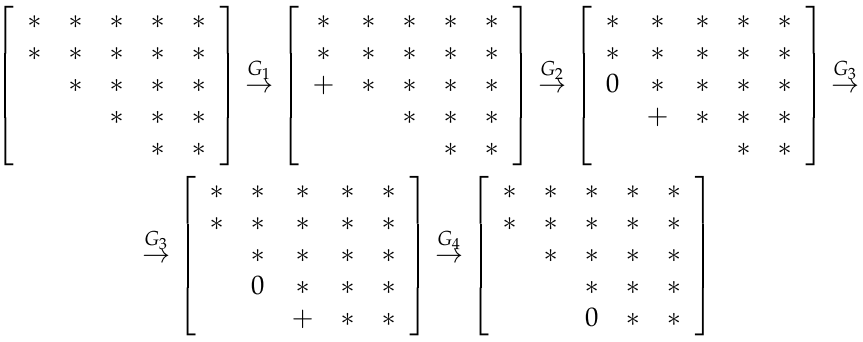



**Fig. 2.5.6** Hessenberg chasing algorithm

$Q$ theorem. We say an $n \times n$ Hessenberg matrix $H$ is *unreduced* if $h_{k+1,k} \neq 0$ for $k = 1, 2, \ldots, n-1$.

**Theorem 2.29** *(Implicit Q theorem) Suppose that B is a real or complex matrix and there are matrices Q and V both orthogonal if B real, and unitary if B complex, such that both* $\overline{Q}^T B\, Q$ *and* $\overline{V}^T B\, V$ *are unreduced Hessenberg matrices. If the first columns of Q and V are the same, then for every j there is a number* $|\gamma_j| = 1$ *where* $Q\boldsymbol{e}_j = \gamma_j\, V\boldsymbol{e}_j$*. Thus there is a diagonal matrix D with diagonal entries of magnitude one where*

$$\overline{Q}^T B\, Q = \overline{D}^T \overline{V}^T B\, V\, D.$$

For a proof, see, for example, [105, Thm. 7.4.2, pp. 346–347].

In our situation, provided $H$ is unreduced Hessenberg, then the result of the chasing algorithm computing the Givens rotations sequentially is essentially equivalent to the original QR factorization approach. In fact, because two QR factorizations $B = Q_1 R_1 = Q_2 R_2$ with $Q_1$ and $Q_2$ real orthogonal or complex unitary are related by $Q_1 = Q_2\, D$ and $R_1 = \overline{D} R_2$ where $D$ is diagonal with diagonal entries of magnitude one, the magnitudes of the entries of $H$ do not depend on how that QR factorization is performed.

On the other hand, if $H$ is Hessenberg but *not unreduced* Hessenberg, then some $h_{k+1,k} = 0$, and the matrix can be deflated.

The remaining question is how to compute $G_1$. If the shift is $\mu$ we want to choose $G_1$ so that

$$\begin{bmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{bmatrix} \begin{bmatrix} h_{11} - \mu \\ h_{21} \end{bmatrix} = \begin{bmatrix} * \\ 0 \end{bmatrix}.$$

The remaining $G_j$'s are computed by the chasing algorithm outlined above. This gives a fast $\mathcal{O}(n^2)$ algorithm of performing the shifted QR step.

It should be noted that the chasing algorithm outlined is not numerically stable in the sense that small errors or perturbations in $G_1$ can result in large errors in the following $G_k$'s, especially if there is some $h_{k+1,k} \approx 0$. Fortunately, this only

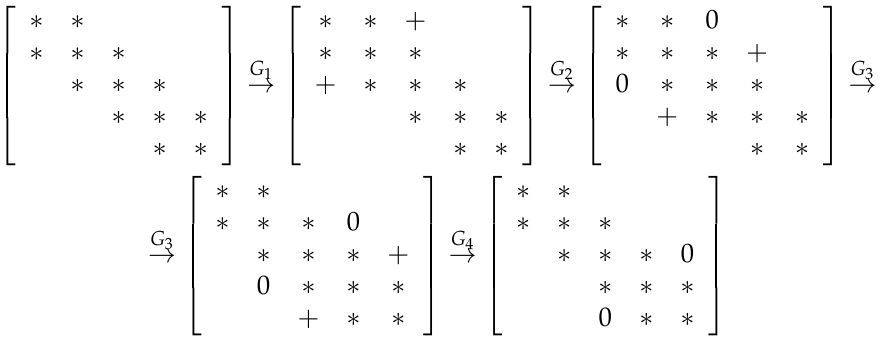



**Fig. 2.5.7** Tridiagonal chasing algorithm

affects the rate of convergence of the method, not the orthogonality of the resulting $Q$ matrix, nor the accuracy of the similarity $fl(\overline{Q_j}^T A_j Q_j) \approx \overline{Q_j}^T A_j Q_j$ using the chasing algorithm.

A special case is where $A$ is real symmetric or complex Hermitian. Then the Hessenberg matrix computed by Algorithm 38 is tridiagonal. Furthermore, we can perform the QR algorithm step using a streamlined chasing algorithm as illustrated in Figure 2.5.7.

There is a variant of the QR algorithm for general real matrices that avoids complex arithmetic. It does not produce a Schur decomposition, as that would necessarily be complex if the matrix has complex eigenvalues. Instead, it computes a *real Schur decomposition*

$$Q^T A Q = T$$

with $Q$ real orthogonal and $T$ real *block* upper triangular with $1 \times 1$ or $2 \times 2$ blocks. Each diagonal $2 \times 2$ represents a pair of complex conjugate eigenvalues. The most difficult part of implementing the QR algorithm for this case is dealing with complex shifts. By appealing to the implicit $Q$ theorem (Theorem 2.29) we can create a chasing algorithm. If we want to use a complex shift $\mu = \rho + i\sigma$ we actually need to also apply the complex conjugate shift $\overline{\mu} = \rho - i\sigma$ in order to keep the arithmetic real. Applying this to a real Hessenberg matrix $H$ we need to find the first column of $(H - \mu I)(H - \overline{\mu} I) = H^2 - 2\rho H + (\rho^2 + \sigma^2)I$. When we compute $(H - \mu I)(H - \overline{\mu} I)\boldsymbol{e}_1$ we get a vector with (in general) three non-zero leading entries:

$$\begin{bmatrix} x \\ y \\ z \\ 0 \\ \vdots \end{bmatrix} = \begin{bmatrix} h_{11}^2 + h_{12}h_{21} - 2\rho h_{11} + \rho^2 + \sigma^2 \\ h_{21}(h_{11} + h_{22} - 2\rho) \\ h_{21}h_{32} \\ 0 \\ \vdots \end{bmatrix}.$$

We need to use Householder reflectors with $\boldsymbol{v}$ vectors of length three, rather than Givens rotations, to zero out all but the first entry. The resulting chasing algorithm now has three "non-Hessenberg" entries in a triangle to chase down and out of the matrix. This means we use $3 \times 3$ Householder reflectors throughout to do the "chasing". Remarkably, Francis worked out all these details in 1961. For full details, see [105, Sec. 7.5, pp. 352–361].

## 2.5.4 Singular Value Decomposition

The *singular value decomposition* (SVD) of an $m \times n$ matrix $A$ is the factorization

$$A = U \, \Sigma \, \overline{V}^T, \tag{2.5.9}$$

where $U$ and $V$ are real orthogonal if $A$ is real or unitary if $A$ is complex, and $\Sigma$ is $m \times n$ and diagonal with diagonal entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$. The values $\sigma_j$ are called *singular values* of $A$. The singular value decomposition is a valuable tool in many areas including statistics where it forms the computational foundation of *principal components analysis* (PCA). It can be used to define the pseudo-inverse of a general rectangular matrix. The SVD is key part of many *dimension reduction* algorithms. It is a common tool in certain data mining tasks, such as in recommender systems.

### 2.5.4.1 Existence of the SVD

**Theorem 2.30** *The singular value decomposition (SVD) exists for any $m \times n$ matrix $A$.*

Many discussions of the SVD point out that $\overline{A}^T A$ is an $n \times n$ positive semi-definite Hermitian or real symmetric matrix, and so has non-negative eigenvalues which can be ordered from largest to smallest $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \geq 0$. An orthonormal basis of eigenvectors can be used to form the $V$ matrix, and we take $\sigma_j = \sqrt{\lambda_j}$. An immediate consequence is that the singular values $\sigma_j$ are indeed functions of $A$ alone, even if there can be different singular value decompositions of $A$. These differences can only occur in $U$ or $V$ when there are repeated eigenvalues.

To complete the existence argument we need to construct the $U$ matrix. We can apply the same arguments to $A \, \overline{A}^T$ for $U$ as we applied to $\overline{A}^T A$ for $V$. But this is not sufficient. We need to show that the non-zero eigenvalues of $\overline{A}^T A$ and $A \, \overline{A}^T$ are the same. Also, we need the $U$ and $V$ to be consistent with each other, which might not occur from the previous arguments if there are repeated eigenvalues. Finally, working with $\overline{A}^T A$ or $A \, \overline{A}^T$ numerically is not a good approach because the error in $\lambda_n$ is $\mathcal{O}(\mathbf{u} \left\| \overline{A}^T A \right\|_2) = \mathcal{O}(\mathbf{u} \, \|A\|_2^2)$; taking square roots amplifies this error if $\lambda_n$ is

small. We use a different approach in the proof that better reflects the computational and numerical issues.

***Proof*** Let

$$C = \left[\begin{array}{c|c} & A \\ \hline \overline{A}^T & \end{array}\right].$$

This is a complex Hermitian or real symmetric matrix. As such there is a complex unitary or real orthogonal matrix $Z$ where $\overline{Z}^T C Z$ is real diagonal with the diagonal entries in decreasing order. Suppose that $\lambda \neq 0$ is an eigenvalue of $C$ with eigenvector $\boldsymbol{z} \neq \boldsymbol{0}$. Then we can split $\boldsymbol{z}$ consistently with the block structure of $C$: $\boldsymbol{z} = [\boldsymbol{u}^T,\ \boldsymbol{v}^T]^T/\sqrt{2}$. Then

$$\left[\begin{array}{c|c} & A \\ \hline \overline{A}^T & \end{array}\right]\left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right] = \lambda\left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right], \qquad \text{so}$$

$$\left[\begin{array}{c|c} & A \\ \hline \overline{A}^T & \end{array}\right]\left[\begin{array}{c} \boldsymbol{u} \\ -\boldsymbol{v} \end{array}\right] = -\lambda\left[\begin{array}{c} \boldsymbol{u} \\ -\boldsymbol{v} \end{array}\right].$$

Thus eigenvalues come in pairs $(\lambda, -\lambda)$ with eigenvectors $[\boldsymbol{u}^T, \boldsymbol{v}^T]^T$ and $[\boldsymbol{u}^T, -\boldsymbol{v}^T]^T$. By orthogonality of eigenvectors with distinct eigenvalues,

$$0 = \overline{\left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right]}^T \left[\begin{array}{c} \boldsymbol{u} \\ -\boldsymbol{v} \end{array}\right] = \overline{\boldsymbol{u}}^T\boldsymbol{u} - \overline{\boldsymbol{v}}^T\boldsymbol{v}.$$

That is $\|\boldsymbol{u}\|_2 = \|\boldsymbol{v}_2\| = 1$. Furthermore, if $\boldsymbol{z}_j, \boldsymbol{z}_k$ are distinct columns of $Z$ with eigenvalues $\lambda_j,\ \lambda_k > 0$ then splitting $\boldsymbol{z}_j$ and $\boldsymbol{z}_k$ in the same way as above gives

$$0 = \overline{\boldsymbol{z}}_j^T\boldsymbol{z}_k = \overline{\left[\begin{array}{c} \boldsymbol{u}_j \\ \boldsymbol{v}_j \end{array}\right]}^T \left[\begin{array}{c} \boldsymbol{u}_k \\ \boldsymbol{v}_k \end{array}\right] = \overline{\left[\begin{array}{c} \boldsymbol{u}_j \\ \boldsymbol{v}_j \end{array}\right]}^T \left[\begin{array}{c} \boldsymbol{u}_k \\ -\boldsymbol{v}_k \end{array}\right].$$

That is, $0 = \overline{\boldsymbol{u}_j}^T\boldsymbol{u}_k + \overline{\boldsymbol{v}_j}^T\boldsymbol{v}_k = \overline{\boldsymbol{u}_j}^T\boldsymbol{u}_k - \overline{\boldsymbol{v}_j}^T\boldsymbol{v}_k$. Then $0 = \overline{\boldsymbol{u}_j}^T\boldsymbol{u}_k = \overline{\boldsymbol{v}_j}^T\boldsymbol{v}_k$. Thus the sets $\{\, \boldsymbol{u}_j \mid \lambda_j > 0 \,\}$ and $\{\, \boldsymbol{v}_j \mid \lambda_j > 0 \,\}$ are sets of orthonormal vectors. We also need to look at eigenvectors with eigenvalue zero:

$$\left[\begin{array}{c|c} & A \\ \hline \overline{A}^T & \end{array}\right]\left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right] = 0\left[\begin{array}{c} \boldsymbol{u} \\ \boldsymbol{v} \end{array}\right] = \left[\begin{array}{c} \boldsymbol{0} \\ \boldsymbol{0} \end{array}\right],$$

so $A\boldsymbol{v} = \boldsymbol{0}$ and $\overline{A}^T\boldsymbol{u} = \boldsymbol{0}$. The dimension of the set of all such vectors $[\boldsymbol{u}^T, \boldsymbol{v}^T]^T$ is the sum of the dimensions of the null spaces of $A$ and $\overline{A}^T$. So we can add a basis for the null space of $A$ for the $\boldsymbol{v}$'s and the null space of $\overline{A}^T$ for the $\boldsymbol{u}$'s. Any $\boldsymbol{u}$ in the

null space of $\overline{A}^T$ is orthogonal to $\boldsymbol{u}_k$ with $\lambda_k > 0$. Let $r$ be the number of positive eigenvalues of $C$. Then we set

$$U = [\boldsymbol{u}_1,\ \boldsymbol{u}_2,\ \ldots,\ \boldsymbol{u}_r, \boldsymbol{u}_{r+1}, \cdots,\ \boldsymbol{u}_m],$$
$$V = [\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \ldots,\ \boldsymbol{v}_r, \boldsymbol{v}_{r+1}, \cdots,\ \boldsymbol{v}_n],$$

where $\boldsymbol{u}_{r+1}, \ldots, \boldsymbol{u}_m$ form an orthonormal basis for null$(\overline{A}^T)$ and $\boldsymbol{v}_{r+1}, \ldots, \boldsymbol{v}_n$ form an orthonormal basis for null$(A)$. Then $A\boldsymbol{v}_j = \lambda_j \boldsymbol{u}_j$ and $\overline{A}^T \boldsymbol{u}_j = \lambda_j \boldsymbol{v}_j$ for all appropriate $j$. Let $\sigma_j = \lambda_j$ where $\lambda_j \geq 0$. Then

$$\overline{U}^T A \boldsymbol{v}_j = \lambda_j \boldsymbol{e}_j = \Sigma\, \boldsymbol{e}_j \qquad \text{for } j = 1, 2, \ldots, n,$$
$$\overline{V}^T \overline{A}^T \boldsymbol{u}_j = \lambda_j \boldsymbol{e}_j = \Sigma^T \boldsymbol{e}_j \qquad \text{for } j = 1, 2, \ldots, m.$$

That is, $\overline{U}^T A\, V = \Sigma$ and so $A = U\, \Sigma\, \overline{V}^T$ as we wanted. □

#### 2.5.4.2 Uses of the SVD

The pseudo-inverse of a matrix $A$ (2.2.4) can be generalized by defining the pseudo-inverse as

$$A^+ = V\, \Sigma^+\, \overline{U}^T, \tag{2.5.10}$$

where $\Sigma^+$ is the $n \times m$ diagonal matrix with $(\Sigma^+)_{jj} = 1/\sigma_j$ if $\sigma_j > 0$ and $(\Sigma^+)_{jj} = 0$ if $\sigma_j = 0$. With this definition, the solution of $\min_{\boldsymbol{x}} \|A\boldsymbol{x} - \boldsymbol{b}\|_2$ with the minimum value of $\|\boldsymbol{x}\|_2$ is $\boldsymbol{x} = A^+\boldsymbol{b}$.

The rank of $A$ is $\max\left\{j \mid \sigma_j > 0\right\}$. The 2-norm of $A$ is $\|A\|_2 = \sigma_1$. On the other hand, the Frobenius norm $\|A\|_F = \left[\sum_j \sigma_j^2\right]^{1/2}$. The singular values are invariant under multiplication by real orthogonal and complex unitary matrices: $\sigma_j(QA) = \sigma_j(A\widetilde{Q}) = \sigma_j(A)$ for real orthogonal or complex unitary matrices $Q$ and $\widetilde{Q}$.

The closest matrix to $A$ in the $\|\cdot\|_2$ norm that has rank $r < \text{rank}(A)$ is

$$A_r = [\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_r] \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \overline{[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_r]}^T. \tag{2.5.11}$$

This means that we look for low-rank approximations to a given matrix using the SVD.

### 2.5.4.3 Computing the SVD

As we have seen in the proof of the existence of the SVD (Theorem 2.30) and from $\overline{A}^T A = V(\Sigma^T \Sigma)\overline{V}^T$ that the SVD is closely related to the real symmetric or complex Hermitian eigenvalue problem. For convenience of analysis, we expand either the number of rows or number of columns with zero entries so that the matrix $A$ is square.

The basic QR algorithm can be applied to $C = \left[\begin{array}{c|} A \\ \hline \overline{A}^T \end{array}\right]$. Let $J = \left[\begin{array}{c|c} & I \\ \hline I & \end{array}\right]$ which swaps the block rows or columns and is a real orthogonal matrix. Assume that $m \geq n$. Then $JC = \left[\begin{array}{c|} \overline{A}^T \\ \hline A \end{array}\right]$. The QR factorization of $JC$ is given by

$$\left[\begin{array}{c|} \overline{P}^T \\ \hline \overline{Q}^T \end{array}\right] JC = \left[\begin{array}{c|} \overline{P}^T \overline{A}^T \\ \hline \overline{Q}^T A \end{array}\right] = \left[\begin{array}{c|} R \\ \hline S \end{array}\right]$$

with $P$ and $Q$ real orthogonal or complex Hermitian and $R$, $S$ upper triangular. Then the result of one step of the QR algorithm is

$$\begin{aligned} C' &= \left[\begin{array}{c|} R \\ \hline S \end{array}\right] \left[\begin{array}{c|c} & I \\ \hline I & \end{array}\right] \left[\begin{array}{c|} P \\ \hline Q \end{array}\right] = \left[\begin{array}{c|} RQ \\ \hline SP \end{array}\right] \\ &= \left[\begin{array}{c|} \overline{P}^T \overline{A}^T Q \\ \hline \overline{Q}^T A P \end{array}\right]. \end{aligned}$$

Note that $A = QS$ is the QR factorization of $A$ and $\overline{A}^T = PR$ is the QR factorization of $\overline{A}^T$. To accelerate each iteration of the QR algorithm, we transform the matrix $A$ into bidiagonal form.

For computing eigenvalues in the symmetric or Hermitian cases, we first reduce the matrix to a tridiagonal matrix. For the SVD, we can reduce the matrix to a bidiagonal form: we find real orthogonal or complex unitary $U_0$ and $V_0$ where

$$\overline{U_0}^T A V_0 = \begin{bmatrix} \alpha_1 & \beta_1 & & & & \\ & \alpha_2 & \beta_2 & & & \\ & & \alpha_3 & \ddots & & \\ & & & \ddots & \beta_{k-1} & \\ & & & & \alpha_k & \\ 0 & 0 & 0 & \cdots & 0 & \\ \vdots & \vdots & \vdots & & \vdots & \\ 0 & 0 & 0 & \cdots & 0 & \end{bmatrix} \quad \text{for } m > n. \tag{2.5.12}$$

We can do this by setting $A' \leftarrow P_1 A$ where $P_1$ is a Householder reflector where

$$P_1 \begin{bmatrix} a_{11} \\ a_{21} \\ \vdots \\ a_{m1} \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

For the next step, we find a new Householder reflector $P_2$ where

$$\overline{P_2} \begin{bmatrix} a_{12} \\ a_{13} \\ \vdots \\ a_{1n} \end{bmatrix} = \begin{bmatrix} * \\ 0 \\ \vdots \\ 0 \end{bmatrix},$$

and set $A'' \leftarrow A' \begin{bmatrix} 1 & \\ & \overline{P_2}^T \end{bmatrix}$. This did not change the first column of $A'$ and so keeps all entries of the first column except $a'_{11}$ equal to zero. Then

$$P_1 A \left[\begin{array}{c|c} 1 & \\ \hline & \overline{P_2}^T \end{array}\right] = \left[\begin{array}{c|c} a'_{11} & \begin{bmatrix} a''_{12} & \mathbf{0}^T \end{bmatrix} \\ \hline \mathbf{0} & \widetilde{A} \end{array}\right].$$

Recursively applying this approach to $\widetilde{A}$ to give $\overline{U_1}^T \widetilde{A} V_1 = \widetilde{B}$ is bidiagonal. Then

$$\left[\begin{array}{c|c} 1 & \\ \hline & \overline{U_1}^T \end{array}\right] P_1 A \left[\begin{array}{c|c} 1 & \\ \hline & \overline{P_2}^T V_1 \end{array}\right] = \left[\begin{array}{c|c} a'_{11} & \begin{bmatrix} a''_{12} & \mathbf{0}^T \end{bmatrix} \\ \hline \mathbf{0} & \widetilde{B} \end{array}\right].$$

Setting $U_0 = \overline{P_1}^T \left[\begin{array}{c|c} 1 & \\ \hline & U_1 \end{array}\right]$ and $V_0 = \left[\begin{array}{c|c} 1 & \\ \hline & \overline{P_2}^T V_1 \end{array}\right]$, we have $B = \overline{U_0}^T A V_0$ which is bidiagonal.

Once we have bidiagonal $B = \overline{U_0}^T A V_0$ we can develop chasing algorithms with shifts. We should simultaneously compute QR factorizations $B = QS$ and $\overline{B}^T = PR$. Since $B$ is an *upper* bidiagonal matrix it is already upper triangular and so we can set $Q = I$. But we still need to compute the QR factorization of $\overline{B}^T$. This can be done using a chasing algorithm that preserves the bidiagonal structure of $B$. At each step we apply a Givens' rotation either from the right or from the left. Let $G_j$ be the Givens' rotation applied from the left that is applied to rows $j$ and $j+1$, while $K_j$ is the Givens' rotation applied from the right that is applied to columns $j$ and $j+1$. The chasing algorithm is illustrated by Figure 2.5.8.

From the implicit $Q$ theorem (Theorem 2.29) once we have decided on $K_1$, the rest of the Givens rotations will specify the QR factorization in an essentially unique

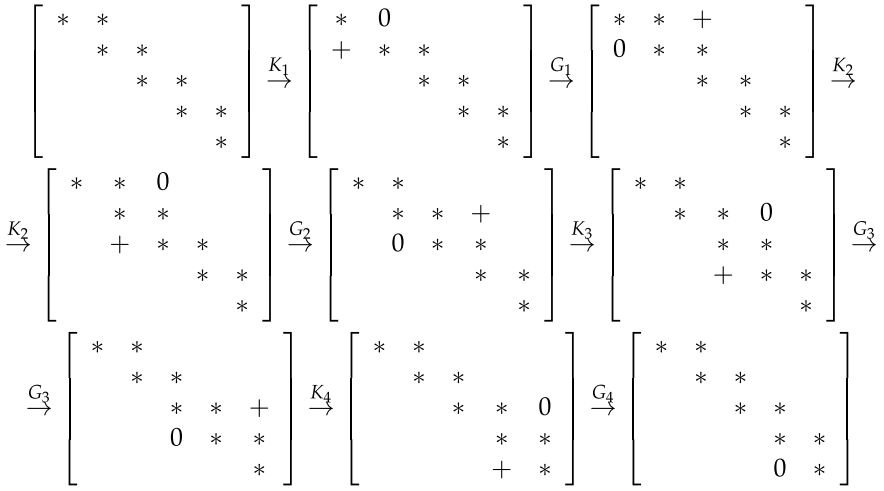


**Fig. 2.5.8** Chasing algorithm for bidiagonal matrix

way. The problem is, how to decide on what $K_1$ should be? To answer that, we need to understand how to do the shifting.

The singular values are the square roots of the eigenvalues of $B^T B$. The shift $\mu$ should be smallest eigenvalue of the bottom $2 \times 2$ submatrix of $B^T B$. Using form (2.5.12), this submatrix is

$$\left[\begin{array}{c|c} \alpha_{n-1}^2 + \beta_{n-2}^2 & \alpha_{n-1}\beta_{n-1} \\ \hline \alpha_{n-1}\beta_{n-1} & \alpha_n^2 + \beta_{n-1}^2 \end{array}\right].$$

The shift $\mu$ is subtracted from the diagonal of the top-left $2 \times 2$ submatrix:

$$\left[\begin{array}{c|c} \alpha_1^2 - \mu & \alpha_1\beta_1 \\ \hline \alpha_1\beta_1 & \alpha_2^2 + \beta_1^2 - \mu \end{array}\right].$$

We choose $K_1$ so that

$$\left[\alpha_1^2 - \mu,\ \alpha_1\beta_1\right] \left[\begin{array}{c|c} c_1 & s_1 \\ \hline -s_1 & c_1 \end{array}\right] = \left[*,\ 0\right].$$

Again, an accurate value of the shift mainly helps to increase the speed of convergence, but errors in $\mu$ do not directly cause substantial numerical errors in the solution. Combining all these elements together gives the Golub–Kahan algorithm for computing the SVD [104].

### 2.5.5 The Lanczos and Arnoldi Methods

Computing the eigenvalues of large matrices is a challenging problem. Usually only a few eigenvalues are desired. We have seen the power method used for computing the dominant eigenvalue for the Google PageRank algorithm (Section 2.5.1). Other problems may require the determination of multiple eigenvalues and their eigenvectors. For a real matrix $A$, a common approach is to find an orthonormal basis for a subspace forming the columns of $V_m$, and then computing the eigenvalues of $V_m^T A V_m$. The subspace used is typically a Krylov subspace. And so we will use the Arnoldi and Lanczos iterations (Algorithms 29, 30). If we want an approximate SVD of a large matrix, then Lanczos bidiagonalization (2.4.17) may be more appropriate.

#### 2.5.5.1 The Lanczos Method

The Lanczos iteration (Algorithm 30) for a symmetric $n \times n$ matrix $A$ gives a symmetric tridiagonal matrix $T_m$ and an $n \times m$ matrix of orthonormal columns $V_m$ where

$$A\, V_m = V_m T_m + \beta_m \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T.$$

Normally we compute $\boldsymbol{y}$ where $T_m \boldsymbol{y} = \lambda \boldsymbol{y}$ with $\boldsymbol{y} \neq \boldsymbol{0}$ and then our approximate eigenvector is $\boldsymbol{v} = V_m \boldsymbol{y}$. However, there is an error in $\boldsymbol{v}$:

$$\begin{aligned} A\boldsymbol{v} &= A V_m \boldsymbol{y} = V_m T_m \boldsymbol{y} + \beta_m \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T \boldsymbol{y} \\ &= \lambda V_m \boldsymbol{y} + \beta_m \boldsymbol{v}_{m+1} y_m = \lambda \boldsymbol{v} + \beta_m \boldsymbol{v}_{m+1} y_m. \end{aligned}$$

Then $(\boldsymbol{v}, \lambda)$ is the exact eigenpair of $A + E$ for a matrix $E$ with $\|E\|_2 = |\beta_m y_m|$. Since $A$ is symmetric, the eigenvalue $\lambda$ is in error by no more than $|\beta_m y_m|$. Thus we have an accurate eigenvector if $\beta_m \approx 0$ *or* $y_m \approx 0$. This is true even if the columns of $V_m$ are far from orthonormal.

Since the Lanczos iteration does not re-orthogonalize the columns of $V_m$, the columns will diverge from being orthonormal. With each iteration of Algorithm 30, the multiplication by $A$ and division by $\beta_j$ will tend to amplify any perturbations, so that roundoff error will slowly be amplified until distant columns of $V_m$ are far from being orthogonal to each other.

The other fact about the Lanczos method for eigenvalues is that it is much better at finding extreme eigenvalues than finding interior eigenvalues. The convergence theory is based on optimization and the Rayleigh quotient

$$R(z) = \frac{z^T A z}{z^T z}. \tag{2.5.13}$$

The maximum of $R(z)$ is the largest eigenvalue of $A$ provided $A$ is symmetric. For $A$ symmetric,

$$\nabla R(z) = 2\frac{Az(z^T z) - (z^T Az)z}{(z^T z)^2};$$

$\nabla R(z) = \mathbf{0}$ if and only if $z$ is an eigenvector of $A$. In this case, the eigenvalue for $z$ is $R(z)$. A maximum exists because we can restrict attention to all $z$ where $\|z\|_2 = 1$; this is a compact set so any continuous function on this set will have a maximum and a minimum. So the maximum value of $R(z)$ is the maximum eigenvalue of $A$.

Suppose the starting vector of the Lanczos iteration is $z_1$ with $\|z_1\|_2 = 1$, and that $A$ has eigenvalues $\lambda_1 > \lambda_2 \geq \lambda_3 \geq \cdots \geq \lambda_n$. Let $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n$ be an orthonormal basis of eigenvectors of $A$ where $\boldsymbol{v}_j$ has eigenvalue $\lambda_j$. Let $\phi_1$ be the angle between $\boldsymbol{v}_1$ and $z_1$: $\cos\phi_1 = \left|z_1^T \boldsymbol{v}_1\right|$. The main theorem regarding convergence is due to Kaniel and Paige [136, 196].

**Theorem 2.31** *(Lanczos convergence) Under the notation above, if $\theta_1$ is the maximum eigenvalue of $T_m$, then*

$$\lambda_1 \geq \theta_1 \geq \lambda_1 - \frac{(\lambda_1 - \lambda_n)\tan^2\phi_1}{T_{m-1}(1+2\rho)^2},$$

*where $T_{m-1}$ is the Chebyshev polynomial of degree $m-1$ (4.6.3) and $\rho = (\lambda_1 - \lambda_2)/(\lambda_2 - \lambda_n)$.*

For a proof, see [105, Sec. 9.1.4, pp. 475–479]. The basis of the proof is the fact that $\theta_1$ is the maximum eigenvalue of $T_{m-1} = V_{m-1}^T A\, V_{m-1}$ where $\text{range}(V_{m-1}) = \mathcal{K}_{m-1}(A, z_1)$ the Krylov subspace generated by $z_1$ of dimension $m$. Thus $\theta_1$ is the maximum of $R(z)$ over all $z = p(A)z_1$ where $p$ ranges over all polynomials of degree $\leq m-1$. If $z_1 = \sum_{j=1}^n c_j \boldsymbol{v}_j$, then $p(A)z_1 = \sum_{j=1}^n c_j\, p(\lambda_j)\boldsymbol{v}_j$. So

$$R(p(A)z_1) = \frac{\sum_{j=1}^n c_j^2\, \lambda_j\, p(\lambda_j)^2}{\sum_{j=1}^n c_j^2\, p(\lambda_j)^2}.$$

We can get a lower bound on this maximum by setting $p(\lambda) = T_{m-1}(-1 + 2(s - \lambda_n)/(\lambda_2 - \lambda_n))$ to emphasize $p(\lambda_1)$ compared with $p(\lambda)$ for any $\lambda \in [\lambda_n, \lambda_2]$. This gives the bounds in Theorem 2.31.

Details on how the Lanczos method behaves in practice and how to obtain accurate estimates of eigenvalues and eigenvectors can be found in Cullum and Willoughby [64, 65].

Because $T_{m-1}(1+2\rho) = \cosh((m-1)\cosh^{-1}(1+2\rho)) \approx \frac{1}{2}\exp((m-1)2\sqrt{\rho})$ for $\rho$ small but $(m-1)\sqrt{\rho}$ modest to large, convergence is fairly rapid. Convergence is fast enough that once a near-invariant subspace is achieved, we have $\beta_m \approx 0$. Small errors are amplified by division with $\beta_m$, and the method can be thought of as restarting with a random start vector. The method does not remember past vectors,

so the method again converges to the largest (and the smallest) eigenvalue of $A$. Consequently, the method gives repeated extreme eigenvalues, and even repeated near-extreme eigenvalues. The good news is that when the Lanczos method gives an apparent repeated eigenvalue, this eigenvalue is almost always accurate. One approach to improving the accuracy of the Lanczos method is to keep previous accurately computed eigenvectors and orthogonalize every new vector in the method against these accurate eigenvectors. This avoids the spurious multiplicity issue. The method is called *selective orthogonalization*.

While in exact arithmetic, the Arnoldi iteration for a real symmetric matrix $A$ is equivalent to the Lanczos iteration, it has an important practical difference: every new Arnoldi vector is orthogonalized against all the previous Arnoldi vectors. This is called *complete orthogonalization*. This prevents the bad numerical behavior of the Lanczos method, but at the cost of $\mathcal{O}(m^2 n)$ operations for $m$ Arnoldi steps with $A$ an $n \times n$ matrix, compared with $\mathcal{O}(mn)$ operations for $m$ Lanczos steps.

The Arnoldi iteration produces a matrix of orthonormal columns $V_m$ and a Hessenberg matrix $H_m$ where

$$A\, V_m = V_m\, H_m + h_{m+1,m} \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T.$$

To compute approximate eigenvalues and eigenvectors of $A$, we find an eigenvector of $H_m$: $H_m \boldsymbol{y} = \lambda \boldsymbol{y}$ with $\|\boldsymbol{y}\|_2 = 1$ and use $\boldsymbol{v} = V_m \boldsymbol{y}$ as our approximate eigenvector. Then

$$A\boldsymbol{v} = \lambda \boldsymbol{v} + h_{m+1,m} y_m \boldsymbol{v}_{m+1}$$

so that $\boldsymbol{v}$ is the exact eigenvector of $A + E$ with eigenvalue $\lambda$ where $\|E\|_2 = \left|h_{m+1,m} y_m\right|$.

The cost of complete orthogonalization can be reduced by restarting the method. The variant of Lehoucq and Sorenson [160] provides an implicitly restarted method. This method captures the information in $[V_m,\ \boldsymbol{v}_{m+1}]$, $H_m$, and $h_{m+1,m}$ to give a subset of vectors so that the method can be effectively restarted. They have a package called ARPACK that implements this method which is an excellent method for computing eigenvalues and eigenvectors of large matrices.

**Exercises.**

(1) The matrix

$$A = \begin{bmatrix} 5 & 3 & 0 \\ -2 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

has eigenvalues 1, 2, and 3. Apply the power method with starting vector $\boldsymbol{x}_0 = [1,\ 1,\ 1]^T$. Plot $\|A\boldsymbol{x}_k - \lambda_k \boldsymbol{x}_k\|_2 \,/\, \|\boldsymbol{x}_k\|_2$ against the iteration count $k$.

(2) Show that the $n \times n$ matrix

$$A = \begin{bmatrix} 1 & 1 & & & & \\ & 1 & 2 & & & \\ & & 1 & 3 & & \\ & & & \ddots & \ddots & \\ & & & & 1 & (n-1) \\ & & & & & 1 \end{bmatrix}$$

has one eigenvalue $\lambda = 1$ repeated $n$ times. Use a general eigenvalue solver to compute the eigenvalues of this matrix for $n = 5,\ 10,\ 15$. Symbolically compute the eigenvalues of $A + \epsilon \boldsymbol{e}_n \boldsymbol{e}_1^T$. Numerically compute these eigenvalues for $\epsilon = 10^{-10}$ and $n = 5,\ 10,\ 15$.

(3) Show the *Wielandt–Hoffman bound* [105, Thm. 8.1.4]: If $A, E$ are real symmetric $n \times n$ matrices, and $\boldsymbol{\lambda}(B) \in \mathbb{R}^n$ is the vector of eigenvalues of $B$ in decreasing order (counting multiplicities) then

$$\|\boldsymbol{\lambda}(A+E) - \boldsymbol{\lambda}(A)\|_2 \leq \|E\|_F\,. \tag{2.5.14}$$

[**Hint:** Let $A + tE = Q(t)^T D(t) Q(t)$ where $Q(t)$ is orthogonal and $D(t)$ diagonal. Supposing both $D(t)$ and $Q(t)$ are differentiable in $t$, show that $dD/dt(t)$ is the diagonal part of $Q(t)\ E\ Q(t)^T$. From this, show

$$D(1) - D(0) = \int_0^1 \operatorname{diag}(Q(t)\ E\ Q(t)^T)\, dt;$$

taking Frobenius norms, show (2.5.14).]

(4) Show that the eigenvalues of the $n \times n$ matrix

$$A_n = \begin{bmatrix} 2 & -1 & & & & \\ -1 & 2 & -1 & & & \\ & -1 & 2 & \ddots & & \\ & & \ddots & \ddots & -1 & \\ & & & -1 & 2 & -1 \\ & & & & -1 & 2 \end{bmatrix}$$

are $\lambda_j = 2(1 - \cos(\pi j/(n+1)))$ for $j = 1, 2, \ldots, n$. [**Hint:** If $A\boldsymbol{x} = \lambda \boldsymbol{x}$ then for $1 < k < n$, $2x_k - x_{k-1} - x_{k+1} = \lambda x_k$ which gives linear recurrence relation for the $x_k$'s. Put $x_k = a_1 r_1^k + a_2 r_2^k$ where $r_1$ and $r_2$ are solutions of the characteristic equation $2r - 1 - r^2 = \lambda r$. The boundary conditions can be represented by setting $x_0 = 0$ and $x_{n+1} = 0$.] Use a suitable numerical method to compute the eigenvalues of $A_n$ for $n = 5,\ 10,\ 15,\ 20$, and compare with the exact values you have computed.

(5) Check the Wielandt–Hoffman bound numerically: start with $A_n$ from Exercise 4 for $n = 10$. Create a perturbation matrix $E_n$ that is symmetric, either by hand, or by first setting each entry to be $s$ times a random sample of a standard normal distribution, and then setting $E_n \leftarrow (E_n + E_n^T)/2$ to ensure symmetry. First use $s = 0.1$. Compute the vectors of eigenvalues $\boldsymbol{\lambda}(A_n)$ and $\boldsymbol{\lambda}(A_n + E_n)$, each with the eigenvalues sorted in increasing order. (What is important is that they are sorted *consistently*.) Then check if $\|\boldsymbol{\lambda}(A_n + E_n) - \boldsymbol{\lambda}(A_n)\|_2 \leq \|E_n\|_F$. Repeat for $s = 10^{-2}$ and $s = 1$.

(6) The Schur decomposition $A = \overline{Q}^T T\, Q$ with $T$ upper triangular and $Q$ unitary is essentially unique if the diagonal entries of $T$ (the eigenvalues of $A$) are distinct and in identical order. Show this by noting that if $A = \overline{Q_1}^T T_1\, Q_1 = \overline{Q_2}^T T_2\, Q_2$ we have $T_1\, Q_1 \overline{Q_2}^T = Q_1 \overline{Q_2}^T\, T_2$. Set $U = Q_1 \overline{Q_2}^T$, which is unitary. Show that if $T\, U = U\, S$ where $T$ and $S$ are upper triangular with $\mathrm{diag}(T) = \mathrm{diag}(S)$ and distinct diagonal entries and $U$ unitary, then $U$ is a diagonal matrix with diagonal entries that are complex numbers of size one. Do this by induction on $n$ where $T$ and $S$ are $n \times n$. For the induction step let

$$T = \left[\begin{array}{c|c} \tau & \boldsymbol{w}^T \\ \hline & \widehat{T} \end{array}\right], \qquad S = \left[\begin{array}{c|c} \sigma & \boldsymbol{r}^T \\ \hline & \widehat{S} \end{array}\right], \qquad U = \left[\begin{array}{c|c} \psi & \boldsymbol{z}^T \\ \hline \boldsymbol{u} & \widehat{U} \end{array}\right].$$

Note that $\tau = \sigma$ because $\mathrm{diag}(T) = \mathrm{diag}(S)$. Show that equating $T\, U = U\, S$ implies $\boldsymbol{u} = \boldsymbol{0}$, $\boldsymbol{z} = \boldsymbol{0}$, and $\widehat{T}\widehat{U} = \widehat{U}\widehat{S}$.

(7) The Schur decomposition can also be used to solve *Sylvester equations*: find an $m \times n$ matrix $X$ where $AX + XB = C$ for given matrices $A$ ($m \times m$), $B$ ($n \times n$), and $C$ ($m \times n$). Since these are linear equations, they can be solved using an $mn \times mn$ matrix, which would take $\mathcal{O}(m^3 n^3)$ floating point operations and use $\mathcal{O}(m^2 n^2)$ memory. Instead we can use the Schur decompositions $A = \overline{Q}^T R\, Q$ and $B = \overline{U}^T S\, U$ where $Q$ and $U$ are unitary, and $R$, $S$ are upper triangular. Create a recursive algorithm based on the splittings

$$R = \left[\begin{array}{c|c} \rho & \boldsymbol{r}^T \\ \hline & \widehat{R} \end{array}\right], \qquad S = \left[\begin{array}{c|c} \sigma & \boldsymbol{s}^T \\ \hline & \widehat{S} \end{array}\right], \qquad Y = \left[\, \boldsymbol{y},\ \widehat{Y} \,\right],$$

where $Y = Q\, X\, \overline{U}^T$. The algorithm should take $\mathcal{O}(m^3 + mn(m+n))$ operations once the Schur decompositions have been computed. Show that the method you create succeeds provided $A$ and $-B$ have no eigenvalues in common.

(8) The Lanczos iteration in exact arithmetic generates a tridiagonal matrix $T_m$ and a matrix $V_m$ or orthonormal columns where

$$A\, V_m = V_m T_m + \beta_m \boldsymbol{v}_{m+1} \boldsymbol{e}_m^T.$$

Implement the Lanczos method or use an implementation of it applied to the matrix $A_h$ from Exercise 1. Use $h = 1/10$ as a concrete value. For applying the

Lanczos algorithm use $m = 40$ and a random start vector. You may notice that the smallest and largest eigenvalues of $T_m$ are repeated. Compute $V_m^T V_m$. How much does it deviate from $I$? Even if $V_m$ is not close to having orthonormal columns, as long as the columns are linearly independent, show that if $T_m \boldsymbol{z} = \lambda \boldsymbol{z}$ and $z_m = 0$ then $V_m \boldsymbol{z}$ is an eigenvector of $A$. Use this to argue that repeated eigenvalues of $T_m$ are eigenvalues of $A$.

(9) The *Perron–Frobenius theorem* says that if $A$ is a square matrix of non-negative entries then there is an eigenvalue $\lambda_1 \geq 0$ (positive unless $A = 0$) which has an eigenvector $\boldsymbol{v}_1$ ($A\boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1$, $\boldsymbol{v}_1 \neq \boldsymbol{0}$) with non-negative entries. Furthermore, *every* eigenvalue $\lambda$ of $A$ satisfies $|\lambda| \leq \lambda_1$. Prove that the existence of $\lambda_1$ and $\boldsymbol{v}_1$ implies $|\lambda| \leq \lambda_1$ for every eigenvalue of $A$.

# Chapter 3
# Solving nonlinear equations


Unlike solving linear equations, solving nonlinear equations cannot be done by a "direct" method except in special circumstances, such as solving a quadratic equation in one unknown. We therefore focus on iterative methods. From the numerical point of view, there are a number of issues that arise:

- How rapidly do the methods converge?
- Can the methods fail, and if so, do they fail gracefully?
- What is the trade-off between computational cost and accuracy?

We normally discuss the asymptotic speed of convergence as this is the easiest to understand theoretically and apply to practice. However, if our initial "guess" is far from the true solution, it may take many iterations to begin to approach a true solution, even if the method is asymptotically fast.

Methods can fail by falling into infinite loops far from a solution, failing to converge, or resulting in impossible computations such as division by zero or taking the square root of a negative number in real arithmetic. After all, not all equations have solutions. How should a numerical method handle such problems? We would like our methods to be *robust* (able to fail gracefully for a difficult or impossible problem) and *reliable* (able to provide results even for difficult problems), as well as being accurate and efficient.

Many of the methods we present here are applicable only to single equations in a single, scalar, unknown. Others can be applied to problems of arbitrarily high dimension, but with a corresponding loss of guarantees. Which method should be used depends, at least in part, on the circumstances.

## 3.1 Bisection method

The bisection method is the best method we know of for reliability and robustness.

We start with a theorem, well known from calculus and basic analysis:



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_3

**Algorithm 40** Bisection method

```
1  function bisect(f, a, b, ε)
2    if sign f(a) ≠ −sign f(b)
3      return fail
4    end if
5    while |b − a| > ε
6      m ← (a + b)/2
7      if f(m) = 0: return m; end if
8      if sign f(m) = sign f(a)
9        a ← m
10     else // sign f(m) = sign f(b)
11       b ← m
12     end if
13   end while
14   return (a + b)/2
15 end function
```

**Theorem 3.1** *(Intermediate Value Theorem) Suppose that* $f : [a, b] \to \mathbb{R}$ *is continuous and* $y$ *is a real number between* $f(a)$ *and* $f(b)$*. Then there is a* $c \in [a, b]$ *where* $f(c) = y$*.*

The special case with $y = 0$ gives the fact that if $f(a)$ and $f(b)$ have opposite signs, then there must be a $c$ between $a$ and $b$ where $f(c) = 0$. This theorem does not give a way of *computing* the value of a solution $c$. The bisection algorithm remedies this problem. In fact, the bisection algorithm can give a constructive proof of intermediate value theorem. Pseudo-code for the bisection algorithm is given in Algorithm 40.

**Example 3.2** As an example, take $f(x) = x\,e^x - 3$. Then $f(1) = 1 \times e^1 - 3 = e - 3 < 0$ since $e = 2.718\ldots$. On the other hand, $f(2) = 2 \times e^2 - 3 > 2 \times 2^2 - 3 = 5 > 0$. Thus we can take $a = 1$ and $b = 2$. Then applying the algorithm gives the results in Table 3.1.1.

### *3.1.1 Convergence*

To study the convergence of this method we need to introduce some notation: let $a_k$ and $b_k$ be the values of $a$ and $b$ at the start of the `while` loop (line 5) after completing $k$ passes through the body of the while loop. The initial values of $a$ and $b$ are then $a_0$ and $b_0$.

The most important facts for establishing convergence are that $|b_{k+1} - a_{k+1}| = \frac{1}{2}|b_k - a_k|$ and assuming $a_0 < b_0$ we have $a_k \le a_{k+1} < b_{k+1} \le b_k$ for $k = 0, 1, 2, \ldots$. Note that if $b_0 < a_0$ then corresponding inequalities still hold, but in the

**Table 3.1.1** Results of bisection for $f(x) = x\,e^x - 3$ starting with $[a, b] = [1, 2]$

| $k$ | $a_k$ | $f(a_k)$ | $b_k$ | $f(b_k)$ |
|---|---|---|---|---|
| 0 | 1.0 | −0.28171817 | 2.0 | 11.7781121978 |
| 1 | 1.0 | −0.28171817 | 1.5 | +3.7225336055 |
| 2 | 1.0 | −0.28171817 | 1.25 | +1.3629286968 |
| 3 | 1.0 | −0.28171817 | 1.125 | +0.4652439550 |
| 4 | 1.0 | −0.28171817 | 1.0625 | +0.0744456906 |
| 5 | 1.03125 | −0.10778785 | 1.0625 | +0.0744456906 |
| 6 | 1.046875 | −0.01773065 | 1.0625 | +0.0744456906 |
| 7 | 1.046875 | −0.01773065 | 1.0546875 | +0.0280898691 |
| 8 | 1.046875 | −0.01773065 | 1.05078125 | +0.0051130416 |
| 9 | 1.0488281 | −0.00632540 | 1.05078125 | +0.0051130416 |
| 10 | 1.04980469 | −0.00061033 | 1.05078125 | +0.0051130416 |

reverse direction: $b_k \le b_{k+1} < a_{k+1} \le a_k$ for $k = 0, 1, 2, \ldots$. These facts can be readily established from the code in Algorithm 40.

Mathematical induction can then be used to show that, provided $a_0 < b_0$,

$$|b_k - a_k| = 2^{-k}\,|b_0 - a_0|\,, \qquad \text{and}$$
$$a_0 \le a_k \le a_{k+1} < b_{k+1} \le b_k \le b_0$$

for $k = 0, 1, 2, \ldots$. From this, we can see that the sequence $a_k$ is a non-decreasing sequence that is bounded above by $b_0$, and that $b_k$ is a non-increasing sequence that is bounded below by $a_0$. Since bounded monotone sequences converge, $\lim_{k\to\infty} a_k = \widehat{a}$ and $\lim_{k\to\infty} b_k = \widehat{b}$ for some $\widehat{a}$ and $\widehat{b}$. On the other hand,

$$\left|\widehat{b} - \widehat{a}\right| = \lim_{k\to\infty} |b_k - a_k| = \lim_{k\to\infty} 2^{-k}\,|b_0 - a_0| = 0,$$

so $\widehat{a} = \widehat{b}$. Now $\operatorname{sign} f(a_k) = \operatorname{sign} f(a_0) = -\operatorname{sign} f(b_0) = -\operatorname{sign} f(b_k)$ for $k = 0$, $1, 2, \ldots$. If $f(a_0) > 0$ then $f(b_0) < 0$ and so $f(a_k) > 0 > f(b_k)$ for all $k$. Taking the limit as $k \to \infty$ we use continuity to see that $f(\widehat{a}) \ge 0 \ge f(\widehat{b}) = f(\widehat{a})$ and so $f(\widehat{a}) = f(\widehat{b}) = 0$ and $c = \widehat{a} = \widehat{b}$ is the solution we are looking for. We can treat the case where $f(a_0) < 0 < f(b_0)$ in the same way, by reversing the directions of the inequalities.

### *3.1.2 Robustness and reliability*

The convergence results of the previous section do not depend on the function given to the *bisect* function. This makes the bisection method a very *reliable* one. Convergence only requires that $f$ is continuous on $[a, b]$.

The bisection method is also *robust* to failures in our assumptions. For example, even if $f$ is not continuous, the method can still converge, although not necessarily to a solution. A specific case is $f(x) = \tan x$ on the interval $[1, 2]$. Then the bisection method converges: $a_k, b_k \to \pi/2$ as $k \to \infty$. Now $\pi/2$ does not satisfy $\tan(\pi/2) = 0$. The problem is that $\tan(\pi/2) = \sin(\pi/2)/\cos(\pi/2)$ and $\cos(\pi/2) = 0$ so $\tan(\pi/2)$ is undefined. However, if $x \approx \pi/2$ and $x < \pi/2$ then $\tan(x) > 0$; if $x \approx \pi/2$ and $x > \pi/2$ then $\tan(x) < 0$.

The point is that even in the few extreme cases that the bisection method fails, it does so gracefully. There is one unavoidable limitation with the bisection method: it can only be applied to solving a single scalar equation in a single real variable. The fundamental reason is that there is no easy generalization of the intermediate value theorem to two or more dimensions.

While the bisection method is extremely robust and reliable, the same cannot be said about most other algorithms for solving equations.

**Exercises.**

(1) Solve the equation $x\,e^x = 4$ to an accuracy of $10^{-6}$ using the bisection algorithm on $f(x) = x\,e^x - 4$.
(2) Solve the equation $e^x = 2 + x^2$ to an accuracy of $10^{-6}$ by using the bisection algorithm on $f(x) = e^x - (2 + x^2)$ with $a = 0$ and $b = 2$.
(3) Solve the equation of the previous question to the same accuracy by using the bisection algorithm on $g(x) = x - \ln(2 + x^2)$ and $a = 0$ and $b = 2$. Compare the two computed solutions. Why are they the same?
(4) The equation $\tan x = x$ has infinitely many solutions ($x = 0$ is one of them), but your task is to find the one closest to $3\pi/2$. But beware! The solution is quite close to $3\pi/2$, and $x = 3\pi/2$ is a singularity of $\tan x$. Report an estimate of the solution with an error of no more than $10^{-6}$.
(5) Solve the equation $f(x) = 0$ where $f(x) = x^5 - 10x^4 + 40x^3 - 80x^2 + 80x - 32$ with starting interval $[a, b] = [1.3, 3.2]$ to an accuracy of $10^{-15}$. Compare this with solving $g(x)$ where $g(x) = (x - 2)^5$. Explain the difference in the results, given that $f(x)$ is simply $g(x)$ expanded symbolically.
(6) The stopping criterion in the bisection method is "$|b - a| < \epsilon$". For most other algorithms the stopping criterion is "$|f(x)| < \epsilon$". Show that if $f$ is smooth and $\epsilon$ is small, then these two stopping criteria are within a factor of approximately $\left|f'(x^*)\right|$.
(7) Apply the bisection method to solve $\cos(1/x) = 0$ on the interval $[10^{-3}, 1]$. There are infinitely many solutions of $\cos(1/x) = 0$ in the interval $(0, 1)$. Which one is chosen?
(8) Why can't the bisection method be generalized to solving two equations in two variables?

## 3.2 Fixed-point iteration

Fixed-point iteration is a simple algorithm:

```
for k = 0, 1, 2, ...
  x_{k+1} ← g(x_k)
end for
```

If we consider "$x_k$" not be just a single number, but rather something more complex, fixed-point iterations represent a very wide range of computational processes. For example, if $\boldsymbol{x}_k \in \mathbb{R}^n$, an $n$-dimensional real vector, then the iteration $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{g}(\boldsymbol{x}_k)$ for $k = 0, 1, 2, 3, \ldots$ represents a large number of computational processes.

If $\boldsymbol{g}$ is a continuous function $\mathbb{R}^n \to \mathbb{R}^n$ then if the $\boldsymbol{x}_k$'s converge to a limit $\widehat{\boldsymbol{x}}$, then this limit is a *fixed point* of $\boldsymbol{g}$:

$$\widehat{\boldsymbol{x}} = \lim_{k\to\infty} \boldsymbol{x}_{k+1} = \lim_{k\to\infty} \boldsymbol{g}(\boldsymbol{x}_k) = \boldsymbol{g}(\lim_{k\to\infty} \boldsymbol{x}_k) = \boldsymbol{g}(\widehat{\boldsymbol{x}}).$$

That is, $\widehat{\boldsymbol{x}}$ is a fixed point of $\boldsymbol{g}$. There are well-known conditions under which convergence is assured.

**Theorem 3.3** *(Contraction mapping theorem) Suppose* $\boldsymbol{g}\colon \mathbb{R}^n \to \mathbb{R}^n$ *is a* contraction mapping *(that is, there is a constant* $L < 1$ *where* $\|\boldsymbol{g}(\boldsymbol{u}) - \boldsymbol{g}(\boldsymbol{v})\| \le L\,\|\boldsymbol{u} - \boldsymbol{v}\|$ *for all* $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$*), then the iterates* $\boldsymbol{x}_k$ *of* $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{g}(\boldsymbol{x}_k)$ *for* $k = 0, 1, 2, \ldots$ *converge to the unique fixed point* $\widehat{\boldsymbol{x}}$*:* $\boldsymbol{g}(\widehat{\boldsymbol{x}}) = \widehat{\boldsymbol{x}}$.

Contraction maps, then, are very useful for fixed-point iterations. But contraction maps are hard to find.

Consider the problem of solving $x\,e^x = 3$. We can re-write this equation as

$$\begin{aligned} x &= 3\,e^{-x}, \qquad \text{or} \\ x &= \ln(3/x) = \ln 3 - \ln x. \end{aligned}$$

The corresponding fixed-point iterations are

$$\begin{aligned} x_{k+1}^{(1)} &= 3\,e^{-x_k^{(1)}}, \qquad \text{or} \\ x_{k+1}^{(2)} &= \ln 3 - \ln x_k^{(2)}. \end{aligned}$$

We let

$$\begin{aligned} g_1(x) &= 3\,e^{-x}, \qquad \text{and} \\ g_2(x) &= \ln 3 - \ln x, \end{aligned}$$

so we have two fixed-point iterations

**Table 3.2.1** Results for two fixed-point iterations

| $k$ | $x_k^{(1)}$ | $f(x_k^{(1)})$ | $x_k^{(2)}$ | $f(x_k^{(2)})$ |
|---|---|---|---|---|
| 0 | 1.05029297 | $+2.250 \times 10^{-3}$ | 1.05029297 | $+2.250 \times 10^{-3}$ |
| 1 | 1.04950572 | $-2.361 \times 10^{-3}$ | 1.04954314 | $-2.141 \times 10^{-3}$ |
| 2 | 1.05033227 | $+2.480 \times 10^{-3}$ | 1.05025732 | $+2.041 \times 10^{-3}$ |
| 3 | 1.04946449 | $-2.602 \times 10^{-3}$ | 1.04957709 | $-1.943 \times 10^{-3}$ |
| 4 | 1.05037559 | $+2.735 \times 10^{-3}$ | 1.05022498 | $+1.852 \times 10^{-3}$ |
| 5 | 1.04941902 | $-2.868 \times 10^{-3}$ | 1.04960788 | $-1.763 \times 10^{-3}$ |
| 6 | 1.05042334 | $+3.014 \times 10^{-3}$ | 1.05019564 | $+1.680 \times 10^{-3}$ |
| 7 | 1.04936891 | $-3.162 \times 10^{-3}$ | 1.04963581 | $-1.599 \times 10^{-3}$ |
| 8 | 1.05047598 | $+3.323 \times 10^{-3}$ | 1.05016902 | $+1.524 \times 10^{-3}$ |
| 9 | 1.04931368 | $-3.485 \times 10^{-3}$ | 1.04966116 | $-1.451 \times 10^{-3}$ |
| 10 | 1.05053401 | $+3.663 \times 10^{-3}$ | 1.05014488 | $+1.382 \times 10^{-3}$ |

$$x_{k+1}^{(1)} \leftarrow g_1(x_k^{(1)}), \qquad \text{and}$$
$$x_{k+1}^{(2)} \leftarrow g_2(x_k^{(2)}).$$

We will start with the solution estimate provided by the bisection method after 10 steps (1.05029297) and use $f(x) = x\,e^x - 3$ to estimate how close the solution estimate is to the exact solution: if $x \approx x^*$ where $f(x^*) = 0$, then $f(x) \approx f(x^*) + f'(x^*)(x - x^*) = f'(x^*)(x - x^*)$. Using $x^* \approx 1.05$ gives $f'(x^*) \approx 5.86$, so $f(x)$ is approximately 5.86 times the error, provided the error is small. The results are shown in Table 3.2.1.

It appears from Table 3.2.1 that the error for $x_k^{(1)}$ is growing in size, but slowly, while the error for $x_k^{(2)}$ is shrinking in size, but slowly. If this trend continues, then $x_k^{(1)}$ will *not* converge to the solution $x^*$, while $x_k^{(2)}$ will. Why the difference? This is the topic of the next section.

### *3.2.1 Convergence*

If we consider a scalar fixed-point iteration

$$x_{k+1} \leftarrow g(x_k) \qquad \text{for } k = 0, 1, 2, \ldots,$$

we suppose that $x_k \to x^*$ as $k \to \infty$ where $g(x^*) = x^*$ is a fixed point. From the mean value theorem, there must be a $c_k$ between $x^*$ and $x_k$ where $g(x_k) - g(x^*) = g'(c_k)(x_k - x^*)$. So

$$x_{k+1} - x^* = g(x_k) - x^* = g(x_k) - g(x^*) = g'(c_k)(x_k - x^*).$$

Writing $e_j = x_j - x^*$, which is the error in $x_j$, we see that

$$e_{k+1} = g'(c_k)\, e_k.$$

If $x_k \to x^*$ as $k \to \infty$, then by the squeeze theorem, $c_k \to x^*$ as well. Provided $g'$ is continuous, we see that

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k} = \lim_{k\to\infty} g'(c_k) = g'(x^*).$$

**Theorem 3.4** *(Convergence of fixed-point iterations) Consider the fixed-point iteration $x_{k+1} \leftarrow g(x_k)$ for $k = 0, 1, 2, 3, \ldots$ where $g$ is differentiable and $g(x^*) = x^*$. Then if $\left|g'(x^*)\right| < 1$ there is a $\delta > 0$ where $x_k \to x^*$ as $k \to \infty$ provided $|x_0 - x^*| < \delta$; if $\left|g'(x^*)\right| > 1$ then* ***either*** *$x_k = x^*$ exactly for some $k$* ***or*** *$x_k \not\to x^*$.*

Before we start with the proof, it should be noted that the existence of a $\delta > 0$ where convergence is guaranteed for all $x_0$ within $\delta$ of $x^*$ can be colloquially described as "*convergence is guaranteed for all $x_0$ sufficiently close to $x^*$*". How close "*sufficiently close*" is, of course, depends on the iteration function $g$.

In the alternative case where $\left|g'(x^*)\right| > 1$, there is the possibility that at some point in the iteration, we have $x_k = x^*$ *exactly*. This is wildly improbable, but it is possible to contrive instances where it occurs. For example, if $g(x) = x^2$ then $x_0 = -1$ will give $x_1 = +1$, which is a fixed point of $g$. Any other starting point other than $x_0 = x^*$ will result in either divergence of the sequence $x_k$ from the fixed-point iteration or convergence to the other fixed point (which is zero).

***Proof*** (Of Theorem 3.4.) Suppose that $\left|g'(x^*)\right| < 1$. Let $\epsilon = \frac{1}{2}(1 - \left|g'(x^*)\right|) > 0$. Then by continuity of $g'$, there is a $\delta > 0$ where $|x - x^*| \le \delta$ implies $\left|g'(x) - g'(x^*)\right| \le \epsilon$. Let $L = \frac{1}{2}(1 + \left|g'(x^*)\right|)$ which is less than one. Thus if $|x - x^*| \le \delta$,

$$\left|g'(x)\right| \le \left|g'(x^*)\right| + \epsilon = \left|g'(x^*)\right| + \frac{1}{2}(1 - \left|g'(x^*)\right|) = L < 1.$$

So if $|x - x^*| \le \delta$,

$$\begin{aligned} \left|g(x) - x^*\right| = \left|g(x) - g(x^*)\right| &= \left|g'(c)(x - x^*)\right| \qquad \text{for some } c \text{ between } x \text{ and } x^* \\ &= \left|g'(c)\right| \left|x - x^*\right| \le L \left|x - x^*\right| < \left|x - x^*\right| \le \delta. \end{aligned}$$

Therefore, $g$ maps the interval $[x^* - \delta,\ x^* + \delta]$ onto itself. In particular, since $|x_0 - x^*| \le \delta$, we have $|x_k - x^*| \le \delta$ for $k = 0, 1, 2, 3, \ldots$. Furthermore, for any $x^* - \delta \le x_k \le x^* + \delta$,

$$\left|x_{k+1} - x^*\right| = \left|g(x_k) - g(x^*)\right| = \left|g'(c)(x_k - x^*)\right| = \left|g'(c)\right| \left|x_k - x^*\right|$$

for some $c$ between $x_k$ and $x^*$; therefore, $c$ is in $[x^* - \delta,\ x^* + \delta]$ and so $\left|g'(c)\right| \leq L$. Thus $|x_{k+1} - x^*| \leq L\,|x_k - x^*|$ for $k$=0, 1, 2, .... Then $|x_k - x^*| \leq L^k\,|x_0 - x^*| \to 0$ as $k \to \infty$. That is, $x_k \to x^*$ as $k \to \infty$.

*In the alternative case where* $\left|g'(x^*)\right| > 1$, suppose that $x_k \neq x^*$ for any $k$. Then supposing that $x_k \to x^*$ as $k \to \infty$ would imply that

$$\lim_{k\to\infty} \frac{|x_{k+1} - x^*|}{|x_k - x^*|} = \left|g'(x^*)\right| > 1$$

using the arguments immediately preceding the statement of the theorem. That is, there is a $K$ where $k \geq K$ implies that

$$\frac{|x_{k+1} - x^*|}{|x_k - x^*|} \geq 1.$$

Therefore, $|x_k - x^*| \geq |x_K - x^*| > 0$ for all $k \geq K$, which contradicts the assumption that $x_k \to x^*$ as $k \to \infty$. Thus, the only way in which $x_k \to x^*$ as $k \to \infty$ in this alternative case is if $x_\ell = x^*$ exactly for some $\ell$, and thus $x_j = x^*$ for all $j \geq \ell$. □

**Example 3.5** As a practical application of this result, we consider the iterations $x_{k+1}^{(1)} \leftarrow g_1(x_k^{(1)})$ and $x_{k+1}^{(2)} \leftarrow g_1(x_k^{(2)})$ given in the previous section. The fixed point of interest is $x^* \approx 1.05$, so convergence (or divergence) is determined by $\left|g'(x^*)\right|$. For the first iteration, $\left|g_1'(x^*)\right| \approx \left|g_1'(1.05)\right| \approx |-1.05| = 1.05 > 1$ while $\left|g_2'(x^*)\right| \approx \left|g_2'(1.05)\right| = |-1/1.05| \approx 0.95 < 1$. Thus, the first iteration roughly increases the error by 5%, while the second iteration roughly decreases the error by 5%, with each iteration.

Note that smaller values of $\left|g'(x^*)\right|$ result in faster asymptotic convergence. The optimal case of $\left|g'(x^*)\right| = 0$ is actually of more than theoretical importance as we will discuss later in connection with Newton's method.

### *3.2.2 Robustness and reliability*

We have seen two iteration functions for solving the same problem: one gave convergence, the other divergence. Clearly more needs to be understood about the iteration for it to converge. Furthermore, even if the convergence condition $\left|g'(x^*)\right| < 1$ holds, convergence is only guaranteed if "$x_0$ is sufficiently close to $x^*$". That is, convergence is only known to hold *locally*, around the exact solution.

Worse than this, general fixed-point iterations can result in extremely bad behavior. Take, for example, $g(x) = x^2$. The fixed points are $x = 0$ and $x = 1$. But if we start at $x_0 = 2$ we get $x_1 = 2^2, x_2 = (2^2)^2 = 2^4, x_3 = (2^4)^2 = 2^8$, and, in general, $x_k = 2^{(2^k)}$, which grows very rapidly in size.

In another example, the iteration $x_{k+1} = 4x_k(1 - x_k)$ is known to have all iterates $x_k \in [0, 1]$ if $x_0 \in [0, 1]$, but the behavior of these iterates can be extremely hard to predict. In fact, this is a standard example in nonlinear dynamics of a "chaotic" system. To see better how it behaves, write $x_k = \sin^2 \theta_k$. Then

$$\begin{aligned} x_{k+1} &= 4x_k(1 - x_k) = 4 \sin^2 \theta_k (1 - \sin^2 \theta_k) \\ &= 4 \sin^2 \theta_k \cos^2 \theta_k = (2 \sin \theta_k \cos \theta_k)^2 = \sin^2(2\theta_k). \end{aligned}$$

Writing $\theta_{k+1} = 2\theta_k \mod \pi$ gives an exact solution which, nevertheless, exposes the "chaotic" nature of its dynamics: The behavior of the iterates depends on the binary expansion of $\theta_0/\pi$; each iteration shifts this binary expansion one place left and zeros out the bit(s) before the "binary point". The two fixed points are $x = 0$ and $x = 3/4$; in terms of $\theta$ they are $\theta = 0$ and $\theta = 2\pi/3$. Both fixed points are unstable. Typical behavior of the iterates $x_k$ is aperiodic and does not converge to fixed point or periodic orbit.

The range of possible behavior of the iterates of fixed-point iterations is enormous, especially for multivariate fixed-point iterations, which are covered in the following section. We will see later that this family of methods can be extremely fast as well as frustratingly slow.

### 3.2.3 *Multivariate fixed-point iterations*

Consider iterations of the form

$$\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{g}(\boldsymbol{x}_k) \qquad \text{for } k = 0, 1, 2, \ldots \tag{3.2.1}$$

with $\boldsymbol{x}_k \in \mathbb{R}^n$. We again assume that $\boldsymbol{g}$ is differentiable and that there is a designated exact fixed point $\boldsymbol{x}^* = \boldsymbol{g}(\boldsymbol{x}^*)$. We use a multivariate version of the mean value theorem:

$$\boldsymbol{g}(\boldsymbol{v}) - \boldsymbol{g}(\boldsymbol{u}) = \int_0^1 \nabla \boldsymbol{g}(\boldsymbol{u} + s(\boldsymbol{v} - \boldsymbol{u}))\,(\boldsymbol{v} - \boldsymbol{u})\, ds. \tag{3.2.2}$$

We assume that we are using compatible matrix and vector norms, both represented by $\|\cdot\|$.

**Theorem 3.6** *If $\nabla \boldsymbol{g}$ is continuous and $\|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\| < 1$ then there is a $\delta > 0$ where $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| \le \delta$ implies the iterates $\boldsymbol{x}_k$ of (3.2.1) converge to $\boldsymbol{x}^*$.*

***Proof*** Let $\epsilon = (1 - \|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\|)/2$. Let $L = (1 + \|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\|)/2 < 1$. By continuity of $\nabla \boldsymbol{g}$, there is a $\delta > 0$ where $\|\boldsymbol{x} - \boldsymbol{x}^*\| \le \delta$ implies $\|\nabla \boldsymbol{g}(\boldsymbol{x}) - \nabla \boldsymbol{g}(\boldsymbol{x}^*)\| \le \epsilon$. We define the closed ball $\overline{B(\boldsymbol{x}^*, \delta)} = \{\, \boldsymbol{x} \mid \|\boldsymbol{x} - \boldsymbol{x}^*\| \le \delta \,\}$. Then if $\boldsymbol{x} \in \overline{B(\boldsymbol{x}^*, \delta)}$ we have $\|\nabla \boldsymbol{g}(\boldsymbol{x})\| \le \|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\| + \epsilon = L < 1$ and

$$\begin{aligned}\left\|\boldsymbol{g}(\boldsymbol{x})-\boldsymbol{x}^*\right\| &= \left\|\boldsymbol{g}(\boldsymbol{x})-\boldsymbol{g}(\boldsymbol{x}^*)\right\| = \left\|\int_0^1 \nabla \boldsymbol{g}(\boldsymbol{x}+s(\boldsymbol{x}^*-\boldsymbol{x}))\,(\boldsymbol{x}-\boldsymbol{x}^*)\,ds\right\| \\ &\le \int_0^1 \left\|\nabla \boldsymbol{g}(\boldsymbol{x}+s(\boldsymbol{x}^*-\boldsymbol{x}))\,(\boldsymbol{x}^*-\boldsymbol{x})\right\| ds \\ &\le \int_0^1 \left\|\nabla \boldsymbol{g}(\boldsymbol{x}+s(\boldsymbol{x}^*-\boldsymbol{x}))\right\| \left\|\boldsymbol{x}^*-\boldsymbol{x}\right\| ds \\ &\le \int_0^1 L\left\|\boldsymbol{x}^*-\boldsymbol{x}\right\| ds = L\left\|\boldsymbol{x}^*-\boldsymbol{x}\right\| \le L\,\delta < \delta,\end{aligned}$$

and so $\boldsymbol{g}(\boldsymbol{x}) \in \overline{B(\boldsymbol{x}^*,\delta)}$. First, we see that $\boldsymbol{g}$ maps $\overline{B(\boldsymbol{x}^*,\delta)} \to \overline{B(\boldsymbol{x}^*,\delta)}$. Next we see that if $\boldsymbol{x}_k \in \overline{B(\boldsymbol{x}^*,\delta)}$ then $\|\boldsymbol{x}_{k+1}-\boldsymbol{x}^*\| \le L\,\|\boldsymbol{x}_k-\boldsymbol{x}^*\|$. Therefore, provided $\boldsymbol{x}_0 \in \overline{B(\boldsymbol{x}^*,\delta)}$, $\|\boldsymbol{x}_k-\boldsymbol{x}^*\| \le L^k\,\|\boldsymbol{x}_0-\boldsymbol{x}^*\| \to 0$ as $k\to\infty$, and so $\boldsymbol{x}_k \to \boldsymbol{x}^*$ as $k\to\infty$. □

The alternative case that $\|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\| > 1$ does *not* imply a lack of convergence. If $\boldsymbol{g}$ is linear, then $\nabla \boldsymbol{g}$ is constant, and the question of convergence of the fixed-point iteration amounts to asking if $A^k \to 0$ as $k\to\infty$ where $A = \nabla \boldsymbol{g}(\boldsymbol{x}^*)$. We know from the properties of matrix norms that $\left\|A^k\right\| \le \|A\|^k$, but there is no reverse inequality. In fact, we can have $A^2 = 0$ with $\|A\| > 1$, such as $A = \begin{bmatrix} 0 & 2 \\ 0 & 0 \end{bmatrix}$ for which $\|A\|_\infty = \|A\|_2 = 2$.

The question of which matrices $A$ satisfy $A^k \to 0$ as $k\to\infty$ is answered by the *spectral radius* (2.4.5):

$$\rho(A) = \max\{\,|\lambda| : \lambda \text{ is an eigenvalue of } A\,\}.$$

For a square matrix $A$, Theorem 2.15 shows that

$$A^k \to 0 \text{ as } k\to\infty \text{ if and only if } \rho(A) < 1.$$

It should be noted that for any induced matrix norm, $\|A\| \ge \rho(A)$. To see why this is so, suppose $\boldsymbol{v}$ is an eigenvector of $A$ with eigenvalue $\lambda$: $A\boldsymbol{v} = \lambda\boldsymbol{v}$ with $\boldsymbol{v} \ne \boldsymbol{0}$. Then $\|A\|\,\|\boldsymbol{v}\| \ge \|A\boldsymbol{v}\| = \|\lambda\boldsymbol{v}\| = |\lambda|\,\|\boldsymbol{v}\|$ and dividing by $\|\boldsymbol{v}\| > 0$ gives $\|A\| \ge |\lambda|$ for every eigenvalue $\lambda$. Taking the maximum over all eigenvalues gives $\|A\| \ge \rho(A)$ as desired.

Using Theorem 2.15 with Theorem 3.6 we can show convergence for a fixed-point iteration provided $\rho(\nabla \boldsymbol{g}(\boldsymbol{x}^*)) < 1$.

**Example 3.7** Here is an example of a fixed-point iteration to solve

$$\begin{aligned} x^3 - x^2 + 2y + 3 &= 0, \\ e^y + xy + x - x^2 - 10 &= 0. \end{aligned}$$

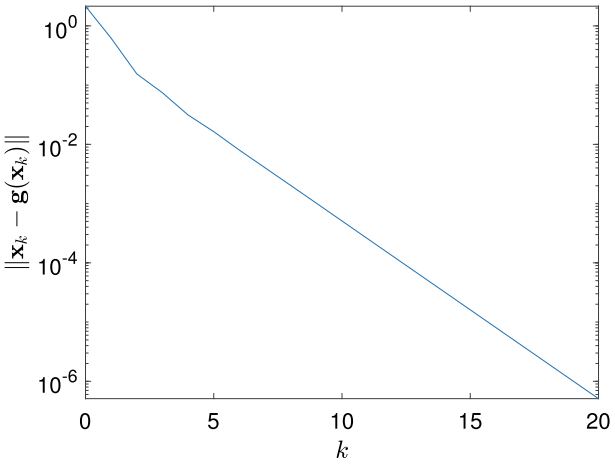



**Fig. 3.2.1** Convergence of multivariate fixed-point iteration example

The iteration function is

$$\boldsymbol{g}\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} (x^2 - 2y - 3)^{1/3} \\ \ln(10 + x^2 - x - xy) \end{bmatrix}.$$

With starting value $\boldsymbol{x}_0 = [x_0, y_0]^T = [0,\ 1]^T$, the values of $\|\boldsymbol{x}_k - \boldsymbol{g}(\boldsymbol{x}_k)\|_2$ are shown in Figure 3.2.1. Geometric convergence is apparent from the figure. Furthermore, we can compute

$$\begin{aligned} \nabla \boldsymbol{g}(\boldsymbol{x}^*) &\approx \left[\begin{array}{c|c} -0.370925 & -0.206379 \\ \hline -0.372222 & +0.087879 \end{array}\right], \\ \left\|\nabla \boldsymbol{g}(\boldsymbol{x}^*)\right\|_2 &\approx 0.5713577, \qquad \text{while} \\ \rho(\nabla \boldsymbol{g}(\boldsymbol{x}^*)) &\approx 0.5013064. \end{aligned}$$

Estimating the slope in Figure 3.2.1 gives a reduction by a factor of $\approx 0.49686$ per iteration, which is quite close to the spectral radius $\rho(\nabla \boldsymbol{g}(\boldsymbol{x}^*)) \approx 0.5013064$.

**Exercises.**

(1) Consider the equation $e^x + x^2 = 4$. We look for positive solutions. Show, using the fact that $(d/dx)(e^x + x^2) > 0$ for $x > 0$, that there is only one positive solution. Consider the two iteration schemes

$$\begin{aligned} x_{n+1} &= \sqrt{4 - e^{x_n}}, \\ x_{n+1} &= \ln(4 - x_n^2). \end{aligned}$$

Which of these methods converges near to the solution? What is the rate of convergence?

(2) The contraction mapping theorem (Theorem 3.3) implies that if $f : \mathbb{R} \to \mathbb{R}$ is differentiable and $\left|f'(x)\right| \leq L$ for all $x$, with $L < 1$, then $f$ has a unique fixed point. However, show that the function $f(x) = x + e^{-x}$ maps $[0, \infty) \to [0, \infty)$ has $f'(x) < 1$ for all $x$, and yet has no fixed point. Explain the paradox.

(3) Suppose that $\boldsymbol{f} : \mathbb{R}^n \to \mathbb{R}^n$ has the property that $\nabla \boldsymbol{f}(\boldsymbol{x})$ is positive definite (but not necessarily symmetric!) with $\lambda_{\min}(\nabla \boldsymbol{f}(\boldsymbol{x}) + \nabla \boldsymbol{f}(\boldsymbol{x})^T) \geq \alpha$ for all $\boldsymbol{x}$ with $\alpha > 0$, and bounded ($\|\nabla \boldsymbol{f}(\boldsymbol{x})\|_2 \leq B$ for all $\boldsymbol{x} \in \mathbb{R}^n$), then show that $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x} - (\alpha/(2B^2))\boldsymbol{f}(\boldsymbol{x})$ is a contraction mapping. [**Hint:** Show that $\|\nabla \boldsymbol{g}(\boldsymbol{x})\|_2^2 \leq 1 - (\alpha/(2B))^2$. You can use $\|\nabla \boldsymbol{g}(\boldsymbol{x})z\|_2^2 = z^T \nabla \boldsymbol{g}(\boldsymbol{x})^T \nabla \boldsymbol{g}(\boldsymbol{x}) z$ and expand $\nabla \boldsymbol{g}(\boldsymbol{x}) = I - (\alpha/(2B^2))\nabla \boldsymbol{f}(\boldsymbol{x})$.]

(4) Fixed-point iterations can be very slowly convergent if $\left|g'(x^*)\right| = 1$. Take, for example, $x_{n+1} = \sin x_n$. Starting from $x_0 = 1$, note that all $x_n > 0$. Plot $x_n$ against $n$ on a log-log plot. Estimate $C$ and $\alpha$ where $x_n \sim C\, n^{-\alpha}$ as $n \to \infty$. Use the fact that $\sin x \approx x - x^3/6$ to explain these values of $C$ and $\alpha$. [**Hint:** Match $C(n+1)^{-\alpha} \sim C\, n^{-\alpha} - (Cn^{-\alpha})^3/6$ as $n \to \infty$.]

(5) Solve the simultaneous equations below using the obvious fixed-point iteration. Perform 20 iterations of fixed-point iteration.

$$x = \ln(1 + x^2 + y^2) - y/2 + 1,$$
$$y = \exp(-x) + x^2 - y - 1.$$

What is the solution $\boldsymbol{x}^*$ you obtain? What is the Jacobian matrix for the iteration function $\boldsymbol{g}$ at $\boldsymbol{x}^*$? What convergence rate is predicted by the Jacobian matrix? What is the convergence rate you obtain empirically? Are the two rates consistent?

(6) A method of accelerating a fixed-point iteration $x_{n+1} = g(x_n)$ is to assume that $x_n \approx x^* + a\, r^n$ for large $n$. Using $x_k \approx x^* + a\, r^k$ for $k = n-1,\ n,\ n+1$ solve the approximate equations for $a$, $r$, and $x^*$ from $x_{n-1}$, $x_n$, and $x_{n+1}$. The value of $x^*$ obtained from this is not expected to be the exact solution, but rather a better approximation to it. [**Hint:** Use $(x_{n+1} - x_n)/(x_n - x_{n-1})$ to estimate $r$.]

(7) Apply the acceleration method of Exercise 6 to the convergent iteration in Exercise 1. How quickly do the accelerated iterates converge?

(8) A vector version of the acceleration method of Exercise 6 is to suppose that $\boldsymbol{x}_k \approx \boldsymbol{x}^* + \boldsymbol{a}r^k$. Estimate $r$ using $r \approx (\boldsymbol{x}_{n+1} - \boldsymbol{x}_n)^T(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})/(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})^T(\boldsymbol{x}_n - \boldsymbol{x}_{n-1})$. Develop formulas for the estimates of $\boldsymbol{a}$ and $\boldsymbol{x}^*$. Use it to accelerate the fixed point in Exercise 5. What rate of convergence does the accelerated estimates of $\boldsymbol{x}^*$ have?

(9) Another idea of accelerating fixed-point iterations is to suppose that if $\boldsymbol{x}^*$ is the exact fixed point, then $\boldsymbol{x}_{n+1} - \boldsymbol{x}^* = \boldsymbol{g}(\boldsymbol{x}_n) - \boldsymbol{g}(\boldsymbol{x}^*) \approx A(\boldsymbol{x}_n - \boldsymbol{x}^*)$. If $A$ has been estimated, then show that $\boldsymbol{x}_{n+1} - A\boldsymbol{x}_n \approx (I - A)\boldsymbol{x}^*$ and so a better estimate of $\boldsymbol{x}^*$ can be computed.

(10) If

$$A = \begin{bmatrix} 0.9 & 2 & & & \\ & 0.9 & 2 & & \\ & & \ddots & \ddots & \\ & & & 0.9 & 2 \\ & & & & 0.9 \end{bmatrix} \quad (20 \times 20),$$

compute $\|A^n\|_2$ for $n = 1, 2, \ldots, 50$, and plot these values against $n$. Check that $A^n \to 0$ exponentially fast as $n \to \infty$. Explain why we seem to need large $n$ for this to become apparent.

## 3.3 Newton's method

Newton's method is a method for solving equations that is sometimes taught as part of an introductory calculus course. Nevertheless, it is a powerful and general tool that finds many applications.

A way of deriving Newton's method in one variable is to use the tangent line through a point on the graph to obtain a new estimate of the solution $x^*$ of $f(x) = 0$, as illustrated in Figure 3.3.1.

In Figure 3.3.1, the initial guess $x_0$ is used to create the tangent line which crosses the $x$-axis at $x_1$. Then the tangent line at $x_1$ gives the next guess $x_2$, which in turn leads to the next guess $x_3$, which is already much closer to the solution $x^*$ where $f(x^*) = 0$.

To derive the method in a more formal way, we note that if $x \approx x_0$ then $f(x) \approx f(x_0) + f'(x_0)(x - x_0)$. Instead of trying to directly solve $f(x) = 0$, we

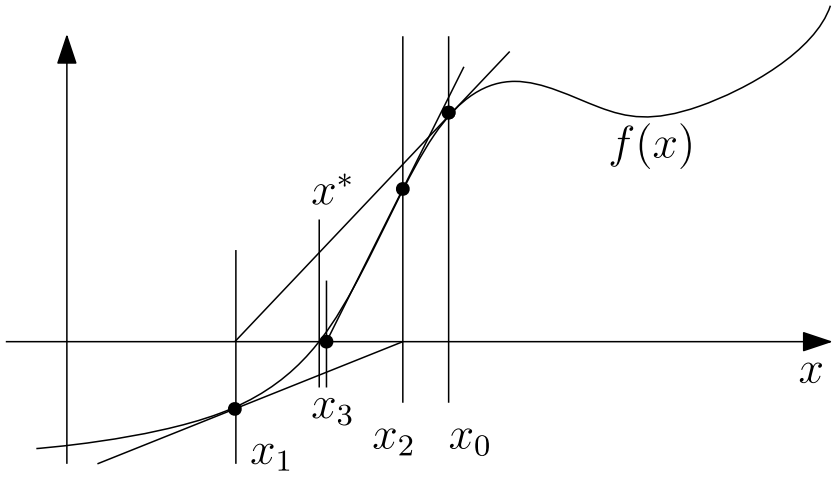



**Fig. 3.3.1** Newton's method

**Algorithm 41** Newton's method

```
1 function newton(f, f', x_0, ε)
2   k ← 0
3   while |f(x_k)| > ε
4     x_{k+1} ← x_k − f(x_k)/f'(x_k)
5     k ← k + 1
6   end while
7   return x_k
8 end function
```

solve $f(x_0) + f'(x_0)(x - x_0) = 0$ we obtain

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

Using this as the next guess $x_1$, and repeating the process gives Algorithm 41.

Newton's method is a particular kind of fixed-point iteration $x_{k+1} \leftarrow g(x_k)$ where $g(x) = x - f(x)/f'(x)$.

### 3.3.1 Convergence of Newton's method

To get an empirical example of how Newton's method converges, consider the problem of solving $e^x - x^2 - 3 = 0$. We apply Newton's method with initial guess $x_0 = 1$. The results are shown in Table 3.3.1.

Table 3.3.1 shows the typical rapid convergence of Newton's method, once the iterates start approaching the true solution. The final entry shows $f(x_7)$ is a small integer multiple of unit roundoff. This means that the actual value of $f(x_7)$ is very close to zero. Also note that the *exponents* of the values $f(x_k)$ roughly double with each iteration for $k \geq 3$. This is indicative of *quadratic convergence*: $|e_{k+1}| \leq C\ |e_k|^2$.

**Table 3.3.1** Newton's method applied to $e^x - x^2 - 3 = 0$

| $k$ | $x_k$ | $f(x_k)$ |
|---|---|---|
| 0 | 1.000000000000000 | $-1.2817 \times 10^{0}$ |
| 1 | 2.784422382354665 | $+5.4375 \times 10^{0}$ |
| 2 | 2.272498962410127 | $+1.5394 \times 10^{0}$ |
| 3 | 1.974092118993297 | $+3.0304 \times 10^{-1}$ |
| 4 | 1.880903342392723 | $+2.1630 \times 10^{-2}$ |
| 5 | 1.873171697204092 | $+1.3577 \times 10^{-4}$ |
| 6 | 1.873122549684362 | $+5.4455 \times 10^{-9}$ |
| 7 | 1.873122547713043 | $-4.4409 \times 10^{-16}$ |

**Theorem 3.8** *(Convergence of Newton's method) Suppose that $f$ has a continuous second derivative, $f(x^*) = 0$, and $f'(x^*) \neq 0$. Then there is a $\delta > 0$ where $|x_0 - x^*| < \delta$ implies that $x_k \to x^*$ and $(x_{k+1} - x^*)/(x_k - x^*)^2$ converges to $f''(x^*)/(2\, f'(x^*))$ as $k \to \infty$.*

***Proof*** Newton's method is

$$x_{k+1} = x_k - f(x_k)/f'(x_k). \tag{3.3.1}$$

Using Taylor series with remainder, and $e_k = x_k - x^*$, we see that

$$\begin{aligned} f(x_k) &= f(x^*) + f'(x^*)\, e_k + \frac{1}{2} f''(c_k)\, e_k^2, \\ f'(x_k) &= f'(x^*) + f''(d_k)\, e_k, \end{aligned}$$

where $c_k$ and $d_k$ are between $x_k$ and $x^*$. Subtracting $x^*$ from both sides of (3.3.1) gives

$$\begin{aligned} e_{k+1} &= e_k - \frac{f(x_k)}{f'(x_k)} \\ &= e_k - \frac{f(x^*) + f'(x^*)e_k + \frac{1}{2} f''(c_k)\, e_k^2}{f'(x^*) + f''(d_k)\, e_k} \\ &= e_k - \frac{(f'(x^*) + \frac{1}{2} f''(c_k)\, e_k)\, e_k}{f'(x^*) + f''(d_k)\, e_k} \\ &= e_k \left[ 1 - \frac{f'(x^*) + \frac{1}{2} f''(c_k)\, e_k}{f'(x^*) + f''(d_k)\, e_k} \right] \\ &= e_k \frac{\left[ f''(d_k) - \frac{1}{2} f''(c_k) \right] e_k}{f'(x^*) + f''(d_k)\, e_k} \\ &= e_k^2 \frac{f''(d_k) - \frac{1}{2} f''(c_k)}{f'(x^*) + f''(d_k)\, e_k}. \end{aligned}$$

Choose $C > \left| \frac{1}{2} f''(x^*)/f'(x^*) \right|$. By continuity of $f''$, there is a $\delta_1 > 0$ such that

$$\left| \frac{f''(v) - \frac{1}{2} f''(u)}{f'(x^*) + f''(v)\,(w - x^*)} \right| \leq C \qquad \text{for all } u, v, w \in [x^* - \delta_1, x^* + \delta_1].$$

Then provided $|e_k| \leq \delta_1$, we have $|e_{k+1}| \leq C\, |e_k|^2 = (C\, |e_k|)\, |e_k|$. If we also have $C\, |e_k| \leq \frac{1}{2}$ then $|e_{k+1}| \leq \frac{1}{2}\, |e_k|$. Let $\delta_2 = 1/(2C) > 0$. Then provided $|e_k| \leq \min(\delta_1, \delta_2)$, we $|e_{k+1}| \leq C\, |e_k|^2 \leq \frac{1}{2}\, |e_k|$. This in turn implies that $|e_{k+1}| \leq |e_k| \leq \min(\delta_1, \delta_2)$; from this we can see that $e_k \to 0$ as $k \to \infty$ provided $|e_0| \leq \min(\delta_1, \delta_2)$.

**Table 3.3.2** Quadratic convergence

| $k$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $C\,\lvert e_k\rvert$ | $5 \times 10^{-1}$ | $2.5 \times 10^{-1}$ | $6.25 \times 10^{-2}$ | $3.91 \times 10^{-3}$ | $1.53 \times 10^{-5}$ |
| $k$ | 5 | 6 | 7 | 8 | 9 |
| $C\,\lvert e_k\rvert$ | $2.33 \times 10^{-10}$ | $5.42 \times 10^{-20}$ | $2.94 \times 10^{-39}$ | $8.64 \times 10^{-78}$ | $7.46 \times 10^{-155}$ |

Now that convergence has been clearly established,

$$\frac{e_{k+1}}{e_k^2} = \frac{f''(d_k) - \frac{1}{2}f''(c_k)}{f'(x^*) + f''(d_k)\,e_k} \to \frac{\frac{1}{2}f''(x^*)}{f'(x^*)} \qquad \text{as } k \to \infty,$$

since $c_k, d_k \to x^*$ and $e_k \to 0$ as $k \to \infty$. □

To give a better idea, just how fast this rate of convergence is, suppose that $|e_{k+1}| \leq C\,|e_k|^2$. Multiplying by $C$ gives $C\,|e_{k+1}| \leq (C\,|e_k|)^2$. If $C\,|e_0| \leq 1/2$, then Table 3.3.2 shows how quickly $C\,|e_k|$ goes to zero.

### 3.3.2 *Reliability of Newton's method*

A major problem is getting a starting guess $x_0$ that is "close enough". How close is "close enough" depends very much on the problem. And, yes, Newton's method can fail to converge. As a simple example, consider the equation $\tan^{-1} x = 0$. The solution is easy: $x = \tan 0 = 0$. But let's try using Newton's method:

$$x_{n+1} = x_n - \frac{\tan^{-1} x_n}{(1/(1+x_n^2))} = x_n - (1+x_n^2)\tan^{-1} x_n.$$

Convergence, or failure to converge, depends on the starting point. Table 3.3.3 shows what happens with two different starting points.

The precise point where convergence gives way to divergence can be precisely computed for this problem, although this is not the case with most problems.

Another case where convergence may be (somewhat) in doubt is the case where $f'(x^*) = 0$. This is a rarer case, since it requires exact equality, but cases where $f'(x^*)$ is small may cause numerical difficulties even if asymptotic quadratic convergence is

**Table 3.3.3** Newton's method applied to $\tan^{-1} x = 0$

| $k$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $x_k$ | 1 | $-5.707 \times 10^{-1}$ | $1.169 \times 10^{-1}$ | $-1.061 \times 10^{-3}$ | $7.963 \times 10^{-10}$ |
| $x_k$ | 2 | $-3.536 \times 10^{0}$ | $1.395 \times 10^{+1}$ | $-2.793 \times 10^{+2}$ | $1.220 \times 10^{+5}$ |

assured. To understand the typical situation where $f'(x^*) = 0$, assume that $f(x) = (x - x^*)^p\, g(x)$ where $p > 1$ and $g(x^*) \neq 0$. Then $f'(x) = p(x - x^*)^{p-1} g(x) + (x - x^*)^p\, g'(x)$. Newton's method then becomes

$$\begin{aligned} x_{n+1} &= x_n - \frac{f(x_n)}{f'(x_n)} \\ &= x_n - \frac{(x_n - x^*)^p\, g(x_n)}{p(x_n - x^*)^{p-1}\, g(x_n) + (x_n - x^*)^p\, g'(x_n)} \\ &= x_n - \frac{(x_n - x^*)\, g(x_n)}{p\, g(x_n) + (x_n - x^*)\, g'(x_n)}. \end{aligned}$$

Subtracting $x^*$ from both sides, and writing $e_n = x_n - x^*$,

$$\begin{aligned} e_{n+1} &= e_n - \frac{e_n\, g(x_n)}{p\, g(x_n) + e_n\, g'(x_n)} \\ &= e_n \left[ 1 - \frac{1}{p + e_n (g'(x_n)/g(x_n))} \right] \end{aligned}$$

so $e_{n+1}/e_n \to 1 - (1/p) \neq 0$ if $e_n \to 0$ as $n \to \infty$. That is, Newton's method in these circumstances shows *linear convergence*, not quadratic convergence. Nevertheless, it still converges provided $x_0$ is, again, "sufficiently close" to $x^*$.

The main reliability issue is then about starting points $x_0$ not being "sufficiently close". Starting far from the solution $x^*$ (assuming there is only one) means that the derivative $f'(x_0)$ does not necessarily give useful information about the correct direction to move in. There are going to be limits on what algorithms can achieve. After all, not all equations have solutions. In the next section, we look at one way of improving the reliability of Newton's method.

### 3.3.3 *Variant: Guarded Newton method*

The failure of the Newton method in the case of solving $\tan^{-1} x = 0$ with $x_0 = 2$ can be located in the fact that the linear approximation $f(x) \approx f(x_0) + f'(x_0)\,(x - x_0)$ is not a good one if $x - x_0$ is large, and $x_1 - x_0 = -f(x_0)/f'(x_0)$ is large. The step we take in the Newton method, $d_k = -f(x_k)/f'(x_k)$, can be very large. When $d_k$ is large, we should take a step that is fraction of this: $x_{k+1} = x_k + s\, d_k$ for some $0 < s \leq 1$.

Where possible, we should use $s = 1$, because then we are taking a full Newton step which gives quadratic convergence. But we need some way of checking the step to see if it significantly improves our approximate solution. Of course, if an equation does not have a solution (or if the algorithm does not find one) then we want the algorithm to fail gracefully. Ideally, the method should be able to identify when the method cannot make significant progress. Simply having *some* improvement

**Table 3.3.4** Example of failure of guarded Newton method with an any decrease strategy

| $k$ | $x_k$ | $f(x_k)$ |
|---|---|---|
| 0 | +3.162015182 | +6.303807760 |
| 1 | −3.079829414 | −6.220782816 |
| 2 | +3.069566528 | +6.210397393 |
| 3 | −3.065443184 | −6.206222764 |
| 4 | +3.063612597 | +6.204369002 |
| 5 | −3.062768079 | −6.203513705 |
| 6 | +3.062371889 | +6.203112440 |

($|f(x_{k+1})| < |f(x_k)|$ for all $k$) is not sufficient to drive $|f(x_k)|$ to a local minimum. An example of this is $f(x) = 2x + \alpha \sin(x)$ where $\alpha = -1/(\cos(\widehat{x}) - \sin(\widehat{x})/(2\widehat{x})) \approx 0.990279$, $\widehat{x} = \pi - 1/(4\pi)$, and $x_0 = \widehat{x} + 0.1$. The unique solution to the equation $f(x^*) = 0$ is $x^* = 0$. The iterates and their function values are shown in Table 3.3.4, which demonstrates $|f(x_k)|$ decreasing in $k$ with limit $|f(\widehat{x})| \neq 0$. This kind of failure is rare, but can occur.

Instead we will require a "sufficient decrease" in the function values that is achievable. Since

$$\begin{aligned} \frac{d}{ds} f(x_k + s d_k)|_{s=0} &= f'(x_k)\, d_k \qquad \text{and} \\ d_k &= -f(x_k)/f'(x_k) \qquad \text{so} \\ \frac{d}{ds} f(x_k + s d_k)|_{s=0} &= f'(x_k) \left( \frac{-f(x_k)}{f'(x_k)} \right) = -f(x_k). \end{aligned}$$

Therefore, using Taylor series with second-order remainder, we have

$$\begin{aligned} f(x_k + s\, d_k) &= f(x_k) + \frac{d}{ds} f(x_k + s d_k)|_{s=0} \cdot s + \frac{1}{2} \frac{d^2}{ds^2} f(x_k + s d_k)|_{s=c_s} \cdot s^2 \\ &= f(x_k) - s\, f(x_k) + \frac{1}{2} f''(x_k + c_s d_k)\, d_k^2 s^2 \\ &= (1 - s)\, f(x_k) + \frac{1}{2} f''(x_k + c_s d_k)\, (s d_k)^2 \end{aligned}$$

for some $c_s$ between 0 and $s$. We can operationalize the idea of "significant improvement that is achievable" by requiring that

$$|f(x_k + s d_k)| \leq (1 - \frac{1}{2} s)\, |f(x_k)|\,. \tag{3.3.2}$$

This is achievable because for sufficiently small $s > 0$ we have

**Algorithm 42** Guarded Newton method

```
1  function gnewton(f, f', x_0, ε, η)
2    k ← 0
3    while |f(x_k)| > ε
4      d_k ← −f(x_k)/f'(x_k)
5      s ← 1
6      while |f(x_k + s d_k)| > (1 − ½ s) |f(x_k)|
7        s ← s/2
8        if s < η then return x_k
9      end while
10     x_{k+1} ← x_k + s d_k
11     k ← k + 1
12   end while
13   return x_k
14 end function
```

**Table 3.3.5** Guarded Newton method near a local minimizer of $|f(x)|$

| $k$ | $x_k$ | $d_k$ | $s_k$ | $f(x_k)$ |
|---|---|---|---|---|
| 0 | 0.500000000000000 | $-1.3621 \times 10^{0}$ | $2^{-2}$ | 1.23437500 |
| 1 | 0.159482758620690 | $-3.3123 \times 10^{0}$ | $2^{-5}$ | 1.02492769 |
| 2 | 0.055972268830070 | $-9.0558 \times 10^{0}$ | $2^{-8}$ | 1.00311098 |
| 3 | 0.020597954605748 | $-2.4379 \times 10^{1}$ | $2^{-11}$ | 1.00042318 |
| 4 | 0.008694301949811 | $-5.7607 \times 10^{1}$ | $2^{-13}$ | 1.00007551 |
| 5 | 0.001662175199375 | $-3.0091 \times 10^{2}$ | $2^{-18}$ | 1.00000276 |
| 6 | 0.000514312730217 | $-9.7227 \times 10^{2}$ | $2^{-21}$ | 1.00000026 |

$$|f(x_k + sd_k)| \leq (1 - s)\,|f(x_k)| + M(sd_k)^2,$$

where $M$ is a bound on $f''$. As long as

$$(1 - s)\,|f(x_k)| + M(sd_k)^2 \leq (1 - \frac{1}{2}s)\,|f(x_k)|\,,$$

the sufficient decrease criterion (3.3.2) is satisfied. Algorithm 42 shows how to implement a guarded Newton method.

This guarded Newton method can still fail: take, for example, $f(x) = 1 + x^2 - (x/2)^3$ and $x_0 = 1/2$. Since we require that $|f(x_{k+1})| < |f(x_k)|$, the magnitude of the function values must decrease, which means that the iterates tend to become trapped near a local minimizer of $|f(x)|$ that might be far from the solution. In the case of this function, $x = 0$ is a local minimizer of $|f(x)|$ while the true solution is $x^* \approx 8.12129$. Table 3.3.5 illustrates how the guarded Newton method behaves when it gets trapped near a local minimum of $|f(x)|$.

The behavior of our guarded Newton method can be summarized as follows.

**Theorem 3.9** *If $x_k$, $k = 0, 1, 2, \ldots$ are the iterates of the guarded Newton method Algorithm 42 with $f$ twice continuously differentiable and the iterates are bounded, then either $f(x_k) \to 0$ as $k \to \infty$, or $f'(x_k) \to 0$ as $k \to \infty$.*

***Proof*** Suppose that $f(x_k) \not\to 0$ as $k \to \infty$. Let $B$ be a bound on $|x_k|$ and $M$ a bound on $\left|f''(x)\right|$ for $|x| \leq B$. Also, let $s_k > 0$ be the value of $s$ used in line 10 of Algorithm 42, so that $x_{k+1} = x_k + s_k d_k$.

Since $|f(x_k)| > |f(x_{k+1})| > 0$ for all $k$, the sequence $|f(x_k)|$ converges to a limit $f^*$ as $k \to \infty$. Since $f(x_k) \not\to 0$ as $k \to \infty$, it follows that $f^* > 0$. Thus $|f(x_{k+1})| / |f(x_k)| \to f^*/f^* = 1$ as $k \to \infty$. As $|f(x_{k+1})| \leq (1 - \frac{1}{2}s_k)\,|f(x_k)|$, it follows that $s_k \to 0$ as $k \to \infty$. Since $f(x_k + s_k d_k) = f(x_k) + f'(x_k)s_k d_k + \frac{1}{2}f''(x_k + c_k s_k d_k)(s_k d_k)^2$ for some $0 < c_k < 1$, using the bound $M$ on $\left|f''(x)\right|$ for $|x| \leq B$, we see that $|f(x_k + s d_k)| \geq (1 - s)\,|f(x_k)| - M\,s^2 d_k^2$. We choose $s_k$ to be the first value of $s$ that is a negative power of two where $|f(x_k + s d_k)| \geq (1 - \frac{1}{2}s)\,|f(x_k)|$, which will happen if $M\,s^2\,d_k^2 \leq \frac{1}{2}\,|f(x_k)|$. We can therefore guarantee that the value of $s = s_k$ used is $\geq \frac{1}{2}\left[\frac{1}{2}\,|f(x_k)|\,/(M d_k^2)\right]^{1/2}$, and so $s_k\,|d_k| \geq [f^*/M]^{1/2}\,/(2\sqrt{2})$. Since $s_k \to 0$ it then follows that $|d_k| = |f(x_k)|\,/\left|f'(x_k)\right| \to \infty$ as $k \to \infty$. But this in turn implies that $f'(x_k) \to 0$, as we wanted. □

Note that the use of the sufficient decrease criterion (3.3.2) is needed to ensure that $s_k \to 0$ if $f(x_k) \not\to 0$. This means that there are circumstances in which we can guarantee that the guarded Newton method converges: if $\{\, x \mid |f(x)| \leq |f(x_0)| \,\}$ is a bounded set, and $f'(x) \neq 0$ for any $x$ in this set.

### 3.3.4 *Variant: Multivariate Newton method*

If we wish to solve a system of nonlinear equations $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ where $\boldsymbol{f}\colon \mathbb{R}^n \to \mathbb{R}^n$, we can use a well-known variant of Newton's method. Note that for $\boldsymbol{d} \approx \boldsymbol{0}$, $\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{d}) \approx \boldsymbol{f}(\boldsymbol{x}) + \nabla \boldsymbol{f}(\boldsymbol{x})\,\boldsymbol{d}$ where $\nabla \boldsymbol{f}(\boldsymbol{x})$ is the *Jacobian matrix*:

$$\nabla \boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} \partial f_1/\partial x_1(\boldsymbol{x}) & \partial f_1/\partial x_2(\boldsymbol{x}) & \cdots & \partial f_1/\partial x_n(\boldsymbol{x}) \\ \partial f_2/\partial x_1(\boldsymbol{x}) & \partial f_2/\partial x_2(\boldsymbol{x}) & \cdots & \partial f_2/\partial x_n(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1(\boldsymbol{x}) & \partial f_n/\partial x_2(\boldsymbol{x}) & \cdots & \partial f_n/\partial x_n(\boldsymbol{x}) \end{bmatrix}. \tag{3.3.3}$$

Solving $\boldsymbol{f}(\boldsymbol{x}) + \nabla \boldsymbol{f}(\boldsymbol{x})\,\boldsymbol{d} = \boldsymbol{0}$ for $\boldsymbol{d}$ instead of directly solving $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ means solving linear equations. In fact, $\boldsymbol{d} = -\nabla \boldsymbol{f}(\boldsymbol{x})^{-1}\boldsymbol{f}(\boldsymbol{x})$, and so the improved approximation for the solution is $\boldsymbol{x} + \boldsymbol{d} = \boldsymbol{x} - \nabla \boldsymbol{f}(\boldsymbol{x})^{-1}\boldsymbol{f}(\boldsymbol{x})$. Repeating this improvement gives the multivariate Newton method, as shown in Algorithm 43.

The convergence theory follows that for the one-variable case, except that we must be more careful in our application of Taylor series with remainder, which should be kept in integral form.

**Algorithm 43** Multivariate Newton method

```
function mvnewton(f, ∇f, x_0, ε)
  k ← 0
  while ‖f(x_k)‖_2 > ε
    x_{k+1} ← x_k − ∇f(x_k)^{-1} f(x_k)
    k ← k + 1
  end while
  return x_k
end function
```

**Theorem 3.10** *(Convergence of multivariate Newton's method) Suppose that* $\boldsymbol{f}\colon \mathbb{R}^n \to \mathbb{R}^n$ *has continuous second derivatives,* $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$*, and* $\nabla \boldsymbol{f}(\boldsymbol{x}^*)$ *is invertible. Then there is a* $\delta > 0$ *where* $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \delta$ *implies that* $\boldsymbol{x}_k \to \boldsymbol{x}^*$ *and there is a constant* $C$ *such that* $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\| \leq C \|\boldsymbol{x}_k - \boldsymbol{x}^*\|^2$ *for* $k = 0, 1, 2, \ldots$.

***Proof*** We start with multivariate Taylor series with second-order remainder in integral form (1.6.5):

$$\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{d}) = \boldsymbol{f}(\boldsymbol{x}) + \nabla \boldsymbol{f}(\boldsymbol{x})\, \boldsymbol{d} + \int_0^1 (1 - s)\, D^2 \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})[\boldsymbol{d}, \boldsymbol{d}]\, ds.$$

We first want to find $\delta > 0$ where $\|\boldsymbol{x}_0 - \boldsymbol{x}^*\| < \delta$ implies $\|\boldsymbol{x}_k - \boldsymbol{x}^*\| < \delta$ for $k = 1, 2, \ldots$. Let $M = \max_{\boldsymbol{y}:\|\boldsymbol{y}-\boldsymbol{x}^*\|\leq 1} \left\| D^2 \boldsymbol{f}(\boldsymbol{y}) \right\|$; we will ensure that $\delta \leq 1$ in what follows. Noting that if $A$ is invertible, then $B$ is invertible provided $\left\| A^{-1} \right\| \|B - A\| < 1$ by (2.1.1). If $A = \nabla \boldsymbol{f}(\boldsymbol{x}^*)$ and $B = \nabla \boldsymbol{f}(\boldsymbol{y})$ and $\|\boldsymbol{y} - \boldsymbol{x}^*\| \leq 1$ then $\|B - A\| \leq M \|\boldsymbol{y} - \boldsymbol{x}^*\|$. To ensure invertibility of $B = \nabla \boldsymbol{f}(\boldsymbol{y})$, we require $\left\| A^{-1} \right\| \|B - A\| \leq 1/2$; this is guaranteed provided we additionally require that $\|\boldsymbol{y} - \boldsymbol{x}^*\| \leq 1/(2 \left\| A^{-1} \right\| M)$. Set $\delta_1 = \min(1,\ 1/(2 \left\| A^{-1} \right\| M))$. This requirement also ensures that

$$\left\| B^{-1} \right\| \leq \frac{\left\| A^{-1} \right\|}{1 - \left\| A^{-1}(B - A) \right\|} \leq 2 \left\| A^{-1} \right\|.$$

Let $\boldsymbol{e}_k = \boldsymbol{x}_k - \boldsymbol{x}^*$. Then

$$\begin{aligned} \boldsymbol{0} = \boldsymbol{f}(\boldsymbol{x}^*) &= \boldsymbol{f}(\boldsymbol{x}_k - \boldsymbol{e}_k) \\ &= \boldsymbol{f}(\boldsymbol{x}_k) + \nabla \boldsymbol{f}(\boldsymbol{x}_k)(-\boldsymbol{e}_k) + \boldsymbol{\eta}_k, \\ &\qquad \text{with } \left\| \boldsymbol{\eta}_k \right\| \leq M \|\boldsymbol{e}_k\|^2, \text{ so} \\ \boldsymbol{f}(\boldsymbol{x}_k) &= \nabla \boldsymbol{f}(\boldsymbol{x}_k)\boldsymbol{e}_k - \boldsymbol{\eta}_k. \end{aligned}$$

Therefore

$$\begin{aligned} \boldsymbol{e}_{k+1} = \boldsymbol{x}_{k+1} - \boldsymbol{x}^* &= \left(\boldsymbol{x}_k - \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \boldsymbol{f}(\boldsymbol{x}_k)\right) - \boldsymbol{x}^* \\ &= \boldsymbol{x}_k - \boldsymbol{x}^* - \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \boldsymbol{f}(\boldsymbol{x}_k) \end{aligned}$$

$$
\begin{aligned}
&= \boldsymbol{e}_k - \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \left[ \nabla \boldsymbol{f}(\boldsymbol{x}_k) \boldsymbol{e}_k - \boldsymbol{\eta}_k \right] \\
&= \boldsymbol{e}_k - \boldsymbol{e}_k + \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \boldsymbol{\eta}_k = \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \boldsymbol{\eta}_k .
\end{aligned}
$$

Then provided $\|\boldsymbol{e}_k\| \leq \delta_1$,

$$
\begin{aligned}
\|\boldsymbol{e}_{k+1}\| &\leq \left\| \nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1} \right\| \left\| \boldsymbol{\eta}_k \right\| \\
&\leq 2 \left\| \nabla \boldsymbol{f}(\boldsymbol{x}^*)^{-1} \right\| M \, \|\boldsymbol{e}_k\|^2 .
\end{aligned}
$$

Provided $2 \left\| \nabla \boldsymbol{f}(\boldsymbol{x}^*)^{-1} \right\| M \, \|\boldsymbol{e}_k\| \leq 1/2$ we have $\|\boldsymbol{e}_{k+1}\| \leq \frac{1}{2} \|\boldsymbol{e}_k\|$. Let $\delta_2 = \min(\delta_1, \, 1/(4 \left\| \nabla \boldsymbol{f}(\boldsymbol{x}^*)^{-1} \right\| M))$. Then $\|\boldsymbol{e}_k\| \leq \delta_2$, implies $\|\boldsymbol{e}_{k+1}\| \leq \frac{1}{2} \|\boldsymbol{e}_k\| < \delta_2$. By induction, it is clear that if $\|\boldsymbol{e}_0\| \leq \delta_2$ then $\|\boldsymbol{e}_k\| \leq \delta_2$ for $k = 1, 2, \ldots$. Furthermore, $\|\boldsymbol{e}_{k+1}\| \leq \frac{1}{2} \|\boldsymbol{e}_k\|$ for all $k$ and so $\boldsymbol{e}_k \to \boldsymbol{0}$ as $k \to \infty$. For the rate of convergence, we use

$$
\|\boldsymbol{e}_{k+1}\| \leq 2 \left\| \nabla \boldsymbol{f}(\boldsymbol{x}^*)^{-1} \right\| M \, \|\boldsymbol{e}_k\|^2 .
$$

Choosing, $C = 2 \left\| \nabla \boldsymbol{f}(\boldsymbol{x}^*)^{-1} \right\| M$ gives $\|\boldsymbol{e}_{k+1}\| \leq C \, \|\boldsymbol{e}_k\|^2$ as we wanted. □

If $\boldsymbol{x}_0$ is *not* close to $\boldsymbol{x}^*$, we have no guarantee that Newton's method converges. We can instead make Newton's method more reliable by using a multivariate guarded Newton method (see Algorithm 42). Note that

$$
\begin{aligned}
\frac{d}{ds} \left( \|\boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})\|_2^2 \right) &= \frac{d}{ds} \left( \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})^T \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d}) \right) \\
&= 2 \, \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})^T \frac{d}{ds} \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d}) \\
&= 2 \, \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})^T \nabla \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d}) \, \boldsymbol{d} .
\end{aligned}
$$

If we choose $\boldsymbol{d}$ to be $-\nabla \boldsymbol{f}(\boldsymbol{x})^{-1} \boldsymbol{f}(\boldsymbol{x})$ then

$$
\begin{aligned}
\frac{d}{ds} \left( \|\boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})\|_2^2 \right) \Big|_{s=0} &= 2 \, \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})^T \nabla \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d}) \, \boldsymbol{d} \big|_{s=0} \\
&= -2 \, \boldsymbol{f}(\boldsymbol{x}) \, \nabla \boldsymbol{f}(\boldsymbol{x}) \, \boldsymbol{f}(\boldsymbol{x})^{-1} \boldsymbol{f}(\boldsymbol{x}) \\
&= -2 \, \|\boldsymbol{f}(\boldsymbol{x})\|_2^2 ,
\end{aligned}
$$

so

$$
\frac{d}{ds} \left( \|\boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})\|_2 \right) \Big|_{s=0} = - \, \|\boldsymbol{f}(\boldsymbol{x})\|_2 .
$$

Thus our sufficient decrease criterion can be written as

$$
\|\boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})\|_2 \leq (1 - \frac{1}{2} s) \, \|\boldsymbol{f}(\boldsymbol{x})\|_2 . \tag{3.3.4}
$$

**Algorithm 44** Guarded Newton method

```
1  function mvgnewton(f, ∇f, x_0, ϵ, η)
2    k ← 0
3    while ‖f(x_k)‖_2 > ϵ
4      d_k ← −∇f(x_k)^{−1} f(x_k)
5      s ← 1
6      while ‖f(x_k + s d_k)‖_2 > (1 − (1/2)s) ‖f(x_k)‖_2
7        s ← s/2
8        if s < η: return x_k
9      end while
10     x_{k+1} ← x_k + s d_k
11     k ← k + 1
12   end while
13   return x_k
14 end function
```

A complete multivariate guarded Newton method is given in Algorithm 44.

Theorem 3.9 can be modified to show that using (3.3.4) for the sufficient decrease criterion (line 6 of Algorithm 42), provided that the iterates $\boldsymbol{x}_k$ are bounded and that $\boldsymbol{f}$ has continuous second derivatives, results *either* in $\boldsymbol{x}_k \to \boldsymbol{x}^*$ as $k \to \infty$ where $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$, *or* $s_k \to 0$, $\|\boldsymbol{d}_k\| \to \infty$, and $\left\|\nabla \boldsymbol{f}(\boldsymbol{x}_k)^{-1}\right\| \to \infty$ as $k \to \infty$. If $\widehat{\boldsymbol{x}}$ is any limit point of the $\boldsymbol{x}_k$'s that is not a solution, then $\nabla \boldsymbol{f}(\widehat{\boldsymbol{x}})$ is not invertible. The proof of this follows Theorem 3.9, except that we first use $\|\boldsymbol{f}(\boldsymbol{x}_k)\| \downarrow f^*$ as $k \to \infty$.

**Exercises.**

(1) Apply Newton's method to solving $e^x + x^2 - 4 = 0$ with starting point $x_0 = 1$. How many iterations are needed for an accuracy of about $10^{-12}$?

(2) A method for computing square roots is to apply Newton's method to solving $x^2 - a = 0$ for $x$. Show that Newton's method applied to this equation is

$$x_{n+1} = \frac{1}{2}(x_n + \frac{a}{x_n}).$$

Show that the iteration function $g(x) = \frac{1}{2}(x + a/x)$ has a minimum at $x = \sqrt{a}$ and $g(\sqrt{a}) = \sqrt{a}$, so that $x_{n+1} \geq \sqrt{a}$. Show that Newton's method is globally convergent for any positive starting point. [**Hint:** Show that $0 \leq g'(x) \leq \frac{1}{2}$ for all $x \geq \sqrt{a}$, so that $g$ is a contraction mapping on $[\sqrt{a}, \infty)$.]

(3) While Newton's method for square roots is globally convergent (Exercise 2), if we are to use this to create an efficient method for all inputs, we want to ensure that the starting point is close to the solution. Since for floating point numbers, we have $a > 0$ represented by $a = +b \times 2^e$ where $1 \leq b < 2$, show how we can shift $a$ to $2^{2m}a$ in the interval $[\frac{1}{2},\ 2)$ without incurring any roundoff

error. How many iterations of Newton's method will guarantee a relative error of no more than $10^{-16}$ for $x_0 = 1$ and $\frac{1}{2} \le a < 2$? [**Hint:** If $u_n = x_n/\sqrt{a}$, then $u_{n+1} = \frac{1}{2}(u_n + 1/u_n)$. First consider $\frac{1}{2} \le a \le 1$; start with $u_0 = x_0/\sqrt{a}$ which is in the interval $[1, \sqrt{2}]$. The worst case is $u_0 = \sqrt{2}$ in the iteration $u_{n+1} = \frac{1}{2}(u_n + 1/u_n)$.]

(4) Apply Newton's method to the equation $x^m = 0$ with $m > 1$. How quickly does Newton's method converge for this problem?

(5) The guarded Newton method does not guarantee convergence to a solution, even if one exists. Apply the guarded Newton method to $f(x) = \frac{1}{3}x^3 - x + 10$ with $x_0 = 0.7$. What (if anything) do the iterates $x_k$ converge to?

(6) Show that for the guarded Newton method applied to solving $f(x) = 0$ for $f : \mathbb{R} \to \mathbb{R}$, either the iterates $x_k$, $k = 1, 2, \ldots$ are unbounded, or $f(x_k) \to 0$ as $k \to \infty$, or $f'(x_k) \to 0$ as $k \to \infty$.

(7) Show that it is possible that the iterates $x_k$, $k = 1, 2, \ldots$ are unbounded for the guarded Newton method: use $f(x) = e^{-x}$ and $x_0 = 0$.

(8) Apply the guarded multivariate Newton method (Algorithm 44) to the function

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} yx - y^2 \\ e^x - 4y + 1 \end{bmatrix} \qquad \text{where } \boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$$

with $\boldsymbol{x}_0 = [1,\ 1]^T$, $\boldsymbol{x}_0 = [1,\ -\frac{1}{2}]^T$, and with $\boldsymbol{x}_0 = [\frac{1}{2},\ -\frac{1}{2}]^T$. Report the solution you obtain with a tolerance $10^{-10}$. How many iterations does this method need?

(9) The guarded multivariate Newton method (Algorithm 44) uses the 2-norm. Would the method work well with a different norm here? Try the previous exercise with the 1-norm and the $\infty$-norm? Specifically answer the questions for using a norm different from the 2-norm:

(a) For $\boldsymbol{f}$ smooth and $\nabla \boldsymbol{f}(\boldsymbol{x}_k)$ invertible, is the line search in lines 5–9 guaranteed to terminate in exact arithmetic (even with $\eta = 0$)?

(b) Does the method accept $s = 1$ in line 6 for $\boldsymbol{x}_k \approx \boldsymbol{x}^*$ provided $\nabla \boldsymbol{f}(\boldsymbol{x}^*)$ is invertible and $\boldsymbol{f}$ smooth?

(10) ⚠ In this extensive exercise, we will establish bounds on the number of Newton steps for Algorithm 44 under some common assumptions:

$$\left\| \left( \nabla \boldsymbol{f}(\boldsymbol{x}) - \nabla \boldsymbol{f}(\boldsymbol{y}) \right) \nabla \boldsymbol{f}(\boldsymbol{x})^{-1} \right\| \le L \left\| \boldsymbol{x} - \boldsymbol{y} \right\|$$
$$\left\| \nabla \boldsymbol{f}(\boldsymbol{x})^{-1} \right\| \le M$$

for all $\boldsymbol{x}$ and $\boldsymbol{y}$.

(a) Show that for $\boldsymbol{d} = \nabla \boldsymbol{f}(\boldsymbol{x})^{-1} \boldsymbol{f}(\boldsymbol{x})$,

$$\begin{aligned} \boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d}) &= \boldsymbol{f}(\boldsymbol{x}) + s\nabla \boldsymbol{f}(\boldsymbol{x})\boldsymbol{d} + \int_0^s \left[\nabla \boldsymbol{f}(\boldsymbol{x} + t\boldsymbol{d}) - \nabla \boldsymbol{f}(\boldsymbol{x})\right] \boldsymbol{d}\, dt \\ &= (1 - s)\boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{\eta}(s) \qquad \text{where} \\ \|\boldsymbol{\eta}(s)\| &\le \frac{1}{2} L s^2 \|\boldsymbol{d}\| \, \|\boldsymbol{f}(\boldsymbol{x})\| \le \frac{1}{2} LM s^2 \, \|\boldsymbol{f}(\boldsymbol{x})\|^2 . \end{aligned}$$

(b) Show that the sufficient decrease criterion $\|\boldsymbol{f}(\boldsymbol{x} + s\boldsymbol{d})\| \le (1 - \frac{1}{2}s) \, \|\boldsymbol{f}(\boldsymbol{x})\|$ is satisfied if $s \, \|\boldsymbol{f}(\boldsymbol{x})\| \le 1/(LM)$.

(c) From (b) show that if $\boldsymbol{x} = \boldsymbol{x}_k$ then the chosen step length $s_k$ in Algorithm 44 satisfies either $s_k = 1$ or $1/(LM) \ge s_k \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| \ge \frac{1}{2}/(LM)$.

(d) Show that

$$\|\boldsymbol{f}(\boldsymbol{x}_{k+1})\| \le (1 - \frac{1}{2}s_k) \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| \le \max(\frac{1}{2} \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| \, , \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| - \frac{1}{4LM}) \qquad \text{and}$$

$$LM \, \|\boldsymbol{f}(\boldsymbol{x}_{k+1})\| \le \frac{1}{2} \, (LM \, \|\boldsymbol{f}(\boldsymbol{x}_k)\|)^2 \qquad \text{if } LM \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| \le \frac{1}{2}.$$

(e) Combine these results to show that for the first phase (if $LM \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| > \frac{1}{2}$) we have at most $4LM \, \|\boldsymbol{f}(\boldsymbol{x}_0)\|$ iterations, and a quadratic convergence phase (where $LM \, \|\boldsymbol{f}(\boldsymbol{x}_k)\| > \frac{1}{2}$) which has no more than $1 + \log_2^+(\log_2^+(LM/\epsilon))$ iterations to achieve $\|\boldsymbol{f}(\boldsymbol{x}_k)\| \le \epsilon$. Note that $\log_2^+(u) = \log_2(u)$ if $u \ge 1$ and zero otherwise. Give a bound for the total number of iterations needed.

(f) Give a bound on the total number of *function evaluations* (not just Newton steps). This takes into account the cost of the line search.

## 3.4 Secant and hybrid methods

There are a number of ways of improving on Newton's method. One is to avoid the need for computing derivatives. Another is to improve reliability by incorporating aspects of the bisection method. The first approach leads to the *secant method*, while the second approach leads to *Regula Falsi* and other hybrid methods such as Dekker's method and Brent's method.

All of the methods of this section are restricted to functions of one variable.

### *3.4.1 Convenience: Secant method*

Computing derivatives can be a difficult task, especially if the function is defined through a complex piece of code. Instead, if we have already computed $f(x_k)$ and $f(x_{k-1})$, we can approximate $f'(x_k) \approx (f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$. Substituting this into (3.3.1) gives the secant method:

**Algorithm 45** Secant method

```
function secant(f, x_0, x_1, ε)
  k ← 1
  while |f(x_k)| > ε
    x_{k+1} ← x_k − f(x_k) (x_k − x_{k−1})/(f(x_k) − f(x_{k−1}))
    k ← k + 1
  end while
  return x_k
end function
```

**Table 3.4.1** Comparison of Newton's method with secant method for $f(x) = e^x - x^2 - 3$

| | Newton method | | Secant method | |
|---|---|---|---|---|
| $k$ | $x_k$ | $f(x_k)$ | $x_k$ | $f(x_k)$ |
| 0 | 1.000000000000000 | $-1.2817 \times 10^{0}$ | 1.000000000000000 | $-1.2817 \times 10^{0}$ |
| 1 | 2.784422382354665 | $+5.4375 \times 10^{0}$ | 2.000000000000000 | $+3.8906 \times 10^{-1}$ |
| 2 | 2.272498962410127 | $+1.5394 \times 10^{0}$ | 1.767140238028185 | $-2.6870 \times 10^{-1}$ |
| 3 | 1.974092118993297 | $+3.0304 \times 10^{-1}$ | 1.862265070280968 | $-2.9728 \times 10^{-2}$ |
| 4 | 1.880903342392723 | $+2.1630 \times 10^{-2}$ | 1.874098597392787 | $+2.6983 \times 10^{-3}$ |
| 5 | 1.873171697204092 | $+1.3577 \times 10^{-4}$ | 1.873113870758385 | $-2.3969 \times 10^{-5}$ |
| 6 | 1.873122549684362 | $+5.4455 \times 10^{-9}$ | 1.873122540803784 | $-1.9086 \times 10^{-8}$ |
| 7 | 1.873122547713043 | $-4.4409 \times 10^{-16}$ | 1.873122547713092 | $+1.3456 \times 10^{-13}$ |

$$x_{k+1} = x_k - \frac{f(x_k)}{(f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})} = x_k - \frac{f(x_k)\,(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}. \tag{3.4.1}$$

Note that $x_{k+1}$ is the point where the chord line passing through $(x_k,\ f(x_k))$ and $(x_{k-1},\ f(x_{k-1}))$ crosses the $x$-axis. Thus, the right-hand side of (3.4.1) is actually symmetric in $x_k$ and $x_{k-1}$:

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}.$$

The secant method is shown in Algorithm 45.

An example of using the secant method compared to the Newton method is shown in Table 3.4.1.

As can be seen in Table 3.4.1, secant has a "head start" (probably because of the choice $x_1 = 2$, which is much closer to $x^*$ than $x_0$), but Newton's method overtakes the secant method in terms of accuracy. Nevertheless, the secant method has impressively fast convergence. The reason for this can be seen in the following theorem.

**Theorem 3.11** *If $f$ has continuous second derivatives and $f(x^*) = 0$ but $f'(x^*) \neq 0$, then there is a $\delta > 0$ where $|x_0 - x^*| < \delta$ and $|x_1 - x^*| < \delta$ implies $x_k \to x^*$*

*as* $k \to \infty$. *Furthermore, if* $x_k \to x^*$ *as* $k \to \infty$, $(x_{k+1} - x^*)/((x_k - x^*)(x_{k-1} - x^*)) \to f''(x^*)/(2\,f'(x^*))$ *as* $k \to \infty$.

***Proof*** Note that the chord line for computing $x_{k+1}$ is the linear interpolant $p_{k,1}(x)$ of data points $(x_k,\ f(x_k))$ and $(x_{k-1},\ f(x_{k-1}))$, and $x_{k+1}$ is where this chord line crosses the $x$-axis: $p_{k,1}(x_{k+1}) = 0$. From the error formula for linear interpolation (4.1.7),

$$f(x^*) - p_{k,1}(x^*) = \frac{1}{2} f''(c_k)(x^* - x_k)(x^* - x_{k-1}),$$

for some $c_k$ between $\min(x_{k-1}, x_k, x^*)$ and $\max(x_{k-1}, x_k, x^*)$. Therefore,

$$-\frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}(x^* - x_{k+1}) = \frac{1}{2} f''(c_k)(x^* - x_k)(x^* - x_{k-1}),$$

$p_{k,1}$ is linear with slope $(f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$, and $p_{k,1}(x_{k+1}) = 0$. The fraction $(f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$ equals $f'(d_k)$ for some $d_k$ between $x_{k-1}$ and $x_k$. So

$$-f'(d_k)(x^* - x_{k+1}) = \frac{1}{2} f''(c_k)(x^* - x_k)(x^* - x_{k-1}).$$

Writing $e_j = x_j - x^*$ we get $e_{k+1} = e_k e_{k-1}\, f''(c_k)/(2f'(d_k))$. Choose $\delta_1 > 0$ so that $\left|f'(z)\right| \geq \frac{1}{2}\left|f'(x^*)\right|$ for all $z$ where $|z - x^*| \leq \delta_1$. Let $M = \max_{u:|u-x^*|\leq\delta_1} \left|f''(u)\right| / (\frac{1}{2}\left|f'(x^*)\right|)$. Then $|e_{k+1}| \leq M\,|e_k|\,|e_{k-1}|$ provided $x_k, x_{k-1} \in [x^* - \delta_1,\ x^* + \delta_1]$. Let $\delta_2 = \min(\delta_1,\ 1/(2M))$. Then provided $|e_k|\,, |e_{k-1}| \leq \delta_2$ we have $|e_{k+1}| \leq M\,|e_k|\,|e_{k-1}| \leq \frac{1}{2}\,|e_k| \leq \delta_2$.

Thus, if $|e_0| = |x_0 - x^*| \leq \delta_2$ and $|e_1| = |x_1 - x^*| \leq \delta_2$, we have $|e_{k+1}| \leq M\,|e_k|\,|e_{k-1}| \leq \frac{1}{2}\,|e_k| \leq \delta_2$ for all $k = 1, 2, 3, \ldots$. This gives us convergence: $e_k \to 0$ as $k \to \infty$ and thus $x_k \to x^*$ as $k \to \infty$. Furthermore, $c_k \to x^*$ and $d_k \to x^*$ as $k \to \infty$; finally,

$$\lim_{k\to\infty} \frac{(x_{k+1} - x^*)}{(x_k - x^*)(x_{k-1} - x^*)} = \lim_{k\to\infty} \frac{f''(c_k)}{2\,f'(d_k)} = \frac{f''(x^*)}{2\,f'(x^*)}$$

by the Squeeze theorem, as we wanted. □

Letting $e_j = x_j - x^*$ we see that $e_{k+1}/(e_k e_{k-1}) \to C := f''(x^*)/(2\,f'(x^*))$. Thus if there is convergence, for sufficiently large $k$, we have $|e_{k+1}| \leq 2C\,|e_k|\,|e_{k-1}|$. Multiplying by $2C$ gives

$$2C\,|e_{k+1}| \leq 2C\,|e_k|\,2C\,|e_{k-1}|\,.$$

Taking logarithms,

$$\ln(2C\,|e_{k+1}|) \leq \ln(2C\,|e_k|) + \ln(2C\,|e_{k-1}|).$$

Setting $\eta_j = \ln(2C\,|e_j|)$ we get $\eta_{k+1} \leq \eta_k + \eta_{k-1}$. If we have $\eta_0, \eta_1 < 0$ then we see that $\eta_k \to -\infty$ as $k \to \infty$, and furthermore the size of $\eta_k$ grows exponentially fast.

How fast is this growth? We can determine that by looking at the linear recurrence

$$\widetilde{\eta}_{k+1} = \widetilde{\eta}_k + \widetilde{\eta}_{k-1}, \qquad \widetilde{\eta}_0 = \eta_0 \text{ and } \widetilde{\eta}_1 = \eta_1. \tag{3.4.2}$$

It is easy to check that $\eta_j \le \widetilde{\eta}_j$ for $j = 0, 1, 2, \ldots$ by induction, so $\widetilde{\eta}_j$ is an upper bound on $\eta_j = \ln(2C\left|e_j\right|)$. We can solve (3.4.2) exactly: $\widetilde{\eta}_j = C_1 r_1^j + C_2 r_2^j$ where $r_1$ and $r_2$ are solutions of the characteristic equation $r^2 = r + 1$. The solutions are the *Golden ratio* $\phi := (1 + \sqrt{5})/2 \approx 1.6180$ and $-1/\phi \approx -0.6180$. This means that $\ln(2C\ |e_k|) \approx -\text{const}\,\phi^k$ for $k$ large, provided $e_0$ and $e_1$ sufficiently small.

In comparison with Newton's method where we have $|e_{k+1}| \le C'\,|e_k|^2$ so $\ln(C'\,|e_{k+1}|) \le 2\,\ln(C'\,|e_k|)$. This means that $\ln(C'\,|e_k|) \approx -\text{const}\,2^k$ provided $e_0$ is sufficiently small. The number of iterations needed for Newton's method are therefore generally less than for the secant method, but only by a factor of about $\log\phi/\log 2 \approx 0.69$. Considering that one iteration of Newton's method requires both a function and a derivative evaluation, while the secant method only requires one function evaluation, the additional iterations are often a price worth paying.

### 3.4.2 *Regula Falsi*

Regula Falsi, at least as the name is usually understood today, is an attempt to combine the speed of the secant method with the reliability of the bisection method. While it does achieve the reliability of the bisection method, it fails to obtain the speed of the secant method, and can even be slower than bisection.

The idea is to take the bisection method, but to compute the new point $c$ using the result of the secant method: $c \leftarrow a + f(a)(b-a)/(f(b) - f(a))$ instead of the midpoint $m = (a+b)/2$. We then update either $a \leftarrow c$ or $b \leftarrow c$ according to the sign of $f(c)$. This is shown in Algorithm 46. Note that as $f(a)$ and $f(b)$ have opposite signs, the zero of the linear interpolant (which is $c$) must lie between $a$ and $b$.

Applied to the function $f(x) = e^x - x^2 - 3$ with $a = 1$ and $b = 2$, we get the results shown in Table 3.4.2. Note that the values $a_k$ and $b_k$ are the values at the end of the $k$th execution of lines 6–12 of Algorithm 45.

Several things are apparent from Table 3.4.2: $b_k$=$b_0$ for all $k$, and $f(a_{k+1})/f(a_k) \approx 1/10$ for $k \ge 2$. It appears from this that the ratio of successive errors $e_{k+1}/e_k$ does not go to zero, indicating a linear rate of convergence similar to general fixed-point iterations. This method does not appear to have the rapid convergence of Newton's method or the secant method.

To analyze this method to see why this happens, suppose that $f''(x) > 0$ for all $x \in [a, b]$. If $p_1(x)$ is the linear interpolant of $(a,\ f(a))$ and $(b,\ f(b))$, then for some $d_x$ between $a$ and $b$, by (4.1.7),

**Algorithm 46** Regula Falsi method

```
function regulafalsi(f, a, b, ε)
if sign f(a) ≠ −sign f(b)
  return fail
end if
while min(|f(a)|, |f(b)|) > ε
  c ← a + f(a)(b − a)/(f(b) − f(a))
  if f(c) = 0: return c
  if sign f(c) = sign f(a)
    a ← c
  else
    b ← c
  end if
end while
if |f(a)| < |f(b)| return a else return b
end function
```

**Table 3.4.2** Results for Regula Falsi applied to $f(x) = e^x - x^2 - 3$ with initial $[a, b] = [1, 2]$

| $k$ | $a_k$ | $f(a_k)$ | $b_k$ | $f(b_k)$ |
|---|---|---|---|---|
| 0 | 1 | $-1.28172 \times 10^{0}$ | 2 | 0.389056 |
| 1 | 1.76714023802819 | $-2.68697 \times 10^{-1}$ | 2 | 0.389056 |
| 2 | 1.86226507028097 | $-2.97277 \times 10^{-2}$ | 2 | 0.389056 |
| 3 | 1.87204229781722 | $-2.98139 \times 10^{-3}$ | 2 | 0.389056 |
| 4 | 1.87301539863046 | $-2.95957 \times 10^{-4}$ | 2 | 0.389056 |
| 5 | 1.87311192293272 | $-2.93490 \times 10^{-5}$ | 2 | 0.389056 |
| 6 | 1.87312149420393 | $-2.91015 \times 10^{-6}$ | 2 | 0.389056 |

$$f(x) - p_1(x) = \frac{f''(d_x)}{2!}(x-a)(x-b) \qquad \text{so for the new point } c,$$
$$f(c) - 0 = f(c) - p_1(c) = \frac{f''(d_c)}{2!}(c-a)(c-b) < 0.$$

That is, the sign of $f(c)$ will always be negative. This means that only one end of the interval will be updated in the body of the `while` loop. A similar result holds if $f''(x) < 0$ for all $x \in [a, b]$. Then if, as in our example, $a_k$ increases toward the solution $x^*$, but $b_k = b_0$ for all $k$, then $b_k - a_k \to b_0 - x^* \neq 0$. The slope of the chord line will not approach the slope of the tangent line at $x^*$. This gives us linear convergence rather than the accelerated convergence of the Newton or secant methods.

We now make these arguments more precise. Let $p_{1,k}(x)$ be the linear interpolant of $(a_k,\ f(a_k))$ and $(b_k,\ f(b_k))$ so that $c_k = a_k - f(a_k)(b_k - a_k)/(f(b_k) - f(a_k))$ is the solution of $p_{1,k}(c_k) = 0$. Assume that $f''(x) > 0$ for all $x \in [a_0, b_0]$, and $f(a_0) < 0$. By the above arguments, $f''(x) > 0$ for all $x \in [a_k, b_k]$, $f(a_k) < 0 < f(b_k)$, and $b_k = b_0$ for all $k$. Then

$$a_{k+1} = c_k = a_k - \frac{f(a_k)(b_k - a_k)}{f(b_k) - f(a_k)}.$$

As with the analysis of the secant method, we note that $f(x^*) = 0$, so

$$-p_{1,k}(x^*) = f(x^*) - p_{1,k}(x^*) = \frac{f''(d_k)}{2!}(x^* - a_k)(x^* - b_k),$$

for some $d_k$ between $a_k$ and $b_k$. Noting that $p_{1,k}(x^*) = [(f(b_k) - f(a_k))/(b_k - a_k)](x^* - c_k)$ since $p_{1,k}(c_k) = 0$, we have

$$-\frac{f(b_k) - f(a_k)}{b_k - a_k}(x^* - c_k) = \frac{f''(d_k)}{2!}(x^* - a_k)(x^* - b_k).$$

Let $e_j = x^* - a_j$ be the error in $a_j$. Then as $c_k = a_{k+1}$,

$$-\frac{f(b_k) - f(a_k)}{b_k - a_k}e_{k+1} = \frac{1}{2}f''(d_k)\, e_k\, (x^* - b_k).$$

Thus

$$\begin{aligned}\frac{e_{k+1}}{e_k} &= -\frac{f''(d_k)\,(b_k - a_k)}{2(f(b_k) - f(a_k))}(x^* - b_k) \\ &\to \frac{f''(x^*)\,(b_0 - x^*)^2}{2(f(b_0) - f(x^*))} \qquad \text{as } k \to \infty,\end{aligned}$$

demonstrating linear convergence.

### 3.4.3 *Hybrid methods: Dekker's and Brent's methods*

The first method that truly combined the reliability of bisection with the speed of the secant method was Dekker's method [74]. Dekker's method was later eclipsed by Brent's method [32, Chap. 4].

Dekker's method keeps an interval $[\min(a_k, b_k),\ \max(a_k, b_k)]$ so that $f(a_k)$ and $f(b_k)$ have opposite signs, but with $|f(b_k)| \le |f(a_k)|$. Unlike the bisection method or Regula Falsi, Dekker's method does not treat $a_k$ and $b_k$ symmetrically, but $b_k$ is meant to be the better approximation to the solution. Dekker's method uses the secant update on the $b_k$'s

$$s_k \leftarrow b_k - \frac{f(b_k)\,(b_k - b_{k-1})}{f(b_k) - f(b_{k-1})},$$

where applicable, but uses the midpoint $m_k = (a_k + b_k)/2$ where it is not applicable ($f(b_k) = f(b_{k-1})$). The choice then is to use either $s_k$ or $m_k$. If $s_k$ is strictly

**Algorithm 47** Dekker's method

```
1   function dekker(f, a_0, b_0, ε)
2     if sign f(a_0) ≠ −sign f(b_0)
3       return fail
4     end if
5     k ← 0;  b_{−1} ← a_0
6     if |f(b_k)| > |f(a_k)|: swap a_k and b_k
7     while |f(b_k)| > ε
8       m_k ← (a_k + b_k)/2
9       if f(b_k) ≠ f(b_{k−1})
10        s_k ← b_k − f(b_k)(b_k − b_{k−1}) / (f(b_k) − f(b_{k−1}))
11      else
12        s_k ← m_k
13      end if
14      if m_k ≤ s_k ≤ b_k or b_k ≤ s_k ≤ m_k
15        b_{k+1} ← s_k
16      else
17        b_{k+1} ← m_k
18      end if
19      if sign f(b_{k+1}) = −sign f(a_k)
20        a_{k+1} ← a_k
21      else
22        a_{k+1} ← b_k
23      end if
24      if |f(b_{k+1})| > |f(a_{k+1})|: swap a_{k+1} & b_{k+1}; end if
25      k ← k + 1
26    end while
27    return b_k
28  end function
```

between $b_k$ and $m_k$ then we set $b_{k+1} \leftarrow s_k$, otherwise $b_{k+1} \leftarrow m_k$. This ensures that $b_{k+1} \in [\min(a_k, b_k),\ \max(a_k, b_k)]$, and also that $|b_{k+1} - b_k| \leq \frac{1}{2}|a_k - b_k|$. To obtain $a_{k+1}$ we choose between $a_k$ and $b_k$: if $\operatorname{sign} f(b_{k+1}) = \operatorname{sign} f(b_k) = -\operatorname{sign} f(a_k)$ then we choose $a_{k+1} \leftarrow a_k$; if $\operatorname{sign} f(b_{k+1}) = -\operatorname{sign} f(b_k) = \operatorname{sign} f(a_k)$, we choose $a_{k+1} \leftarrow b_k$. This maintains the sign invariant: $\operatorname{sign} f(a_{k+1}) = -\operatorname{sign} f(b_{k+1})$. Finally, we swap $a_{k+1}$ and $b_{k+1}$ if $|f(b_{k+1})| > |f(a_{k+1})|$. This ensures that $|f(b_{k+1})| \leq |f(a_{k+1})|$. The complete algorithm is shown in Algorithm 47.

Since $[\min(a_{k+1}, b_{k+1}),\ \max(a_{k+1}, b_{k+1})] \subsetneq [\min(a_k, b_k),\ \max(a_k, b_k)]$ there is no possibility of the method "blowing up". The condition that $\operatorname{sign} f(a_k) = -\operatorname{sign} f(b_k)$ ensures that there is a zero of $f$ between $a_k$ and $b_k$, but does not guarantee that the number of iterations is better than bisection. It does, however, perform nearer to the secant method where the secant method converges rapidly. There are cases where the secant method converges slowly and Dekker's method exactly tracks the secant method.

Brent's method is an improvement on Dekker's method in several ways. Inverse quadratic interpolation is used based on the data points $(a_k,\ f(a_k))$, $(b_k,\ f(b_k))$, and $(b_{k-1},\ f(b_{k-1}))$. Because it uses *inverse* quadratic interpolation, there is no need

to solve a quadratic equation; however, it does require that $f(b_k) \neq f(b_{k-1})$ and $f(a_k) \neq f(b_{k-1})$. If any of these conditions fails, we resort to linear interpolation. The condition that sign $f(a_k) = -\text{sign} f(b_k)$ ensures that $f(a_k) \neq f(b_k)$. Because inverse quadratic interpolation is used, the condition for accepting the secant or inverse quadratic interpolation estimate has to be expanded a little: instead of requiring that the inverse quadratic interpolant estimate $s_k$ lies between $m_k = (a_k + b_k)/2$ and $b_k$, we only insist that the interpolant estimate $s_k$ lies between $(3a_k + b_k)/4$ and $b_k$. Brent's method aims to guarantee a halving of the interval width $|b_k - a_k|$ at least *every two iterations*. To do this, if the *previous* step was *not* a bisection step (as indicated by *mflag* = *false*), and $|s_k - b_k| \geq \frac{1}{2}|b_{k-1} - b_{k-2}|$ on line 16, then we do a bisection step.

Ensuring the reduction of the interval width $|b_k - a_k|$ by a fixed ratio in a fixed number of iterations guarantees that $|b_k - a_k| \to 0$ exponentially as $k \to \infty$. To see how rapidly using inverse quadratic interpolation is, suppose that $\overline{p}_{2,k}(y)$ be the quadratic interpolant of the data points $(f(a_k),\ a_k)$, $(f(b_k),\ b_k)$, and $(f(b_{k-1}),\ b_{k-1})$. Supposing that the inverse function $f^{-1}$ is well defined and smooth on $[\min(a_k, b_k, b_{k-1}),\ \max(a_k, b_k, b_{k-1})] \subseteq [\min(a_0,\ b_0),\ \max(a_0,\ b_0)]$, we have from the error formula for polynomial interpolation (4.1.7),

$$f^{-1}(y) - \overline{p}_{2,k}(y) = \frac{(f^{-1})'''(\overline{c}_{k,y})}{3!}(y - f(a_k))(y - f(b_k))(y - f(b_{k-1})), \qquad \text{so}$$
$$f^{-1}(0) - \overline{p}_{2,k}(0) = -\frac{1}{6}(f^{-1})'''(\overline{c}_k)\, f(a_k)\, f(b_k)\, f(b_{k-1}).$$

Now $f^{-1}(0) = x^*$, the solution we seek, and $\overline{p}_{2,k}(0) = b_{k+1}$, so

$$\begin{aligned} b_{k+1} - x^* &= \frac{1}{6}(f^{-1})'''(\overline{c}_k)\, f(a_k)\, f(b_k)\, f(b_{k-1}) \\ &= \mathcal{O}((a_k - x^*)(b_k - x^*)(b_{k-1} - x^*)). \end{aligned}$$

Thus we can expect rapid convergence. We even get superlinear convergence without small $|a_k - x^*|$.

**Exercises.**

(1) Use Regular Falsi to solve $e^x + x^2 = 4$ to 10 digits of accuracy. How many function evaluations does it use?
(2) Use Dekker's method to solve $e^x + x^2 = 4$ to 10 digits of accuracy. How many function evaluations does it use?
(3) Use Brent's method to solve $e^x + x^2 = 4$ to 10 digits of accuracy. How many function evaluations does it use?
(4) Use the three above methods to find the solution of $x = \tan x$ closest to $3\pi/2$. Give its value to 10 digits of accuracy. Report the number of function evaluations needed.

**Algorithm 48** Brent's method

```
1   function brent(f, a_0, b_0, ε, δ)
2     if sign f(a_0) ≠ −sign f(b_0)
3       return fail
4     end if
5     k ← 0;  b_{−1} ← a_0;  mflag ← true
6     if |f(b_k)| > |f(a_k)|: swap a_k & b_k; end if
7     while |f(b_k)| > ε and |b_k − a_k| > δ
8       m_k ← (a_k + b_k)/2
9       if f(b_k) ≠ f(b_{k−1}) and f(a_k) ≠ f(b_{k−1})
          // inverse quadratic interpolation
10        s_k ← a_k f(b_k) f(b_{k−1}) / ((f(a_k) − f(b_k))(f(a_k) − f(b_{k−1}))) +
                 b_k f(a_k) f(b_{k−1}) / ((f(b_k) − f(a_k))(f(b_k) − f(b_{k−1}))) +
                 b_{k−1} f(a_k) f(b_k) / ((f(b_{k−1}) − f(a_k))(f(b_{k−1}) − f(b_k)))
11      else // secant update
12        s_k ← b_k − f(b_k)(b_k − a_k) / (f(b_k) − f(a_k))
13      end if
14      if s_k not between b_k and (3a_k + b_k)/4 or
15            (mflag & [ |s_k − b_k| ≥ 1/2 |b_k − b_{k−1}| or δ >
                         |b_k − b_{k−1}| ]) or
16        (not mflag & [ |s_k − b_k| ≥ 1/2 |b_{k−1} − b_{k−2}| or δ >
                         |b_{k−1} − b_{k−2}| ])
17        s_k ← m_k
18        mflag ← true
19      else
20        mflag ← false
21      end if
22      if sign f(s_k) = −sign f(a_k)
23        a_{k+1} ← a_k;  b_{k+1} ← s_k
24      else
25        b_{k+1} ← b_k;  a_{k+1} ← s_k
26      end if
27      if |f(b_{k+1})| > |f(a_{k+1})|: swap a_{k+1} & b_{k+1}; end if
28      k ← k + 1
29    end while
30    return b_k
31  end function
```

(5) Suppose a modified hybrid method alternates between bisection updates ($c_k \leftarrow (a_k + b_k)/2$) for even $k$ and secant updates ($c_k \leftarrow a_k - f(a_k)(b_k - a_k)/(f(b_k) - f(a_k))$) for odd $k$, and $a_k \leftarrow c_k$ if sign $f(c_k)$=sign $f(a_k)$ and $b_k \leftarrow c_k$ otherwise. Show that $|b_{k+2} - a_{k+2}| \,/\, |b_k - a_k| \rightarrow \frac{1}{2}$ as $k \rightarrow \infty$. Explain why this limits the performance of the hybrid method.

(6) Implement a hybrid method using the following update code:

$\widehat{c}_k \leftarrow a_k - f(a_k)(b_k - a_k)/(f(b_k) - f(a_k))$ `// secant estimate`
`if` $\widehat{c}_k$ `between` $a_k$ `and` $\frac{3}{4}a_k + \frac{1}{4}b_k$`:` $c_k \leftarrow 2\widehat{c}_k - a_k$`;` `end if`
`if` $\widehat{c}_k$ `between` $b_k$ `and` $\frac{3}{4}b_k + \frac{1}{4}a_k$`:` $c_k \leftarrow 2\widehat{c}_k - b_k$`;` `end if`
`if sign` $f(c_k) =$ `sign` $f(a_k)$`:` $a_k \leftarrow c_k$
`else:` $b_k \leftarrow c_k$.

Use this for solving $e^x + x^2 = 4$. Do the secant estimates $\widehat{c}_k$ have superlinear convergence to the exact solution?

(7) Are there conditions in which inverse quadratic interpolation can fail or cause numerical problems? Explain.

(8) Apply Regula Falsi, Dekker's method, and Brent's method to solving $x^2 \sin(1/x) = 0$ in the interval $[-3/(2\pi),\ 2/\pi]$. Do they find the same solution? Compare with the results from the bisection method.

(9) Trust region optimization methods often have to solve the equation $\left\|(B + \lambda I)^{-1}\boldsymbol{g}\right\|_2 = \Delta$ for $\lambda$ given $B$ symmetric $n \times n$ and $\boldsymbol{g} \in \mathbb{R}^n$. Trust region methods also need $B + \lambda I$ positive semi-definite (so that $\lambda + \lambda_{\min}(B) \geq 0$). Find an interval $[a, b]$ for $\lambda$ where $a, b \geq -\lambda_{\min}(B)$ and $\left\|(B + \lambda I)^{-1}\boldsymbol{g}\right\|_2 - \Delta$ has opposite signs at $\lambda = a$ and $\lambda = b$. To find $a$ and $b$, you should use only the quantities $\lambda_{\min}(B)$, $\lambda_{\max}(B)$, and $\|\boldsymbol{g}\|_2$. Adapt a hybrid method of your choice to solve this problem. You should assume that there is a function *isposdef*$(A)$ that returns *true* if matrix $A$ is positive definite and *false* otherwise.

(10) It is possible to use bisection and hybrid methods in two dimensions *in certain circumstances*. Consider a rectangle $R = [a, b] \times [c, d]$ and functions $f, g\colon \mathbb{R}^2 \to \mathbb{R}$ with the properties $f(a, y) < 0$ and $f(b, y) > 0$ for all $y \in [c, d]$, and $g(x, c) < 0$ and $g(x, d) > 0$ for all $x \in [a, b]$. In addition, assume that $f(x, y)$ is a strictly increasing function of $y$ so that for each $x$ there is one and only one solution $y = \widehat{y}(x)$ of $f(x, y) = 0$. See Figure 3.4.1

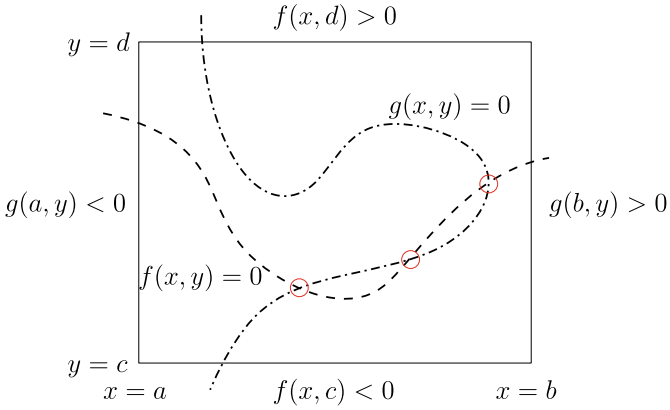



**Fig. 3.4.1** A special case where bisection and hybrid methods can solve a two-dimensional problem

for an illustration. Implement a solver based on nested bisection or hybrid methods. The inner method computes $y = \widehat{y}(x)$ that solves $f(x, y) = 0$ for a given $x$ by solving for $y$. The outer method solves $g(x, \widehat{y}(x)) = 0$ using a hybrid method solving for $x$. Test this on the rectangle $[0, 1] \times [0, 1]$ for functions $f(x, y) = e^x - x^2 + e^y(1 + x) - 1$ and $g(x, y) = \sin(\pi y/2) - x + xy$. Argue that without the requirement for $f(x, y)$ be increasing in $y$ this method can fail.

## 3.5 Continuation methods

When faced with a highly nonlinear system of equations, so far we just have multivariate Newton's method or one of its variants. Ensuring convergence, even assuming $\nabla \boldsymbol{f}(\boldsymbol{x}^*)$ is invertible, requires having an initial guess $\boldsymbol{x}_0$ that is "close enough" to the true solution $\boldsymbol{x}^*$. In the absence of good leads as to where $\boldsymbol{x}^*$ is, we can pick $\boldsymbol{x}_0$ "at random". Hopefully, if we try this enough times, we will chance on an $\boldsymbol{x}_0$ that is "close enough" to $\boldsymbol{x}^*$. But if the dimension is fairly large, the chance of this happening can be extremely small.

Here is another approach that can work in fairly general situations. The approach we develop here is fleshed out in more detail in [4]. We start with the system of equations we want to solve: $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ where $\boldsymbol{f}$ is a smooth function $\mathbb{R}^n \to \mathbb{R}^n$. We start with an easier system of equations: $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$. The function $\boldsymbol{g}$ should also be smooth with an easily determined solution. We could use $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{a}$ for a suitable, or even randomly chosen, $\boldsymbol{a}$. We then connect the easy problem to the hard problem with a *homotopy*: $\boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{0}$. We choose $\boldsymbol{h} \colon \mathbb{R}^n \times [0, 1] \to \mathbb{R}^n$ so that $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{g}(\boldsymbol{x})$ for all $\boldsymbol{x}$ (the "easy" function) and $\boldsymbol{h}(\boldsymbol{x}, 1) = \boldsymbol{f}(\boldsymbol{x})$ for all $\boldsymbol{x}$ (the "hard" function). We start from the solution $\boldsymbol{x}_0$ for $\boldsymbol{h}(\boldsymbol{x}, 0) = \boldsymbol{0}$. We then follow the path

$$C := \left\{ (\boldsymbol{x}, t) \in \mathbb{R}^n \times [0, 1] \mid \boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{0} \right\}$$

from $(\boldsymbol{x}_0, 0)$ to $(\boldsymbol{x}_1, 1)$. Then $\boldsymbol{x}_1$ is the solution of $\boldsymbol{f}(\boldsymbol{x}_1) = \boldsymbol{h}(\boldsymbol{x}_1, 1) = \boldsymbol{0}$.

Methods that follow a path of solutions for a homotopy like this are called *continuation* or *homotopy methods*. We need to identify situations in which this can be done in principle. We also need to fill in the details of the methods to achieve this and answer the questions about what can go wrong.

### *3.5.1 Following paths*

Figure 3.5.1 illustrates the kind of paths that can arise in a homotopy.

The implicit function theorem of multivariate calculus can be used to identify properties of the set

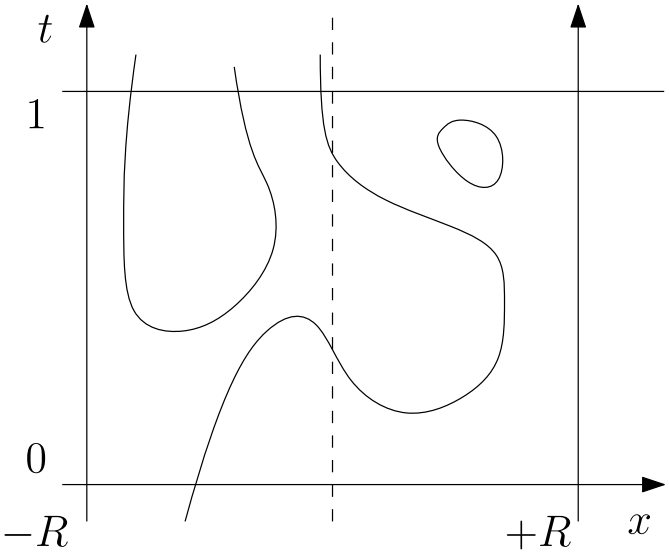



**Fig. 3.5.1** Illustration of homotopy paths

$$C := \left\{ (\boldsymbol{x}, t) \in \mathbb{R}^n \times [0, 1] \mid \boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{0} \right\}.$$

Specifically, if rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)] = n$ for all $(\boldsymbol{x}, t) \in C$, then $C$ is a one-dimensional manifold, possibly with boundary [114]. There are only two essentially different, connected, compact one-dimensional manifolds with or without boundary: an interval $[a, b]$ or a circle. Provided $C$ is compact, the connected component containing $(\boldsymbol{x}_0, 0)$ is therefore a continuous image of an interval $[a, b]$ or the continuous image of a circle. If $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ has only one solution $\boldsymbol{x} = \boldsymbol{x}_0$ and $\nabla \boldsymbol{g}(\boldsymbol{x}_0)$ is invertible, then the only possibility is the image of an interval. The other end of that image of $[a, b]$ must have either $t = 0$ or $t = 1$. But if $\boldsymbol{x} = \boldsymbol{x}_0$ is the only solution of $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$, then it is not possible for the other end point to be at $t = 0$. Thus, the other end point would have to be at $t = 1$, which is a point $(\boldsymbol{x}^*, 1)$ where $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$.

The condition that $C$ is compact means that the curve of $(\boldsymbol{x}, t)$ points cannot "run off to infinity". This depends on the homotopy that is chosen and the function $\boldsymbol{f}$. One of the simplest conditions that can ensure this is the following:

$$\boldsymbol{x}^T \boldsymbol{f}(\boldsymbol{x}) > 0 \qquad \text{for all } \|\boldsymbol{x}\|_2 = R. \tag{3.5.1}$$

If we then choose $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{a}$ for some $\boldsymbol{a}$ with $\|\boldsymbol{a}\|_2 < R$, and the homotopy $\boldsymbol{h}(\boldsymbol{x}, t) = t\, \boldsymbol{f}(\boldsymbol{x}) + (1 - t)\, \boldsymbol{g}(\boldsymbol{x})$ we find that for any $\boldsymbol{x}$ with $\|\boldsymbol{x}\|_2 = R$,

$$\boldsymbol{x}^T \boldsymbol{h}(\boldsymbol{x}, t) = t\, \boldsymbol{x}^T \boldsymbol{f}(\boldsymbol{x}) + (1 - t)\, \boldsymbol{x}^T \boldsymbol{g}(\boldsymbol{x}) > 0.$$

It is clear from this that $\boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{0}$ is impossible for $0 \leq t \leq 1$ and $\|\boldsymbol{x}\|_2 = R$. Thus the set $C$ does not intersect $\{ (\boldsymbol{x}, t) \mid \|\boldsymbol{x}\|_2 = R,\ 0 \leq t \leq 1 \}$. That is, the homotopy curve cannot escape the set $\{ \boldsymbol{x} \mid \|\boldsymbol{x}\|_2 \leq R \} \times [0, 1]$. Then $C$ is bounded. Since it is also closed in $\mathbb{R}^{n+1}$, it is compact. Because it is compact, the component containing $(\boldsymbol{x}_0, 0)$ ends at $(\boldsymbol{x}_1, 1)$ and we have a solution.

This happy ending relies on an assumption we have made: that rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t), \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)] = n$ for all $(\boldsymbol{x}, t) \in C$. This is not guaranteed. However, we can make it likely by incorporating an additional variable $\boldsymbol{a} \in \mathbb{R}^n$ and making the homotopy $\boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a})$ dependent on $\boldsymbol{a}$. Choosing $\boldsymbol{a}$ "at random" makes it likely that rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t; \boldsymbol{a})] = n$ for all $(\boldsymbol{x}, t) \in C_{\boldsymbol{a}}$ where

$$C_{\boldsymbol{a}} = \left\{ (\boldsymbol{x}, t) \in \mathbb{R}^n \times [0, 1] \mid \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}) = \boldsymbol{0} \right\}.$$

The key to showing that choosing $\boldsymbol{a}$ "at random" makes it likely (in fact, probability one) that rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t; \boldsymbol{a})] = n$ for all $(\boldsymbol{x}, t) \in C_{\boldsymbol{a}}$ is the Morse–Sard Theorem:

**Theorem 3.12** *(Morse–Sard theorem) If $\boldsymbol{F} \colon \mathbb{R}^m \to \mathbb{R}^p$ has all partial derivatives of order $\leq r$ continuous, then provided $r \geq 1 + \max(m - p, 0)$, the set*

$$\left\{ \boldsymbol{F}(\boldsymbol{x}) \mid \boldsymbol{x} \in \mathbb{R}^m \,\&\, \mathit{rank}\, \nabla \boldsymbol{F}(\boldsymbol{x}) < p \right\}$$

*has zero Lebesgue measure in $\mathbb{R}^p$.*

A proof can be found in either [227] or [92]. An immediate corollary is

**Corollary 3.13** *If $\boldsymbol{h} \colon \mathbb{R}^n \times [0, 1] \to \mathbb{R}^n$ has continuous second derivatives then the set $\boldsymbol{y} \in \mathbb{R}^n$ for which rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)] < n$ for some $(\boldsymbol{x}, t)$ where $\boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{y}$ has measure zero.*

Thus, perturbing the homotopy $\boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{y}$ with small but random $\boldsymbol{y}$ will give the conditions under which the paths in $C'_{(\boldsymbol{y})} = \{ \boldsymbol{h}(\boldsymbol{x}, t) \mid \boldsymbol{h}(\boldsymbol{x}, t) = \boldsymbol{y} \}$ can be followed. A more useful result is the following parameterized version [50]:

**Theorem 3.14** *If $\boldsymbol{h} \colon \mathbb{R}^n \times [0, 1] \times \mathbb{R}^n \to \mathbb{R}^n$ has continuous second derivatives where $\nabla_{\boldsymbol{a}} \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a})$ is invertible for all $\boldsymbol{a}$, then the set of $\boldsymbol{a} \in \mathbb{R}^n$ for which rank $[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t; \boldsymbol{a})] < n$ for some $(\boldsymbol{x}, t) \in C_{\boldsymbol{a}}$ has measure zero.*

Theorem 3.14 can be applied, for example, to

$$\boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}) = t\, \boldsymbol{f}(\boldsymbol{x}) + (1 - t)\, (\boldsymbol{x} - \boldsymbol{a}).$$

For this function $\nabla_{\boldsymbol{a}} \boldsymbol{h}(\boldsymbol{x}, t; \boldsymbol{a}) = (t - 1)\, I$ which is invertible for $t \neq 1$. If $t = 1$ then $\boldsymbol{h}(\boldsymbol{x}, 1; \boldsymbol{a}) = \boldsymbol{f}(\boldsymbol{x})$ which depends completely on our "hard" function $\boldsymbol{f}$. If $\nabla \boldsymbol{f}(\boldsymbol{x}_1)$ is invertible, then we can follow the path all the way to the solution. If $\nabla \boldsymbol{f}(\boldsymbol{x}_1)$ is not invertible, then the situation is more complex, but the issue lies with $\boldsymbol{f}$, not with following paths.

**Algorithm 49** Simple continuation algorithm

```
function contin1(h, ∇_x h, x_0, ε, N)
  for i = 1, 2, ..., N
    x_i ← newton(x ↦ h(x, i/N), x ↦ ∇_x h(x, i/N), x_{i−1}, ε)
  end
  return x_N
end
```

### 3.5.2 Numerical methods to follow paths

The simplest way to follow the path $\boldsymbol{h}(\boldsymbol{x}, t) = 0$ is to subdivide the interval $[0, 1]$ into pieces $[t_i, t_{i+1}]$ with $t_i = i/N$ for $i = 0, 1, 2, \ldots, N$. At each step we use the solution $\boldsymbol{x}_i$ for $\boldsymbol{h}(\boldsymbol{x}, t_i) = \boldsymbol{0}$ as the starting point for Newton's method for solving $\boldsymbol{h}(\boldsymbol{x}, t_{i+1}) = \boldsymbol{0}$ for $\boldsymbol{x}$. Algorithm 49 shows a straightforward implementation of this idea. This uses, for example, Newton's method code in Algorithm 43. In a good implementation, a guarded Newton method (see Algorithm 44) should be used instead to make the method more reliable.

This method can handle certain problems well, but not others. In particular, if the path being followed "bends back", then Algorithm 49 will fail in the sense of losing the path being followed. In some problems, the path does not "bend back". In these cases, Algorithm 49 can work.

To understand the more general case, we suppose that $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]$ has rank $n$. The matrix $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]$ is $n \times (n+1)$. We can take the QR factorization of its transpose:

$$[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]^T = Q\,R = [Q_1,\ \boldsymbol{q}_{n+1}]\begin{bmatrix} R_1 \\ \boldsymbol{0}^T \end{bmatrix} = Q_1 R_1.$$

Note that under the assumption that $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]$ has rank $n$, $R_1$ is invertible and range $Q_1$ is the range of $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]^T$. The vector $\boldsymbol{q}_{n+1}$ is therefore orthogonal to the range of $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]^T$: $\boldsymbol{q}_{n+1}^T[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]^T = \boldsymbol{0}$, or equivalently $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]\boldsymbol{q}_{n+1} = \boldsymbol{0}$. Writing $\boldsymbol{q}_{n+1}^T = [\boldsymbol{v}^T,\ w]$ we see that

$$\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t)\,\boldsymbol{v} + \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)\,w = \boldsymbol{0}.$$

On the other hand, if we parameterize the path by $\boldsymbol{x} = \boldsymbol{x}(s)$ and $t = t(s)$, differentiating

$$\begin{aligned}
\boldsymbol{0} &= \frac{d}{ds}\boldsymbol{h}(\boldsymbol{x}(s), t(s)) \\
&= \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}(s), t(s))\frac{d\boldsymbol{x}}{ds}(s) + \frac{\partial\boldsymbol{h}}{\partial t}(\boldsymbol{x}(s), t(s))\frac{dt}{ds}(s) \\
&= [\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}, t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x}, t)]\begin{bmatrix} d\boldsymbol{x}/ds \\ dt/ds \end{bmatrix},
\end{aligned}$$

so $[d\boldsymbol{x}/ds^T,\ dt/ds]^T$ is in the null space of $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x},t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x},t)]$. Because the rank of $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x},t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x},t)]$ is $n$, the dimension of the null space is one. Thus $[d\boldsymbol{x}/ds^T,\ dt/ds]$ is a multiple of $\boldsymbol{q}_{n+1}^T = [\boldsymbol{v}^T,\ w]$. Furthermore, if we parameterize the path by its arc length in $(\boldsymbol{x},t)$, then $\left\|[d\boldsymbol{x}/ds^T,\ dt/ds]\right\|_2 = 1 = \left\|\boldsymbol{q}_{n+1}\right\|_2 = \left\|[\boldsymbol{v}^T,\ w]\right\|_2$. This leaves us with only the choice of sign to be made: $d\boldsymbol{x}/ds = \pm\boldsymbol{v}$ and $dt/ds = \pm w$ (same sign in both equations). The choice of sign determines whether we move forward in the path, or backward. Initially we must move forward. Setting $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and $t(0) = 0$, we need $dt/ds(0) > 0$. However, if the path bends back, then we will have part of the curve where $dt/ds < 0$. There also must be a point $(\boldsymbol{x}(\widehat{s}), t(\widehat{s}))$ where $dt/ds(\widehat{s}) = 0$. This gives

$$\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}(\widehat{s}), t(\widehat{s}))\,\frac{d\boldsymbol{x}}{ds}(\widehat{s}) = \boldsymbol{0}.$$

But $d\boldsymbol{x}/ds(\widehat{s}) \neq \boldsymbol{0}$ since otherwise $\left\|[d\boldsymbol{x}/ds^T,\ dt/ds]\right\|_2 = 0$ which contradicts the use of arc length for parameterizing the curve. Thus, $\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x}(\widehat{s}), t(\widehat{s}))$ is *not* invertible. This makes Newton's method dangerous to use, or at least slow, near $(\boldsymbol{x}(\widehat{s}), t(\widehat{s}))$.

Computationally there are two challenges we have identified: how to solve $\boldsymbol{h}(\boldsymbol{x},t) = \boldsymbol{0}$ when we are close to the curve, but not through Newton's method, and how to ensure that we keep moving "forward" on the path even where $dt/ds < 0$.

For the first task, we note that $\boldsymbol{h}(\boldsymbol{x},t) = \boldsymbol{0}$ is an under-determined problem, as there are $n+1$ unknowns in $(\boldsymbol{x},t)$ and only $n$ equations. Given a point $(\boldsymbol{x},t)$ we wish to find a correction $(\delta\boldsymbol{x},\delta t)$ that solves the linearization

$$\boldsymbol{h}(\boldsymbol{x},t) + \nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x},t)\,\delta\boldsymbol{x} + \frac{\partial\boldsymbol{h}}{\partial t}(\boldsymbol{x},t)\,\delta t = \boldsymbol{0}.$$

This is an under-determined linear system of equations in $(\delta\boldsymbol{x},\delta t)$. This can be done using the QR factorization of $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x},t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x},t)]^T$. In general, suppose that $A$ is $p\times q$ with $q < p$ so that $A\boldsymbol{z} = \boldsymbol{b}$ is an under-determined linear system. Take the QR factorization of $A^T = QR = [Q_1,\ Q_2]\begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1R_1$. Then $A = R_1^TQ_1^T$. As long as $A$ has rank $q$, $R_1$ is invertible. Then $A\boldsymbol{z} = \boldsymbol{b}$ is equivalent to $R_1^TQ_1^T\boldsymbol{z} = \boldsymbol{b}$ and so $Q_1^T\boldsymbol{z} = R_1^{-T}\boldsymbol{b}$. Writing $\boldsymbol{z} = Q\boldsymbol{u} = Q_1\boldsymbol{u}_1 + Q_2\boldsymbol{u}_2$ we see that $R_1^{-T}\boldsymbol{b} = Q_1^T(Q_1\boldsymbol{u}_1 + Q_2\boldsymbol{u}_2) = \boldsymbol{u}_1$ and $\boldsymbol{u}_2$ is free. If we set $\boldsymbol{u}_2 = \boldsymbol{0}$ then $\boldsymbol{z} = Q_1\boldsymbol{u}_1 = Q_1R_1^{-T}\boldsymbol{b}$. Note that this is not just any solution to $A\boldsymbol{z} = \boldsymbol{b}$: since $\|\boldsymbol{z}\|_2 = \|\boldsymbol{u}\|_2 = \sqrt{\|\boldsymbol{u}_1\|_2^2 + \|\boldsymbol{u}_2\|_2^2} \geq \|\boldsymbol{u}_1\|_2$ this choice $\boldsymbol{z} = Q_1\boldsymbol{u}_1 = Q_1R_1^{-T}\boldsymbol{b}$ is the smallest solution in the 2-norm. Applied to our linearized equation for $(\delta\boldsymbol{x},\delta t)$ we see that this approach gives us the smallest value of $\|(\delta\boldsymbol{x},\delta t)\|_2$ that solves the equation. The updates $\boldsymbol{x}^+ \leftarrow \boldsymbol{x} + \delta\boldsymbol{x}, t^+ \leftarrow t + \delta t$ give the nearest update $(\boldsymbol{x}^+, t^+)$ to the original $(\boldsymbol{x},t)$ that satisfies the equations. Repeating these corrections in a Newton or guarded Newton way gives us a way to solve under-determined systems of equations. Often we leave out updating the Jacobian matrix $[\nabla_{\boldsymbol{x}}\boldsymbol{h}(\boldsymbol{x},t),\ \partial\boldsymbol{h}/\partial t(\boldsymbol{x},t)]$ after updates to $(\boldsymbol{x},t)$ if the initial guess is close to a solution. This avoids the expense

of the QR factorization at each iteration, while still giving a good (if linear) rate of convergence per iteration. An implementation that recomputes and refactors the Jacobian matrix at each iteration is shown in Algorithm 50.

The QR factorization is also key in the second computational task: finding which direction to continue along, $+\boldsymbol{q}_{n+1}$ or $-\boldsymbol{q}_{n+1}$. Using

$$[\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)]^T = QR = [Q_1,\ \boldsymbol{q}_{n+1}] \begin{bmatrix} R_1 \\ \boldsymbol{0}^T \end{bmatrix},$$

we can add an extra column to get (recall that $\boldsymbol{q}_{n+1} = [\boldsymbol{v}^T,\ w]^T$)

$$\begin{bmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t)^T & \boldsymbol{v} \\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)^T & w \end{bmatrix} = [Q_1,\ \boldsymbol{q}_{n+1}] \begin{bmatrix} R_1 & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix}.$$

The right-hand side is invertible since $Q = [Q_1,\ \boldsymbol{q}_{n+1}]$ is orthogonal, and $\begin{bmatrix} R_1 & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$ is invertible as $R_1$ is invertible. Note that $\boldsymbol{v} = \pm d\boldsymbol{x}/ds$ and $w = \pm dt/ds$ (same choice of sign in both equalities). Thus,

$$\det \begin{bmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t)^T & \boldsymbol{v} \\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)^T & w \end{bmatrix} = \pm \det \begin{bmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t)^T & d\boldsymbol{x}/ds \\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)^T & dt/ds \end{bmatrix} \neq 0.$$

Assuming all quantities continuously differentiable,

$$\det \begin{bmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t)^T & d\boldsymbol{x}/ds \\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)^T & dt/ds \end{bmatrix}$$

is continuous, real, and never zero. Thus its sign must remain constant. We can compute the sign of this determinant $\sigma_0$ at $(\boldsymbol{x}, t) = (\boldsymbol{x}_0, 0)$, so we can use the sign of $\det \begin{bmatrix} \nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t)^T & \boldsymbol{v} \\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)^T & w \end{bmatrix}$ to determine the choice of sign for $\boldsymbol{v} = \pm d\boldsymbol{x}/ds$ and $w = \pm dt/ds$.

The sign of $\det Q$ where $Q$ is the orthogonal factor of a QR factorization can usually be computed easily. If $Q$ is, for example, a product of Householder reflectors each Householder reflector contributes a factor of $(-1)$ to the product. The sign of the determinant of $R_1$ is simply the parity of the number of negative diagonal elements. Then $\boldsymbol{v} = \sigma\, d\boldsymbol{x}/ds$ and $w = \sigma\, dt/ds$ where $\sigma = \sigma_0 \operatorname{sign}(\det Q) \operatorname{sign}(\det R_1)$.

An alternative approach is to keep the directions $[d\boldsymbol{x}/ds^T,\ dt/ds]$ consistent as $s$ changes. More specifically, if $s \approx s'$ then we expect

$$\begin{bmatrix} d\boldsymbol{x}/ds(s) \\ dt/ds(s) \end{bmatrix} \approx \begin{bmatrix} d\boldsymbol{x}/ds(s') \\ dt/ds(s') \end{bmatrix} \quad \text{so} \quad \begin{bmatrix} d\boldsymbol{x}/ds(s) \\ dt/ds(s) \end{bmatrix}^T \begin{bmatrix} d\boldsymbol{x}/ds(s') \\ dt/ds(s') \end{bmatrix} > 0.$$

**Algorithm 50** Guarded Newton method for under-determined systems

```
function gudnewton(f, ∇f, x_0, ε)
  k ← 0
  while ‖f(x_k)‖_2 > ε
    A ← ∇f(x_k); A = QR (QR factorization)
    split Q = [Q_1, Q_2]; R = [R_1; 0] so A = Q_1 R_1
    d_k ← −Q_1 R_1^{−T} f(x_k)
    α ← 1
    while ‖f(x_k + α d_k)‖_2 > (1 − ½α) ‖f(x_k)‖_2
      α ← α/2
    end while
    x_{k+1} ← x_k + α d_k; k ← k + 1
  end while
  return x_k
end function
```

**Algorithm 51** Homotopy algorithm (uses Algorithm 50)

```
function homotopy(h, ∇h, x_0, δs, ε, η)
  t ← 0; s ← 0; x ← x_0 // assume h(x_0, 0) = 0
  oldq ← 0
  A ← ∇h(x, t); A = QR // QR factorization
  split Q = [Q_1, q], R = [R_1; 0^T]
  if q_{n+1} < 0: q ← −q
  oldq ← q
  while t < 1
    if t + δs w > 1: δs ← (1 − t)/w; end if
    x^+ ← x + δs v; t^+ ← t + δs w
    A ← ∇h(x^+, t^+); A = QR // QR factorization
    split Q = [Q_1, q], R = [R_1; 0^T]
    if q^T oldq < 0: q ← −q; end if
    if ∠(q, oldq) > η: δs ← δs/2; continue
    (x^+, t^+) ← gudnewton(h, ∇h, (x^+, t^+), ε)
    (x, t) ← (x^+, t^+)
    oldq ← q
  end while
  x ← newton(f, ∇f, x, ε)
  return x
end
```

This gives a simpler approach to controlling how quickly to increment $s_k$ to $s_{k+1}$: ensure that the angle between $[d\boldsymbol{x}/ds(s_k)^T,\ dt/ds(s_k)]$ and $[d\boldsymbol{x}/ds(s_{k+1}),\ dt/ds(s_{k+1})]$ never exceeds a small user-specified threshold $\eta > 0$. If the threshold is exceeded, go back to $s_k$, reduce the increment $\delta s$ (say by halving it), then setting $s_{k+1} \leftarrow s_k + \delta s$ and try again with the new value of $s_{k+1}$.

In Algorithm 51,$\nabla \boldsymbol{h}(\boldsymbol{x}, t) = [\nabla_{\boldsymbol{x}} \boldsymbol{h}(\boldsymbol{x}, t),\ \partial \boldsymbol{h}/\partial t(\boldsymbol{x}, t)]$.

Details on how to implement homotopy algorithms can be found in the book by Allgower and Georg [4]. The package HOMPACK [257] developed by Watson et al. implements continuation algorithms.

**Exercises.**

(1) Brouwer's fixed-point theorem states that if $B = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid \|\boldsymbol{x}\|_2 \le 1 \,\}$ and $\boldsymbol{f} : \mathbb{R}^n \to B$ is continuous, then there is a fixed point $\boldsymbol{x}^* \in B$ where $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{x}^*$. Show that this is true for $\boldsymbol{f}$ smooth. [**Hint:** Let $\boldsymbol{h}(t, \boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{x} - [(1-t)\boldsymbol{a} + t\,\boldsymbol{f}(\boldsymbol{x})]$. Then provided $t \neq 1$, $\nabla_{\boldsymbol{a}} \boldsymbol{h}(t, \boldsymbol{x}, \boldsymbol{a}) = (t-1)I$ is invertible. Then by Theorem 3.14, for almost all $\boldsymbol{a}$ in the interior of $B$, the solution to $\boldsymbol{h}(t, \boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{0}$ is a union of smooth closed curves. Starting at $\boldsymbol{x} = \boldsymbol{a}$ for $t = 0$ we continue on the curve $C := \{\, (\boldsymbol{x}, t) \mid \boldsymbol{h}(t, \boldsymbol{x}, \boldsymbol{a}) = \boldsymbol{0} \,\}$ until we hit a boundary: either $\boldsymbol{x} \in \partial B$ or $t = 0$ or $t = 1$. We cannot hit $\boldsymbol{x} \in \partial B$ because $\|\boldsymbol{a}\|_2 < 1$. Show that we cannot hit $t = 0$ because this would mean $C$ is tangent to $t = 0$ at $(0, \boldsymbol{a})$. Thus we approach $t = 1$. Use compactness of $B$ to show that there is a limit point $(1, \boldsymbol{x}^*)$ where $\boldsymbol{x}^*$ is a fixed point of $\boldsymbol{f}$.]

(2) Suppose that $f : \mathbb{C} \to \mathbb{C}$ is an analytic function so that $df/dz(z)$ always exists as a complex number for all $z \in \mathbb{C}$. Treat $f$ as a function $\mathbb{R}^2 \to \mathbb{R}^2$, with $f(z) = u(x, y) + i\,v(x, y)$ where $z = x + iy$ and $u,\ v,\ x,\ y$ all real. Show that $\det \nabla f(x, y) \ge 0$ with equality only if $f'(z) = 0$. For a homotopy $h(t, z) = t\,f(z) + (1-t)\,p(z)$, where both $f$ and $p$ are analytic, show that the homotopy curve $\{\, (t, z) \mid h(t, z) = 0 \,\}$ with random choice of $p(z)$ does not "turn back". [**Hint:** If we write $(t, z) = (t(s),\ z(s))$ as smooth functions of the arclength $s$, then $dt/ds = 0$ implies that $\partial h/\partial z(t, z) = 0$. Use the parameterized Sard theorem to show that with probability one $\partial h/\partial z(t, z) \neq 0$ on the homotopy curve.]

(3) We can use homotopy methods to find all complex roots of a polynomial: if $f(z)$ is a polynomial of degree $d$ with leading coefficient one, set $p(z) = \prod_{k=1}^{d} (z - r_k)$ with randomly chosen $r_k \in \mathbb{C}$. Show that the homotopy curve

$$\{\, (t, z) \mid t\,f(z) + (1-t)p(z) = 0,\ 0 \le t \le 1 \,\}$$

is bounded. [**Hint:** The coefficient of $z^d$ in $t\,f(z) + (1-t)\,p(z)$ is one. The other coefficients for $0 \le t \le 1$ are bounded.]

(4) Use HOMPACK or some other implementation of homotopy methods to solve $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{0}$ where

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} e^{x+y} - \cos x + z^3 - 3 \\ \sin z + xy + 2 \\ z^3 - z(x+y) + x^3 - 7 \end{bmatrix}, \qquad \boldsymbol{x} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}.$$

(5) The package AUTO by Eusebius Doedel [77, 78] is designed for *parameter* continuation, that is, it uses homotopy methods to follow solutions as user-specified parameters are changed. Thus, it cannot assume that $\nabla \boldsymbol{h}(t, \boldsymbol{x})$ has full

rank, and must occasionally deal with bifurcations and other singular behavior. Read the documentation for AUTO and describe how it works.

(6) Consider the homotopy $h(t, x) = x^3/3 - x + (6t - 3)$ with $0 \le t \le 1$. At $t = 0$ we start with a negative solution, and end with a positive solution at $t = 1$. Explain why the "turning back" that occurs in this homotopy (twice) means that applying the Newton method just to fix errors in $x$, rather than working with $(t, x)$ together, will have difficulty.

(7) Consider solving the equations $f(x, y) = 0$ and $g(x, y) = 0$ with $f$ and $g$ smooth on $(x, y) \in [a, b] \times [c, d]$. Assume that if $f(x, c) < 0$ and $f(x, d) > 0$ for all $x \in [a, b]$, and $g(a, y) < 0$ and $g(b, y) > 0$ for all $y \in [c, d]$. Show that there must be a solution $(x^*, y^*) \in [a, b] \times [c, d]$. [**Hint:** Let $\boldsymbol{x} = [x, y]^T$, $\boldsymbol{f}(\boldsymbol{x}) = [f(x, y),\ g(x, y)]^T$, and $\boldsymbol{k}(\boldsymbol{x}) = [x - \frac{1}{2}(a + b),\ y - \frac{1}{2}(c + d)]$. Define the homotopy $\boldsymbol{h}(\boldsymbol{x}, \lambda) = (1 - \lambda)\boldsymbol{k}(\boldsymbol{x}) + \lambda\, \boldsymbol{f}(\boldsymbol{x})$. Show that $\boldsymbol{h}(\boldsymbol{x}, \lambda) = \boldsymbol{0}$ has no solutions for $0 \le \lambda \le 1$ and $\boldsymbol{x}$ on the boundary of the rectangle $[a, b] \times [c, d]$. Use Sard's theorem to show that $\{\, \boldsymbol{x} \mid \boldsymbol{h}(\boldsymbol{x}, \lambda) = \boldsymbol{s} \,\}$ consists of smooth curves for almost all $\boldsymbol{s}$. Take a sequence $\boldsymbol{y}_k \to \boldsymbol{0}$ as $k \to \infty$ where this solution set property is true for all $\boldsymbol{y}_k$. Use compactness to $[a, b] \times [c, d]$ to show the existence of a solution $\boldsymbol{x}^*$ of $\boldsymbol{f}(\boldsymbol{x}^*) = \boldsymbol{0}$.]

(8) Generalize Exercise 7 for rectangles $R = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n] \subset \mathbb{R}^n$ and functions $f_i \colon \mathbb{R}^n \to \mathbb{R}$ for $i = 1, 2, \ldots, n$ with $f_i(\boldsymbol{x}) < 0$ if $x_i = a_i$ and $f_i(\boldsymbol{x}) > 0$ if $x_i = b_i$ for all $\boldsymbol{x} \in R$.

(9) As $\lambda(s)$ approaches one, it is possible that we could have a turning point at $\lambda(s^*) = 1$. In this case, the curve $(\boldsymbol{x}(s), \lambda(s)) \to (\widehat{\boldsymbol{x}}, 1)$ as $s \to s^*$ where $\nabla \boldsymbol{h}(\widehat{\boldsymbol{x}}, 1)$ is *not* invertible. In this case, using Newton's method on $\boldsymbol{x} \mapsto \boldsymbol{h}(\boldsymbol{x}, 1)$ is unlikely to work well. An alternative approach is to use quadratic interpolation on $(\boldsymbol{x}(s_0), \lambda(s_0))$, $(\boldsymbol{x}(s_1), \lambda(s_1))$, $(\boldsymbol{x}(s_2), \lambda(s_2))$ with $s_0 < s_1 < s_2 < s^*$ and use this interpolant to estimate $(\widehat{\boldsymbol{x}}, 1)$. Implement a scheme of this type, taking care to make sure that the method is robust to small perturbations. Test this on the homotopy $h(x, \lambda) = \lambda x^2 + (1 - \lambda)(2 + x - e^x)$.

(10) An application of Brouwer's fixed-point theorem (Exercise 1) is the Perron–Frobenius theorem [20]: Any $n \times n$ matrix $A$ with $a_{ij} > 0$ for all $i$, $j$ has positive eigenvalue $\lambda$ with a positive eigenvector $\boldsymbol{v}$. Prove this using the map $\boldsymbol{x} \mapsto A\boldsymbol{x}/\|A\boldsymbol{x}\|_1$ of the unit simplex $\Sigma = \{\, \boldsymbol{x} \in \mathbb{R}^n \mid x_i \ge 0 \text{ for all } i, \text{ and } \sum_{i=1}^n x_i = 1 \,\}$ onto itself. [**Hint:** First show that the map takes $\Sigma \to \Sigma$. Then the Brouwer fixed-point theorem implies that this map has a fixed point. The fixed point is the desired eigenvector.] From this result show that any other eigenvalue $\mu$ of $A$ satisfies $|\mu| < \lambda$.

## Project

Consider the problem of determining the steady flow through a network of pipes. This network is considered to be a directed graph $G = (V, E)$ without cycles where each edge $e \in E$ of the graph has a start vertex $\text{start}(e) \in V$ and an end vertex

end$(e) \in V$. We can represent this graph by a sparse matrix $B$ where $b_{e,x} = +1$ if $x = \text{end}(e)$, $-1$ if $x = \text{start}(e)$, and zero if $x$ is neither a start nor end vertex of $e$. The flows are generated by pressure differences: the flow: $q_e = f_e(p_x - p_y)$ where $x = \text{start}(e)$ and $y = \text{end}(e)$ with the functions $f_e$ given. There is a set of source nodes $\mathcal{S}$ and a set of destination nodes $\mathcal{D}$. We assume that $f_e(0) = 0$ and are smooth and increasing functions. Conservation of mass means that for any node $x \notin \mathcal{S} \cup \mathcal{D}$, the net flow into $x$ is zero: $\sum_e b_{e,x} q_e = 0$. If $x \in \mathcal{S}$ is a source node then there is a flow $\widehat{q}_x$ being injected into this node, which must then move through the network of pipes: $\widehat{q}_x + \sum_e b_{e,x} q_e = 0$; similarly, if $x \in \mathcal{D}$ is a destination node, then there is a flow $-\widehat{q}_x$ being removed from node $x$; the conservation equation at $x$ can also be written as $\widehat{q}_x + \sum_e b_{e,x} q_e = 0$. Set the pressure at one node to be zero. Also, from conservation of the material flowing in the network, the total inflow must equal the total outflow: $\sum_{x \in \mathcal{S}} \widehat{q}_x + \sum_{x \in \mathcal{D}} \widehat{q}_x = 0$. So one inflow or outflow variable should be left "free". This is implemented by removing the equation $\widehat{q}_x + \sum_e b_{e,x} q_e = 0$ for that node from the system of equations to be solved. (This could be the same node where you set $p_x = 0$.)

Set up the nonlinear equations to solve for the steady state of a network of this type. The variables are the flows $q_e$ and the pressures $p_x$ inside the network. The inflow quantities $\widehat{q}_x$ for source vertices $x \in \mathcal{S}$ and outflow quantities $-\widehat{q}_x$ for destination vertices $x \in \mathcal{D}$ are given. Set up a guarded Newton method for solving this system of equations. Implement your method in a suitable programming language. Can you show that the Jacobian matrix for your linear system is invertible?

As a test problem, solve the problem shown in Figure 3.5.2. For this test use the table of flow functions $f_e(\Delta p) = \alpha_e \, \Delta p + \beta_e \, \Delta p / \sqrt{\gamma_e^2 + (\Delta p)^2}$ given by the parameters in Table 3.5.1.

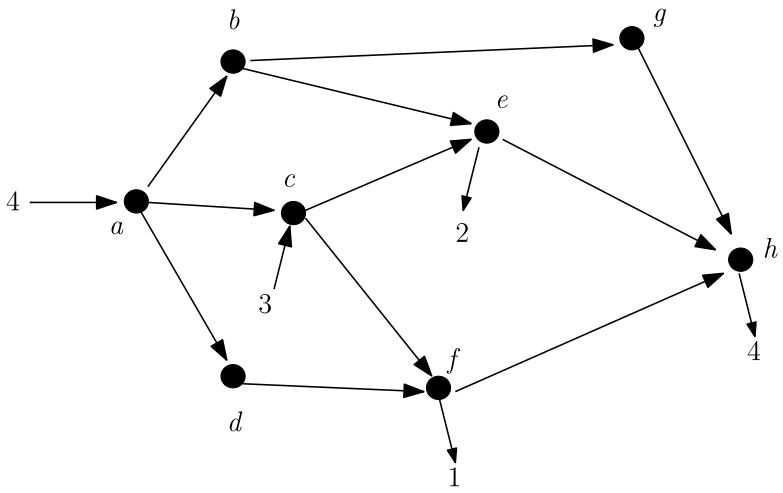



**Fig. 3.5.2** Network of pipes—test problem

**Table 3.5.1** Network of pipes—test problem

| edge ($e$) | $\alpha_e$ | $\beta_e$ | $\gamma_e$ | edge ($e$) | $\alpha_e$ | $\beta_e$ | $\gamma_e$ | edge ($e$) | $\alpha_e$ | $\beta_e$ | $\gamma_e$ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $a \to b$ | 1 | 0 | – | $b \to g$ | 1 | 0 | – | $e \to h$ | 0 | 1 | 1 |
| $a \to c$ | 1 | 1 | 2 | $c \to e$ | 0 | 2 | 1 | $f \to h$ | 1 | 2 | 1 |
| $a \to d$ | 2 | 1 | 1 | $c \to f$ | 2 | 1 | 2 | $g \to h$ | 2 | 2 | 2 |
| $b \to e$ | 1 | 2 | 1 | $d \to f$ | 0 | 1 | 1 | — | – | – | – |

An alternative formulation is to write the pressure difference as a function of the flow through an edge: $p_y - p_x = g_e(q_e)$ where $y = \text{end}(e)$ and $x = \text{start}(e)$. Re-write your system of equations to use the functions $g_e$ instead of $f_e$.

# Chapter 4
# Approximations and Interpolation


The representation and approximation of functions is central to the practice of numerical analysis and scientific computation. The oldest of these is Taylor series developed by James Gregory and later Brook Taylor. However, the development of Taylor series requires knowledge of derivatives of arbitrarily high order. Interpolation instead only requires function values, and/or a few low-order derivatives at interpolation points.

Usually these methods use polynomials as the interpolating functions, although trigonometric functions as the approximating functions are especially useful in methods based on Fourier series, and piecewise polynomials such as splines have properties that make them robust and practical tools.

How well a given function can be approximated by simpler functions in one of these classes is an essential question for many applications. The quality of an approximation can be measured in different ways. The most commonly used measures of closeness of approximation are the $\infty$-norm (or max-norm) to describe worst-case errors, and the 2-norm for least squares approximation. Multivariate approximation properties are especially important in application to partial differential equations. Modern approximation theory has been applied to questions arising in data science and machine learning, where multivariate functions (often functions of many variables) need to be approximated using sparse data sets.

## 4.1 Interpolation—Polynomials

Interpolation is the task, given a collection of data points $(\boldsymbol{x}_i, y_i)$, $i = 0, 1, 2, \ldots, n$, to find a function $p$ in a specified class $\mathcal{F}$ where $p(\boldsymbol{x}_i) = y_i$ for $i = 0, 1, 2, \ldots, n$. Usually $\mathcal{F}$ is a vector space of functions, such as the set of all polynomials of degree $\leq d$. Ideally, the dimension of $\mathcal{F}$ is equal to the number of data points, and the solution exists and is unique. Other classes of functions can be used, such as



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_4

piecewise polynomials like cubic splines, or trigonometric functions $\cos(kx)$ and $\sin(kx)$ for integer values of $k$.

### 4.1.1 Polynomial Interpolation in One Variable

For polynomial interpolation in one variable, given data points $(x_i, y_i)$, $i = 0, 1, 2, \ldots, n$ we want to find a polynomial of degree $\leq d$ where

$$p(x_i) = y_i \qquad \text{for } i = 0, 1, 2, \ldots, n. \tag{4.1.1}$$

In order to have as many equations as unknowns, we note that the number of coefficients of $p(x)$ $(d + 1)$ should be equal to the number of data points $(n + 1)$. That is, we want $d + 1 = n + 1$ so $d = n$. This condition is necessary but not sufficient in order to guarantee existence and uniqueness of the solution.

The simplest non-trivial case is with $n = d = 1$, so we want to have a linear function $p(x)$ going through two data points. Provided the two data points are distinct, there is exactly one straight line going through these points. Provided the data points have different $x$-coordinates $(x_0 \neq x_1)$, there is exactly one linear function going through these data points—the straight line through the data points is not vertical. Linear interpolation has ancient roots—it was used by ancient Babylonians, and by ancient Greeks from about 300 BCE, for astronomical calculations. Chinese and Indian astronomers used quadratic interpolation in the seventh century, and higher order interpolation during the thirteenth and fourteenth centuries. Usually they used interpolation for tables of trigonometric functions to get accurate values "between the lines" of the values given.

Interpolating values in tables of values are no longer so important, but there are many ways in which interpolation is a central topic in numerical analysis. At heart, interpolation allows us to represent a function (approximately) in terms of a few data points. Operations that we would perform on the function, such as integration and differentiation, can be represented in terms of the values at the interpolation points.

#### 4.1.1.1 Existence and Uniqueness

For higher order polynomial interpolation, the crucial condition we need is that none of the $x_i$'s are repeated, that is, we need the interpolation points $x_0, x_1, \ldots, x_n$ to be distinct: if $x_i = x_j$ but $i \neq j$, then we clearly need $y_i = p(x_i) = p(x_j) = y_j$ in order for an interpolant $p$ to exist. In particular, if $x_i = x_j$ but $y_i \neq y_j$, there is no interpolant of the data.

Even if an interpolant exists for repeated interpolation points, the number of equations to be satisfied is reduced by one, so the solution cannot be expected to be unique. But if the interpolation points $x_0, x_1, \ldots, x_n$ are distinct, then there is exactly one interpolant $p(x)$ of degree $\leq n$.

**Theorem 4.1** *If $(x_i, y_i)$, $i = 0, 1, 2, \ldots, n$ and $x_i \neq x_j$ for $i \neq j$, then there is one and only one polynomial $p(x)$ of degree $\leq n$ where*

$$p(x_i) = y_i \quad \text{for all } i = 0, 1, 2, \ldots, n.$$

***Proof*** We represent a polynomial of degree $p(x)$ in terms of the vector of its coefficients $\boldsymbol{a} \in \mathbb{R}^{n+1}$:

$$p(x; \boldsymbol{a}) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n.$$

Clearly, $p(x; \boldsymbol{a})$ is a linear function of $\boldsymbol{a}$. Then the function

$$\boldsymbol{F} : \boldsymbol{a} \mapsto \begin{bmatrix} p(x_0; \boldsymbol{a}) \\ p(x_1; \boldsymbol{a}) \\ \vdots \\ p(x_n; \boldsymbol{a}) \end{bmatrix} \in \mathbb{R}^{n+1}$$

is a linear transformation $\boldsymbol{F} : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1}$. So, this linear function is onto if and only if it is one to one. In particular, we want to show that the only $\boldsymbol{a}$ for which $\boldsymbol{F}(\boldsymbol{a}) = \boldsymbol{0}$ is $\boldsymbol{a} = \boldsymbol{0}$.

So suppose that $\boldsymbol{a}^*$ satisfies $\boldsymbol{F}(\boldsymbol{a}^*) = \boldsymbol{0}$. Then the polynomial $q(x) = p(x; \boldsymbol{a}^*)$ is a polynomial of degree $\leq n$ where $q(x_i) = 0$ for $i = 0, 1, 2, \ldots, n$. Then $q(x)$ can be factored

$$q(x) = (x - x_0)(x - x_1) \cdots (x - x_n)\, g(x)$$

for some polynomial $g$, provided all $x_i$'s are distinct. This implies that the degree of $q$ is $(\deg g) + (n + 1)$. The only way this can be no more than $n$ is if $\deg g < 0$ which only occurs if $g$ is the zero polynomial. That is, $q(x)$ must be the zero polynomial, and hence all its coefficients must be zero. That is, $\boldsymbol{a}^* = \boldsymbol{0}$ as we wanted. Thus, the linear transformation $\boldsymbol{F}$ is one to one and so $\boldsymbol{F}$ is onto, that is, there is exactly one solution $\boldsymbol{a}$ to $\boldsymbol{F}(\boldsymbol{a}) = \boldsymbol{y}$ for any $\boldsymbol{y}$. This solution gives the coefficients of the unique interpolating polynomial. □

#### 4.1.1.2 Computing the Polynomial Interpolant

The next task is computing the interpolant. Since the equations to be satisfied are linear equations in the coefficients, we can solve these equations directly:

$$V_n \boldsymbol{a} = \begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{4.1.2}$$

**Table 4.1.1** Condition numbers $\kappa_2(V_n)$ of Vandermonde matrices

| $n$ | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| $\kappa_2(V_n)$ | $1.51 \times 10^1$ | $9.89 \times 10^1$ | $6.86 \times 10^2$ | $4.92 \times 10^3$ | $3.61 \times 10^4$ | $2.68 \times 10^5$ |
| $n$ | 8 | 9 | 10 | 11 | 12 | 13 |
| $\kappa_2(V_n)$ | $2.01 \times 10^6$ | $1.52 \times 10^7$ | $1.16 \times 10^8$ | $8.83 \times 10^8$ | $6.78 \times 10^9$ | $5.22 \times 10^{10}$ |

The matrix $V_n$ is called a *Vandermonde matrix*. By Theorem 4.1, this matrix must be invertible, provided all the $x_i$'s are distinct. A more direct proof is the calculation that its determinant is $\prod_{i<j}(x_i - x_j)$.

While it is possible to compute the coefficients of the interpolating polynomial this way, it is not recommended. The reason is that the condition number of $V_n$ can easily become very large. For example, if we take $x_i = i/n$ for $i = 0, 1, 2, \ldots, n$, the 2-norm condition numbers are shown in Table 4.1.1.

The condition number of Vandermonde matrices grows exponentially in $n$ [72].

Because of the explosive growth of the condition numbers of Vandermonde matrices, other methods have been devised that have avoided both the computational cost and ill-conditioning of directly solving for the coefficients.

### 4.1.1.3 Lagrange Interpolation Polynomials

While Joseph-Louise Lagrange in 1795 got the title credits for the polynomials used to create interpolants

$$L_i(x) = \prod_{j:j\neq i} \left( \frac{x - x_j}{x_i - x_j} \right), \tag{4.1.3}$$

the formulas were published earlier by Edward Waring (1779). The interpolant $p(x)$ of the data $p(x_i) = y_i$ for $i = 0, 1, 2, \ldots, n$ can be written as

$$p(x) = \sum_{i=0}^{n} y_i \, L_i(x). \tag{4.1.4}$$

The reason for the correctness of these formulas is that

$$L_i(x_k) = \prod_{j:j\neq i} \left( \frac{x_k - x_j}{x_i - x_j} \right) = 0 \qquad \text{if } k \neq i, \text{ and}$$

$$L_i(x_i) = \prod_{j:j\neq i} \left( \frac{x_i - x_j}{x_i - x_j} \right) = 1, \qquad \text{so}$$

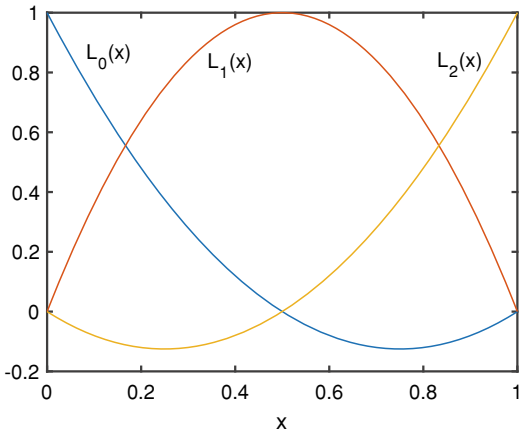



**Fig. 4.1.1** Lagrange interpolation polynomials for $x_0 = 0$, $x_1 = \frac{1}{2}$, $x_2 = 1$

$$p(x_k) = \sum_{i=0}^{n} y_i L_i(x_k) = \sum_{i=0}^{n} y_i \begin{cases} 1, & \text{if } k = i \\ 0, & \text{if } k \neq i \end{cases} = y_k.$$

Directly computing $L_i(x)$ from this formula takes $\approx 4n$ flops; thus computing $p(x)$ would take $\approx 4n^2$ flops via (4.1.3). This compares well with $\sim \frac{2}{3}n^3$ flops for solving the Vandermonde equations. However, this is not the fastest method for computing the value of an interpolating polynomial. In spite of this, Lagrange interpolation polynomials provide a convenient means of representing the polynomial interpolant of a given degree, and we will make use of this in later sections.

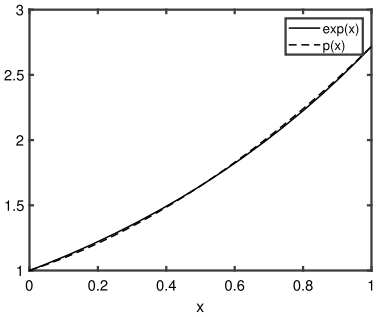
For example, if $x_0 = 0$, $x_1 = 1/2$, and $x_2 = 1$, then

$$\begin{aligned} L_0(x) &= \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} = 2(x - \frac{1}{2})(x - 1), \\ L_1(x) &= \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} = -4x(x - 1) = 4x(1 - x), \\ L_2(x) &= \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} = 2x(x - \frac{1}{2}). \end{aligned}$$

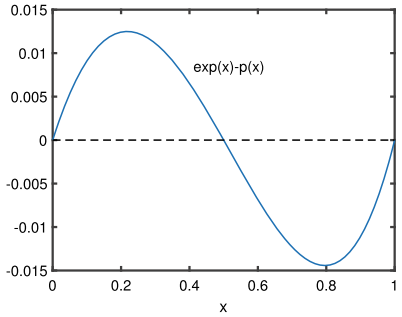
Note that $L_2(x) = L_0(1 - x)$ due to the symmetry of the placement of the interpolation points. Plots of the three quadratic Lagrange interpolation polynomials are shown in Figure 4.1.1.

The quadratic interpolant of $(x_0, y_0)$, $(x_1, y_1)$, and $(x_2, y_2)$ is then

$$\begin{aligned} p(x) &= y_0 L_0(x) + y_1 L_1(x) + y_2 L_2(x) \\ &= 2y_0(x - \frac{1}{2})(x - 1) + 4y_1 x(1 - x) + 2y_2 x(x - \frac{1}{2}). \end{aligned}$$




(A) Plots of $p(x)$ & $e^x$




(B) Plot of $e^x - p(x)$

**Fig. 4.1.2** Comparison between quadratic interpolant $p(x)$ and $f(x) = e^x$

If the $y_i$ values come from some function: $y_i = f(x_i)$, then we can use $p(x)$ as a quadratic approximation to the function $f(x)$. For example, if $f(x) = e^x$ then the interpolating quadratic is

$$p(x) = 2(x - \frac{1}{2})(x - 1) + 4e^{1/2}x(1 - x) + 2ex(x - \frac{1}{2}).$$

A plot showing a comparison between $p(x)$ and $f(x)$, as well as a plot of the difference is shown in Figure 4.1.2.

#### 4.1.1.4 Divided Differences

In a letter from 1675, Isaac Newton described a method of interpolating equally spaced data [96]. That method is basis of the modern divided difference method, which allows non-equally spaced data. The starting point for this method is the definition of divided differences. Firstly, for first-order divided differences,

$$f[x_0, x_1] = \frac{f(x_0) - f(x_1)}{x_0 - x_1} \qquad \text{for } x_0 \neq x_1,$$

and continues to higher orders by the formula

$$f[x_0, x_1, \ldots, x_n] = \frac{f[x_0, x_1, \ldots, x_{n-1}] - f[x_1, x_2, \ldots, x_n]}{x_0 - x_n}$$

provided all $x_i$'s are distinct. For example,

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2} \\ &= \frac{(f(x_0) - f(x_1))/(x_0 - x_1) - (f(x_1) - f(x_2))/(x_1 - x_2)}{x_0 - x_2}. \end{aligned}$$

#### 4.1.1.5 Independence of Ordering

The first-order divided differences do not depend on the order of the inputs:

$$f[x_0, x_1] = \frac{f(x_0) - f(x_1)}{x_0 - x_1} = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_1, x_0].$$

This works more generally for higher order divided differences: the order of the $x_i$'s in the divided difference does not matter.

**Theorem 4.2** *If $y_0, y_1, \ldots, y_n$ is any permutation of $x_0, x_1 \ldots, x_n$ and all $x_i$'s are distinct, then $f[x_0, x_1, \ldots, x_n] = f[y_0, y_1, \ldots, y_n]$.*

***Proof*** We show by induction on $n$ that

$$f[x_0, x_1, \ldots, x_n] = \sum_{i=0}^{n} \frac{f(x_i)}{\prod_{j=0,\ j\neq i}^{n}(x_i - x_j)}.$$

Once we have proven that this formula holds, then it is obvious that the order does not matter.

This formula holds for $n = 1$, since

$$\begin{aligned} f[x_0, x_1] &= \frac{f(x_0) - f(x_1)}{x_0 - x_1} \\ &= \frac{f(x_0)}{x_0 - x_1} + \frac{f(x_1)}{x_1 - x_0}. \end{aligned}$$

Now for induction, we suppose that the formula holds for $n = k$, and we want to show that it holds for $n = k - 1$. Now consider

$$\begin{aligned} &f[x_0, x_1, \ldots, x_{k-1}, x_k] \\ &\quad = \frac{f[x_0, x_1, \ldots, x_{k-1}] - f[x_1, x_2, \ldots, x_k]}{x_0 - x_k} \\ &\quad = \frac{1}{x_0 - x_k}\left[\sum_{i=0}^{k-1} \frac{f(x_i)}{\prod_{j=0,\ j\neq i}^{k-1}(x_i - x_j)} - \sum_{i=1}^{k} \frac{f(x_i)}{\prod_{j=1,\ j\neq i}^{k}(x_i - x_j)}\right] \\ &\quad = \frac{1}{x_0 - x_k}\left[\frac{f(x_0)}{\prod_{j=0,\ j\neq 0}^{k-1}(x_0 - x_j)}\right. \\ &\qquad + \sum_{i=0}^{k-1} f(x_i)\left(\frac{1}{\prod_{j=0,\ j\neq i}^{k-1}(x_i - x_j)} - \frac{1}{\prod_{j=1,\ j\neq i}^{k}(x_i - x_j)}\right) \\ &\qquad \left. - \frac{f(x_k)}{\prod_{j=1,\ j\neq k}^{k}(x_k - x_j)}\right]. \end{aligned}$$

The first and last terms are good since they are $f(x_0)/\prod_{j=0,\ j\neq 0}^{k}(x_0 - x_j)$ and $f(x_k)/\prod_{j=0,\ j\neq k}^{k}(x_k - x_j)$. For the middle term, we need to concentrate on the quantity inside the parentheses $(\cdots)$: Looking for common factors we find (noting that $0 < i < k$)

$$\begin{aligned}
&\frac{1}{\prod_{j=0,\ j\neq i}^{k-1}(x_i - x_j)} - \frac{1}{\prod_{j=1,\ j\neq i}^{k}(x_i - x_j)} \\
&\quad = \frac{1}{\prod_{j=1,\ j\neq i}^{k-1}(x_i - x_j)}\left[\frac{1}{x_i - x_0} - \frac{1}{x_i - x_k}\right] \\
&\quad = \frac{1}{\prod_{j=1,\ j\neq i}^{k-1}(x_i - x_j)}\,\frac{(x_i - x_k) - (x_i - x_0)}{(x_i - x_0)(x_i - x_k)} \\
&\quad = \frac{x_0 - x_k}{\prod_{j=0,\ j\neq i}^{k}(x_i - x_j)}.
\end{aligned}$$

Diving by $(x_0 - x_k)$ gives the correct term for $f(x_i)$. So

$$f[x_0, x_1, \ldots, x_k] = \sum_{i=0}^{k} \frac{f(x_i)}{\prod_{j=0,\ j\neq i}^{k}(x_i - x_j)}$$

as desired. □

#### 4.1.1.6 Repeated Arguments in Divided Differences

Formally, $f[x_0, x_1, \ldots, x_n]$ is only defined if $x_0, x_1, \ldots, x_n$ are all *distinct*. However, we can give a meaning to $f[x_0, x_1, \ldots, x_n]$ even if some or all of the $x_i$'s are the same. Here we will look at what happens with just two are the same.

First, $f[x_0, x_0] = (f(x_0) - f(x_0))/(x_0 - x_0) = 0/0$ is undefined so we cannot use that formula. However, the limit does exist if $f$ is differentiable:

$$\begin{aligned}
f[x_0, x_0] &= \lim_{x_1 \to x_0} f[x_0, x_1] \\
&= \lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_0).
\end{aligned}$$

Once we have this, we can work out

$$f[x_0, x_0, x_1] = \frac{f[x_0, x_0] - f[x_0, x_1]}{x_0 - x_1} = \frac{f'(x_0) - f[x_0, x_1]}{x_0 - x_1},$$

and higher order divided differences for $x_0 \neq x_1$.

#### 4.1.1.7 Divided Differences and Polynomial Interpolation

With the result that the divided differences are independent of the order of the points $x_0, x_1, \ldots, x_n$ we can show how they relate to polynomial interpolation. First we note that

$$\begin{aligned} f(x) &= f(x_0) + \frac{f(x) - f(x_0)}{x - x_0}(x - x_0) \\ &= f(x_0) + f[x, x_0]\,(x - x_0). \end{aligned}$$

However, this trick can be repeated for $f[x, x_0]$:

$$\begin{aligned} f[x, x_0] &= f[x_1, x_0] + \frac{f[x, x_0] - f[x_1, x_0]}{x - x_1}(x - x_1) \\ &= f[x_0, x_1] + f[x, x_0, x_1]\,(x - x_1). \end{aligned}$$

And again:

$$\begin{aligned} f[x, x_0, x_1] &= f[x_2, x_0, x_1] + \frac{f[x, x_0, x_1] - f[x_2, x_0, x_1]}{x - x_2}(x - x_2) \\ &= f[x_0, x_1, x_2] + f[x, x_0, x_1, x_2]\,(x - x_2). \end{aligned}$$

In general,

$$\begin{aligned} f[x, x_0, \ldots, x_{k-1}] &= f[x_k, x_0, x_1, \ldots, x_{k-1}] \\ &\quad + \frac{f[x, x_0, \ldots, x_{k-1}] - f[x_k, x_0, \ldots, x_{k-1}]}{x - x_k}(x - x_k) \\ &= f[x_0, x_1, \ldots, x_{k-1}] + f[x, x_0, x_1, \ldots, x_k]\,(x - x_k). \end{aligned}$$

Now we can use this to unwrap our function to expose its interpolating polynomial:

$$\begin{aligned} f(x) &= f(x_0) + (x - x_0)\, f[x, x_0] \\ &= f(x_0) + (x - x_0)\,(f[x_0, x_1] + f[x, x_0, x_1](x - x_1)) \\ &= f(x_0) + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x, x_0, x_1] \\ &= f(x_0) + (x - x_0)\, f[x_0, x_1] + (x - x_0)(x - x_1)\, f[x_0, x_1, x_2] \\ &\qquad + (x - x_0)(x - x_1)(x - x_2)\, f[x, x_0, x_1, x_2] \\ &\qquad \text{etc.} \end{aligned}$$

In general, we get

$$f(x) = \sum_{i=0}^{k} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1}(x - x_j) + \prod_{j=0}^{k}(x - x_j)\, f[x, x_0, x_1, \ldots, x_k].$$

The first part

$$\sum_{i=0}^{k} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1}(x - x_j)$$

is a polynomial in $x$ of degree $\leq k$. Furthermore, the extra part

$$\prod_{j=0}^{k}(x - x_j)\, f[x, x_0, x_1, \ldots, x_k]$$

is zero whenever $x = x_j$, $j = 0, 1, 2, \ldots, k$. So the interpolating polynomial is

$$p_k(x) = \sum_{i=0}^{k} f[x_0, x_1, \ldots, x_i] \prod_{j=0}^{i-1}(x - x_j), \tag{4.1.5}$$

and the error is

$$f(x) - p_k(x) = \left( \prod_{j=0}^{k}(x - x_j) \right) f[x, x_0, x_1, \ldots, x_k]. \tag{4.1.6}$$

Note that to compute $p_k(x)$ we only need $k + 1$ divided differences: $D_i := f[x_0, x_1, \ldots, x_i]$, $i = 0, 1, 2, \ldots, k$.

#### 4.1.1.8 Error Formula

In many cases, we use polynomial interpolation to approximate a definite function $f$. If $p_n$ is the polynomial interpolant of degree $\leq n$ at interpolation points $x_0, x_1, \ldots, x_n$ then for every $x$ there is a $c_x$ where

$$f(x) - p_n(x) = \frac{f^{(n+1)}(c_x)}{(n+1)!}(x - x_0)(x - x_1)\cdots(x - x_n). \tag{4.1.7}$$

Here is a simple way of deriving it.

**Theorem 4.3** *(Interpolation error formula) Suppose $f$ is $(n + 1)$ times differentiable. Suppose that $p_n$ is the polynomial interpolant of $f$ at $(n + 1)$ distinct*

*interpolation points* $x_0, x_1, \ldots, x_n$. *Then there is a* $c_x$ *between* $\min(x, x_0, x_1, \ldots, x_n)$ *and* $\max(x, x_0, x_1, \ldots, x_n)$ *such that (4.1.10) holds.*

***Proof*** Consider the function

$$h(t) = [f(x) - p_n(x)]\,\Psi(t) - [f(t) - p_n(t)]\,\Psi(x),$$

with $\Psi(t) = \prod_{k=0}^{n}(t - x_k)$. This function is $(n + 1)$ times differentiable. We note that since $f(x_i) - p_n(x_i) = 0$ and $\Psi(x_i) = 0$ for $i = 0, 1, 2, \ldots, n$ we have

$$h(x_i) = [f(x) - p_n(x)]\,\Psi(x_i) - [f(x_i) - p_n(x_i)]\,\Psi(x) = 0, \qquad \text{and}$$
$$h(x) = [f(x) - p_n(x)]\,\Psi(x) - [f(x) - p_n(x)]\,\Psi(x) = 0.$$

Thus $h$ has $(n + 2)$ zeros in the interval $[x_{\min}, x_{\max}]$ with $x_{\min} = \min(x, x_0, x_1, \ldots, x_n)$ and $x_{\max} = \max(x, x_0, x_1, \ldots, x_n)$. Between every adjacent pair of these zeros, by Rolle's theorem, there is a zero of $h'$. Thus $h'$ has at least $(n + 1)$ zeros in $[x_{\min}, x_{\max}]$. Again, we can apply Rolle's theorem for each adjacent pair of zeros of $h'$ to show that $h''$ has at least $n$ zeros in $[x_{\min}, x_{\max}]$. Repeating this argument, we see that $h^{(n+1)}$ has at least one zero in $[x_{\min}, x_{\max}]$. Pick one of them. Call it $c_x$.

Now

$$h^{(n+1)}(t) = [f(x) - p_n(x)]\,\Psi^{(n+1)}(t) - \left[f^{(n+1)}(t) - p_n^{(n+1)}(t)\right]\Psi(x).$$

But $p_n^{(n+1)}(t) = 0$ for any $t$, since $p_n$ is a polynomial of degree $\le n$, and so its $(n + 1)$'st derivative is identically zero. Now $\Psi(t) = \prod_{k=0}^{n}(t - x_k) = t^{n+1} - q(t)$ where $q$ is a polynomial of degree $\le n$. Thus $\Psi^{(n+1)}(t) = (d/dt)^{n+1}(t^n) - (d/dt)^{n+1}(q(t)) = (n + 1)! - 0 = (n + 1)!$. Thus

$$0 = h^{(n+1)}(c_x) = [f(x) - p_n(x)]\,(n + 1)! - f^{(n+1)}(c_x)\Psi(x).$$

Re-arranging gives

$$f(x) - p_n(x) = \frac{f^{(n+1)}(c_x)}{(n + 1)!}\Psi(x),$$

as we wanted. □

This error formula will be very valuable in later estimates of the error in methods based on polynomial interpolation.

#### 4.1.1.9 Divided Differences and Integrals

There is an integral representation of divided differences. Let's start with

$$\begin{aligned}
&\int_0^1 f'(x_0 + t_1(x_1 - x_0))\, dt_1 \\
&\quad = \frac{1}{x_1 - x_0}\, f(x_0 + t_1(x_1 - x_0))|_{t_1=0}^{t_1=1} \\
&\quad = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_0, x_1].
\end{aligned}$$

In general, let

$$\tau_n = \{\, (t_1, \ldots, t_n) \mid t_i \geq 0 \text{ for all } i = 1, \ldots, n, \text{ and } \sum_{i=1}^{n} t_i \leq 1 \,\}.$$

This is the $n$-dimensional standard simplex, which is a polyhedron with the vertices $\mathbf{0}$ and $\boldsymbol{e}_i$, $i = 1, 2, \ldots, n$. Then we can show that

**Theorem 4.4** *If $f$ is $n$ times continuously differentiable, then*

$$f[x_0, x_1, \ldots, x_n] = \int_{\tau_n} f^{(n)}(x_0 + \sum_{i=1}^{n} t_i(x_i - x_0))\, dt_1\, dt_2 \cdots dt_n. \tag{4.1.8}$$

This is known as the *Hermite–Genocchi formula*.

***Proof*** The proof is by induction of $n$. Since $\tau_1 = \{\, t_1 \mid 0 \leq t_1 \leq 1 \,\}$, we have already shown that (4.1.8) is true for $n = 1$.

Now let's suppose that (4.1.8) is true for $n = k - 1$. We want to show that (4.1.8) is true for $n = k$. To show that consider

$$\begin{aligned}
&\int_{\tau_k} f^{(k)}(x_0 + \sum_{i=1}^{k} t_i(x_i - x_0))\, dt_1\, dt_2 \ldots dt_k \\
&\quad = \int_{\tau_{k-1}} dt_1 \cdots dt_{k-1} \int_0^{1-\sum_{i=1}^{k-1} t_i} f^{(k)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0) + t_k(x_k - x_0))\, dt_k.
\end{aligned}$$

The inner integral is

$$\begin{aligned}
&\int_0^{1-\sum_{i=1}^{k-1} t_i} f^{(k)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0) + t_k(x_k - x_0))\, dt_k \\
&\quad = \frac{1}{x_k - x_0}\, f^{(k-1)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0) + t_k(x_k - x_0))\Big|_{t_k=0}^{t_k=1-\sum_{i=1}^{k-1} t_i}
\end{aligned}$$

$$= \frac{1}{x_k - x_0}\left[ f^{(k-1)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0) + (x_k - x_0) - \sum_{i=1}^{k-1} t_i(x_k - x_0)) \right.$$
$$\left. - f^{(k-1)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0)) \right]$$
$$= \frac{1}{x_k - x_0}\left[ f^{(k-1)}(x_k + \sum_{i=1}^{k-1} t_i(x_i - x_k)) - f^{(k-1)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0)) \right].$$

So our original integral is

$$\int_{\tau_k} f^{(k)}(x_0 + \sum_{i=1}^{k} t_i(x_i - x_0))\, dt_1\, dt_2 \ldots dt_k$$
$$= \frac{1}{x_k - x_0} \int_{\tau_{k-1}} dt_1 \cdots dt_{k-1} \left[ f^{(k-1)}(x_k + \sum_{i=1}^{k-1} t_i(x_i - x_k)) - f^{(k-1)}(x_0 + \sum_{i=1}^{k-1} t_i(x_i - x_0)) \right]$$
$$= \frac{f[x_k, x_1, \ldots, x_{k-1}] - f[x_0, x_1, \ldots, x_{k-1}]}{x_k - x_0} = f[x_0, x_1, \ldots, x_{k-1}, x_k]$$

as we wanted.

Then by the principle of induction, (4.1.8) holds for $n = 1, 2, 3, \ldots$. □

We can use this integral representation to obtain an estimate for $f[x_0, x_1, \ldots, x_n]$ in terms of $f^{(n)}$ provided $f$ is $n$ times differentiable: by the mean value theorem for multiple integrals,

$$f[x_0, x_1, \ldots, x_{n-1}, x_n] = f^{(n)}(c)\,\mathrm{vol}(\tau_n) = \frac{1}{n!} f^{(n)}(c) \tag{4.1.9}$$

for some $c$ between $\min(x_0, x_1, \ldots, x_n)$ and $\max(x_0, x_1, \ldots, x_n)$.

**Lemma 4.5** *The $n$-dimensional volume of $\tau_n$ is $vol(\tau_n) = 1/n!$.*

***Proof*** The $n$-dimensional volume is

$$\mathrm{vol}(\tau_n) = \int_{\tau_n} dt_1\, dt_2 \ldots dt_n = \int_0^1 dt_n \int_{\tau'_{n-1}} dt_1\, dt_2 \ldots dt_{n-1},$$

where $\tau'_{n-1} = \{ (t_1, \ldots, t_{n-1}) \mid t_i \geq 0\ \forall i,\ \sum_{i=1}^{n-1} t_i \leq 1 - t_n \}$, which is just $\tau_{n-1}$ scaled by a factor of $1 - t_n$. So

$$\mathrm{vol}(\tau_n) = \int_0^1 dt_n\, \mathrm{vol}((1 - t_n)\tau_{n-1}) = \int_0^1 dt_n\, (1 - t_n)^{n-1}\, \mathrm{vol}(\tau_{n-1}) = \frac{1}{n}\, \mathrm{vol}(\tau_{n-1}).$$

To start the recursion, note that

$$\mathrm{vol}(\tau_1) = \int_0^1 dt_1 = 1,$$

so we get $\mathrm{vol}(\tau_n) = 1/n!$. □

Thus we get

$$f[x_0, x_1, \ldots, x_n] = \frac{1}{n!} f^{(n)}(c)$$

for some $c$ between $\min_i x_i$ and $\max_i x_i$ provided $f$ is $n$ times continuously differentiable. This formula can be applied to the *error formula* for polynomial interpolation:

$$\begin{aligned} f(x) - p_n(x) &= f[x, x_0, x_1, \ldots, x_n] \prod_{j=0}^{n} (x - x_j) \\ &= \frac{1}{(n+1)!} f^{(n+1)}(c) \prod_{j=0}^{n} (x - x_j), \end{aligned} \tag{4.1.10}$$

for some $c$ between $\min_i x_i$ and $\max_i x_i$, provided $f^{(n+1)}$ is continuous.

#### 4.1.1.10 Computing Interpolants via Divided Differences

While we have formulas for the divided differences $f[x_0, x_1, \ldots, x_k]$, implementing these formulas directly does not lead to the most efficient way of computing these divided differences, or in evaluating the interpolating polynomial $p(x)$ at a point $x$.

To compute the divided differences $f[x_0, x_1, \ldots, x_k]$ for $k = 0, 1, 2, \ldots, n$, we consider a divided difference table:

$$\begin{array}{llllll} x_0 & f(x_0) & & & & \\ x_1 & f(x_1) & f[x_0, x_1] & & & \\ x_2 & f(x_2) & f[x_1, x_2] & f[x_0 x_1, x_2] & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ x_n & f(x_n) & f[x_{n-1}, x_n] & f[x_{n-2}, x_{n-1}, x_n] & \cdots & f[x_0, x_1, \ldots, x_n]. \end{array}$$

We wish to compute the second-order divided difference $f[x_0, x_1, x_2]=(f[x_0, x_1] - f[x_1, x_2])/(x_0 - x_2)$, so we need both $f[x_0, x_1]$ and $f[x_1, x_2]$. This uses the values in the table to the entry in the table on its left, and the entry left and up.

It is not necessary to keep all entries in this table, even if we compute each of these entries. The entries we need to keep for evaluating the interpolating polynomial are the diagonal entries $D_0 = f(x_0)$, $D_1 = f[x_0, x_1]$, $D_2 = f[x_0, x_1, x_2]$, ... $D_n = f[x_0, x_1, \ldots, x_n]$.

We begin the computations by first computing the first-order divided differences $f[x_{i-1}, x_i]$. We do not wish to overwrite $f(x_0)$, but we can overwrite $f(x_1), f(x_2), \ldots, f(x_n)$ with $f[x_0, x_1]$, $f[x_1, x_2]$, ..., $f[x_{n-1}, x_n]$. We can first overwrite $f(x_n)$ with $f[x_{n-1}, x_n] = (f(x_{n-1}) - f(x_n)/(x_{n-1} - x_n)$, as there is no further use for $f(x_n)$ in

**Algorithm 52** Divided difference algorithm

```
1  function divdif(x, y)
2    d ← y
3    for k = 1, 2, ..., n
4      for i = n, n − 1, ..., k
5        d_i ← (d_i − d_{i−1})/(x_i − x_{i−k})
6      end for
7    end for
8    return d
9  end function
```

**Algorithm 53** Polynomial interpolant evaluation

```
1  function pinterp(x, D, t)
2    pval ← D_n
3    for k = n − 1, ..., 1, 0
4      pval ← D_k + (x − x_k) · pval
5    end for
6    return pval
7  end function
```

this divided difference table. Then $f(x_{n-1})$ can be overwritten by $f[x_{n-2}, x_{n-1}] = (f(x_{n-2}) - f(x_{n-1}))/(x_{n-2} - x_{n-1})$. We can repeat this process going from the bottom to the top of the first-order divided differences. Once the first-order divided differences are computed, the second-order divided differences can be computed, again from the bottom to the top. Continuing in this way we can compute all the divided differences we need. Algorithm 52 shows pseudo-code for computing the divided differences for $y_i = f(x_i)$.

Once the divided differences $D_k = f[x_0, x_1, \ldots, x_k]$ are computed for $k = 0, 1, 2, \ldots, n$, to evaluate the interpolant at $x$, we evaluate

$$p(x) = D_0 + D_1(x - x_0) + D_2(x - x_0)(x - x_1) + \cdots + D_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}).$$

As written, this requires $\sum_{k=0}^{n}(1 + 2k) = (n + 1)^2$ flops. But this can be improved by pulling out common factors for as many terms as possible:

$$p(x) = D_0 + (x - x_0)\left[D_1 + (x - x_1)\left\{D_2 + (x - x_2)\left(D_3 + \cdots\left[D_{n-1} + (x - x_{n-1})D_n\right]\cdots\right)\right\}\right].$$

Pseudo-code implementing this "nested" approach is in Algorithm 53.

In terms of floating point operations, Algorithm 52 requires $\sim \frac{3}{2}n^2$ flops to compute the divided differences, while the interpolant evaluation algorithm, Algorithm 53, requires $3n$ flops for each evaluation of $p(x)$. Algorithm 53 is reminiscent of *Horner's nested multiplication method* for polynomial evaluation using the coefficients, which is shown in Algorithm 54.

**Algorithm 54** Horner's algorithm for $p(t) = \sum_{k=1}^{n} a_k t^k$

```
1  function horner(a, t)
2    pval ← a_n
3    for k = n − 1, ..., 1, 0
4      pval ← a_k + t · pval
5    end for
6    return pval
7  end function
```

#### 4.1.1.11 Barycentric Formulas

Alternative formulations for representing the interpolating polynomial include the *barycentric formulas* [21]. This starts with $\Psi(x) = \prod_{k=0}^{n}(x - x_k)$. Then we can represent the Lagrange interpolation polynomials by

$$L_i(x) = \frac{\Psi(x)}{\Psi'(x_i)(x - x_i)}. \qquad \text{for } x \neq x_i, \tag{4.1.11}$$

$$\Psi'(x_i) = \prod_{j:j\neq i} (x_i - x_j). \tag{4.1.12}$$

Then the interpolating polynomial has the form

$$p(x) = \Psi(x) \sum_{i=0}^{n} y_i \frac{w_i}{x - x_i}, \qquad w_i = \frac{1}{\Psi'(x_i)}. \tag{4.1.13}$$

This is the *first barycentric form* of the interpolating polynomial.

A better form comes from the fact that $1 = \sum_{i=0}^{n} L_i(x)$. This is the Lagrange representation of the constant polynomial $f(x) = 1$ for all $x$. The barycentric representation of one is

$$1 = \Psi(x) \sum_{i=0}^{n} \frac{w_i}{x - x_i}.$$

Substituting this formula for $\Psi(x)$ into the first barycentric form gives the *second barycentric form*:

$$p(x) = \frac{\displaystyle\sum_{i=0}^{n} y_i \frac{w_i}{x - x_i}}{\displaystyle\sum_{i=0}^{n} \frac{w_i}{x - x_i}}. \tag{4.1.14}$$

This can be implemented with just $\mathcal{O}(n)$ flops with pre-computed weights $w_i$; however, care must be taken to avoid overflow and division by zero issues. This representation is used in the `chebfun` system [205].

#### 4.1.1.12 Asymptotics of the Error

Obtaining estimates of the error from the error formula for polynomial interpolation (4.1.7) is, in general, difficult to do precisely. Instead, we aim to obtain asymptotic estimates that are helpful in designing algorithms.

We first consider the situation where we choose an interpolation pattern: $\xi_0 < \xi_1 < \cdots < \xi_n$; the interpolation points are $x_i = a + h\,\xi_i$ for $i = 0, 1, 2, \ldots, n$. We consider interpolation over an interval that is $h$-dependent: $[a + h\,\xi_{\min},\ a + h\xi_{\max}]$. We assume that $\xi_{\min} \le \xi_0 < \cdots < \xi_n \le \xi_{\max}$. For example, if we are considering equally spaced interpolation, we can take $\xi_i = i$ for $i = 0, 1, 2, \ldots, n$, $\xi_{\min} = 0$, and $\xi_{\max} = n$. This scheme allows us to look at a number of other interpolation approaches, while still dealing with the same overall asymptotic behavior.

As with most situations in numerical analysis, it is helpful to identify the things that one has control over, and the things that are given, or beyond our control. Here we control the $\xi_i$'s, $n$, and $h$. But the function $f$ is what it is. For this section, we focus our interest in the behavior of the error as $h$ becomes small.

For the interpolation points $x_i = a + h\,\xi_i$, our formula for the interpolation error is

$$f(x) - p_n(x) = \frac{f^{(n+1)}(c_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j).$$

We focus now on the maximum error over the interpolation interval $[a + h\,\xi_{\min},\ a + h\xi_{\max}]$:

$$\begin{aligned}
&\max_{x \in [a+h\,\xi_{\min},\, a+h\xi_{\max}]} |f(x) - p_n(x)| \\
&\quad = \max_{x \in [a+h\,\xi_{\min},\, a+h\xi_{\max}]} \left| \frac{f^{(n+1)}(c_x)}{(n+1)!} \prod_{j=0}^{n} (x - x_j) \right| \\
&\quad \le \max_{x \in [a+h\,\xi_{\min},\, a+h\xi_{\max}]} \left| \frac{f^{(n+1)}(c_x)}{(n+1)!} \right| \max_{x \in [a+h\,\xi_{\min},\, a+h\xi_{\max}]} \left| \prod_{j=0}^{n} (x - x_j) \right|.
\end{aligned}$$

Note that $c_x$ is between $a + h\xi_{\min}$ and $a + h\xi_{\max}$. Now the maximum of $\left|f^{(n+1)}(c_x)\right|$ over $x \in [a + h\,\xi_{\min},\ a + h\xi_{\max}]$ is not something we have much control over. We can see that provided $f^{(n+1)}$ is continuous, this maximum will approach $\left|f^{(n+1)}(a)\right|$ as $h \to 0$. Since we have a finite limit, then for all $h > 0$ "sufficiently small" there is a bound $\left|f^{(n+1)}(c)\right| \le M_n$ for all $c \in [a + h\,\xi_{\min},\ a + h\xi_{\max}]$. This leads to

$$\max_{x\in[a+h\,\xi_{\min},\,a+h\xi_{\max}]} |f(x)-p_n(x)|$$
$$\le \frac{M_n}{(n+1)!}\max_{x\in[a+h\,\xi_{\min},\,a+h\xi_{\max}]}\left|\prod_{j=0}^{n}(x-x_j)\right|.$$

Writing $x\in[a+h\,\xi_{\min},\ a+h\xi_{\max}]$ as $x=a+h\xi$ and $x_j=a+h\xi_j$ we get

$$\max_{x\in[a+h\,\xi_{\min},\,a+h\xi_{\max}]} |f(x)-p_n(x)|$$
$$\le \frac{M_n}{(n+1)!}\max_{\xi\in[\xi_{\min},\,\xi_{\max}]}\left|\prod_{j=0}^{n}((a+h\xi)-(a+h\xi_j))\right|$$
$$= \frac{M_n}{(n+1)!}\max_{\xi\in[\xi_{\min},\,\xi_{\max}]}\left|\prod_{j=0}^{n}(h(\xi-\xi_j))\right|$$
$$= h^{n+1}\,\frac{M_n}{(n+1)!}\max_{\xi\in[\xi_{\min},\,\xi_{\max}]}\left|\prod_{j=0}^{n}(\xi-\xi_j)\right|.$$

This leads to first asymptotic results:

$$\max_{x\in[a+h\,\xi_{\min},\,a+h\xi_{\max}]} |f(x)-p_n(x)| \le h^{n+1}\,\frac{M_n}{(n+1)!}\max_{\xi\in[\xi_{\min},\,\xi_{\max}]}\left|\prod_{j=0}^{n}(\xi-\xi_j)\right|$$
$$= h^{n+1}\,\frac{M_n}{(n+1)!}\,K_n(\boldsymbol{\xi}) = \mathcal{O}(h^{n+1}) \qquad \text{as } h\to 0. \tag{4.1.15}$$

Note that while

the maximum error over interpolation interval is $\mathcal{O}(h^{n+1})$ as $h\to 0$

is a simple and straightforward lesson, we should remember that there is a hidden constant that depends on $n$ and the choice of interpolation scheme $\boldsymbol{\xi}$. Nevertheless, it is a useful lesson.

**Example 4.6** As an example, we show the maximum error for polynomial interpolation error for equally spaced interpolation ($\xi_i = i$) and different values of $n$ and $h$ applied to the function $f(x) = e^x/\sqrt{1+x}$ near $a=0$ in Figure 4.1.3. Note the increasing slope as the order increases; the curves do not stay even approximately straight, as the maximum error cannot go much lower than unit roundoff. Estimates of the slopes, being the exponent $p$ in the approximate relationship max error $\approx$ constant $\cdot\, h^p$, derived empirically from the data of Figure 4.1.3, are shown in Table 4.1.2. While most of the slopes are close to $n+1$, for higher degree interpolants, this precise relation is somewhat obscured by the floor of roundoff error.

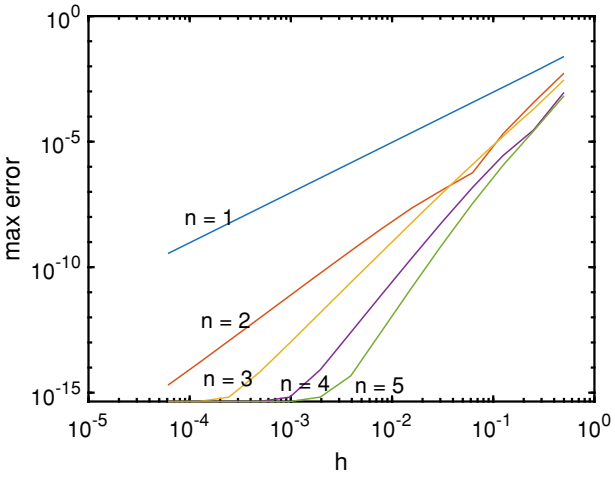



**Fig. 4.1.3** Maximum error over interpolation interval for equally spaced interpolation points; $f(x) = e^x/\sqrt{1+x}$ near $x = 0$

**Table 4.1.2** Slope estimates for log–log plot of maximum error against spacing $h$

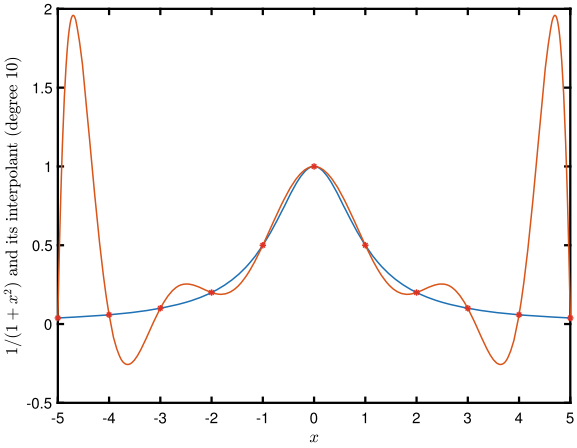
| $n$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Slope estimate | 1.999 | 3.098 | 3.870 | 4.682 | 5.573 |
| Asymptotic slope | 2 | 3 | 4 | 5 | 6 |

#### 4.1.1.13 Runge Phenomenon

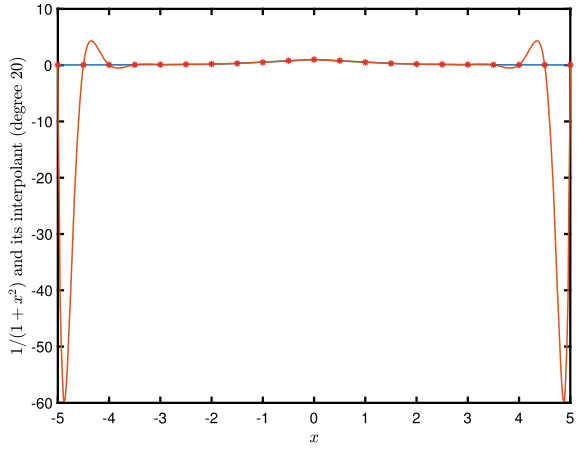
Figure 4.1.3 shows how taking $h \to 0$ affects the maximum error, *for fixed* $n$. But if we vary both $h$ and $n$ together, this simple asymptotic relationship does not necessarily hold. What happens then depends very much on the function being interpolated, the interval over which it is being interpolated, and also the interpolation scheme $(\xi_0, \xi_1, \ldots, \xi_n)$ being used. In fact, the error can easily grow rapidly as $n$ is increased, as Carl Runge discovered in 1901 [223].

The Runge phenomenon is the failure of convergence of polynomial interpolants for certain functions over a fixed interval as the degree of the interpolant $n$ goes to infinity. Runge's specific example was equally spaced interpolation of $f(x) = 1/(1+x^2)$ over the interval $[-5, +5]$. Plots for $n = 10$ and $n = 20$ are shown in Figure 4.1.4. Figure 4.1.5 shows how the maximum error of the interpolant behaves as the degree $n$ increases for this case and for interpolating the same function over $[-2, +2]$, and over $[-1, +1]$.

The moral of this story: reduce $h$ before increasing $n$.




(A) $n = 10$




(B) $n = 20$

**Fig. 4.1.4** Plots of equally spaced interpolants for $f(x) = 1/(1 + x^2)$ on $[-5, +5]$

#### 4.1.1.14 Hermite Interpolation

A variant of traditional interpolation is to use derivative information to specify the interpolant. The simplest of these variants is *Hermite interpolation*: given $(x_i, y_i, y_i')$ for $i = 0, 1, 2, \ldots, n$, we want to find a polynomial $p(x)$ of minimal degree where

$$p(x_i) = y_i, \qquad p'(x_i) = y_i', \qquad \text{for } i = 0, 1, 2, \ldots, n. \tag{4.1.16}$$

Note that this is a system of $2(n + 1)$ equations, so we seek the degree so that $p$ has $2(n + 1)$ coefficients; that is, we seek $p$ with degree no more than $2n + 1$. If $\boldsymbol{a} \in \mathbb{R}^{2n}$

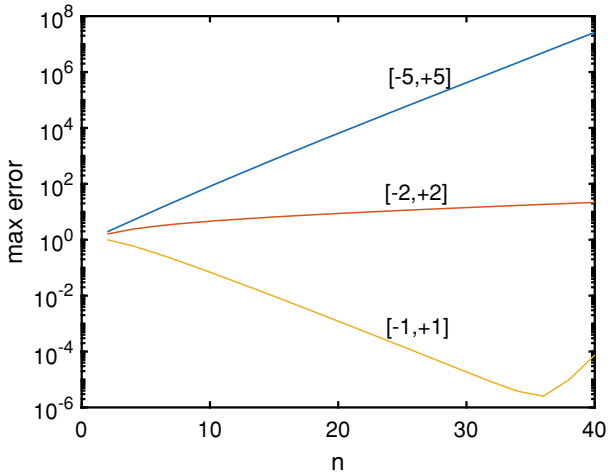



**Fig. 4.1.5** Runge phenomenon: interpolating $1/(1+x^2)$ with equally spaced interpolation points over $[-5, +5]$, $[-2, +2]$ and $[-1, +1]$

is the vector of coefficients for $p$, then the linear map $\boldsymbol{F}: \boldsymbol{a} \mapsto [p(x_i; \boldsymbol{a}),\ p'(x_i; \boldsymbol{a}) \mid i = 0, 1, \ldots, n]$ from $\mathbb{R}^{2n}$ to $\mathbb{R}^{2n}$ is one to one: if $\boldsymbol{F}(\boldsymbol{a}^*) = \boldsymbol{0}$ then $p(x; \boldsymbol{a}^*)$ is the polynomial of degree $\leq 2n+1$ has zeros of multiplicity two at each point $x_i$. Thus $p(x; \boldsymbol{a}^*) = \prod_{i=0}^{n}(x - x_i)^2 \cdot g(x)$. But then the degree of $p$ is $2(n+1) + \deg g$. This implies that $\deg g < 0$, which can only occur if $g(x)$ is identically zero. That means that $\boldsymbol{a}^* = \boldsymbol{0}$. Since $\boldsymbol{F}: \mathbb{R}^{2(n+1)} \to \mathbb{R}^{2(n+1)}$ is linear and one to one, $\boldsymbol{F}$ is also onto. That means there is exactly one Hermite interpolant for the given data as long as the $x_i$'s are distinct.

Computing the Hermite interpolant of this data can be done using divided difference tables using the identity $f[x_i, x_i] = f'(x_i)$:

$$
\begin{array}{llllll}
x_0 & f(x_0) & & & & \\
x_0 & f(x_0) & f[x_0, x_0] = f'(x_0) & & & \\
x_1 & f(x_1) & f[x_0, x_1] & f[x_0, x_0, x_1] & & \\
x_1 & f(x_1) & f[x_1, x_1] = f'(x_1) & f[x_0, x_1, x_1] & \ddots & \\
x_2 & f(x_2) & f[x_1, x_2] & f[x_1, x_1, x_2] & \cdots & \\
\vdots & \vdots & \vdots & \vdots & \cdots & \\
x_n & f(x_n) & f[x_{n-1}, x_n] & f[x_{n-1}, x_{n-1}, x_n] & \cdots & \\
x_n & f(x_n) & f[x_n, x_n] = f'(x_n) & f[x_{n-1}, x_n, x_n] & \cdots &
\end{array}
$$

Initially, the column of function values is filled in with $f(x_i)$ appearing twice, and then the second column has every second entry filled with the derivative $f'(x_i) = f[x_i, x_i]$. The remaining entries in the column for first-order divided differences are filled in using the standard formula. Once that is done, the remainder of the table can be filled in using the standard formulas. The evaluation of the Hermite interpolant can be done by a modification of the algorithm for standard interpolation:

**Algorithm 55** Hermite divided difference algorithm

```
function hdivdif(x, y, y')
  d_0 ← y_0 ; d_1 ← y'_0
  x̃_0 ← x_0 ; x̃_1 ← x_0
  for i = 1, 2, ..., n
    d_2i ← (y_i − y_{i−1})/(x_i − x_{i−1})
    d_{2i+1} ← y'_i
    x̃_2i ← x_i ; x̃_{2i+1} ← x_i
  end for
  for k = 2, ..., 2n
    for i = 2n, 2n − 1, ..., k
      d_i ← (d_i − d_{i−1})/(x̃_i − x̃_{i−k})
    end for
  end for
  return d
end function
```

**Algorithm 56** Evaluation of Hermite interpolant

```
function hinterp(x, D, t)
  for i = 0, 1, 2, ..., n
    x̃_2i ← x_i ; x̃_{2i+1} ← x_i
  end for
  return pinterp(x̃, D, t)
end function
```

$$p(x) = f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_1](x - x_0)^2 + \cdots + f[x_0, x_0, \ldots, x_n, x_n](x - x_0)^2 \cdots (x - x_{n-1})^2 (x - x_n).$$

Pseudo-code for these are shown in Algorithms 55 and 56.

There is also a Lagrange-type representation:

$$p(x) = \sum_{i=0}^{n} \left( y_i \, H_i(x) + y'_i \, K_i(x) \right). \tag{4.1.17}$$

The polynomials $H_i$ and $K_i$ can be written in terms of the standard Lagrange interpolation polynomials:

$$H_i(x) = L_i(x)^2 \left[ 1 - 2L'_i(x_i)(x - x_i) \right],$$
$$K_i(x) = L_i(x)^2 (x - x_i).$$

Note that $L'_i(x_i) = \sum_{j:j \neq i} (x_i - x_j)^{-1}$.

The interpolation error formula has a similar form to the standard one:

$$f(x) - p(x) = \frac{f^{(2n+2)}(c_x)}{(2n+2)!} \prod_{i=0}^{n} (x - x_i)^2, \tag{4.1.18}$$

for some $c_x$ between $\min(x, x_0, \ldots, x_n)$ and $\max(x, x_0, \ldots, x_n)$. While Hermite interpolation promises higher order accuracy, with an error of $\mathcal{O}(h^{2n+2})$ for fixed $n$ as $h \to 0$, it does not perform significantly better than standard equally spaced interpolation on Runge's example.

This idea can be generalized to picking an arbitrary number of derivatives to specify at each interpolation point. The extreme case is where one point is repeated as many times as appropriate in the divided difference formula:

$$p(x) = f(x_0) + f[x_0, x_0](x - x_0) + f[x_0, x_0, x_0](x - x_0)^2 + \cdots + f[\underbrace{x_0, \ldots, x_0}_{n+1 \text{ times}}](x - x_0)^n + f[x, \underbrace{x_0, \ldots, x_0}_{n+1 \text{ times}}](x - x_0)^{n+1}.$$

Application of the Hermite–Genocchi formula (4.1.8) reveals that this formula is simply Taylor series with remainder in integral form.

### 4.1.2 *Lebesgue Numbers and Reliability*

We would like the result of interpolation to be close to the best possible approximation from our family of interpolating functions. But the Runge phenomenon (Section 4.1.1.13) shows that sometimes this is far from being true. In fact, interpolation can be extremely bad. So how can we trust an interpolation scheme? Some functions are difficult to approximate by polynomials. Rough functions are harder to approximate by polynomials. Functions with isolated features, like $f(x) = 1/(1 + x^2)$ over the interval $[-5, +5]$, are also hard to approximate by polynomials. But the Runge phenomenon shows that the maximum interpolation error can *increase* as the degree $n$ increases, while the best approximation by polynomials of degree $\leq n$ can only *decrease* as $n$ increases. Why does this happen?

We want to separate the question of how hard a function is to approximate by polynomials from the questions of the reliability of the interpolation method.

Suppose that we have an interpolation scheme: given function values $f(x_i)$, $i = 0, 1, 2, \ldots, n$, we have an interpolant

$$Pf(x) = \sum_{i=0}^{n} f(x_i)\, \ell_i(x). \tag{4.1.19}$$

In the case of polynomial interpolation, the $\ell_i$ functions are the Lagrange interpolation polynomials $L_i$ in (4.1.3). Other kinds of interpolation can be incorporated into this approach: trigonometric polynomial interpolation, spline interpolation, and multivariate interpolation of different kinds.

We can now consider $P$ as a linear function from continuous functions on $[a, b]$, denoted $C[a, b]$, using

$$\|g\|_\infty = \max_{a\le x\le b} |g(x)|$$

as the norm for continuous functions. Let $\mathcal{P}$ be the set of interpolating functions: $\mathcal{P}$ is the image of $P$. An essential property for the interpolation operator $P$ is that for any $q \in \mathcal{P}$, $Pq = q$, that is, the interpolant of an interpolant is the original interpolant. More succinctly, $P^2 f = Pf$ for any $f \in C[a,b]$. So $P^2 = P$. That is, $P$ is a *projection*.

What is important here is the operator norm

$$\|P\|_\infty = \sup_{f\neq 0} \frac{\|Pf\|_\infty}{\|f\|_\infty} = \sup_{f:\|f\|_\infty=1} \|Pf\|_\infty \,. \tag{4.1.20}$$

We can compute this operator norm:

$$\begin{aligned}
\|Pf\|_\infty &= \max_{a\le x\le b} |Pf(x)| = \max_{a\le x\le b} \left| \sum_{i=0}^n f(x_i)\,\ell_i(x) \right| \\
&\le \max_{a\le x\le b} \sum_{i=0}^n |f(x_i)|\,|\ell_i(x)| \le \max_{a\le x\le b} \sum_{i=0}^n \|f\|_\infty\,|\ell_i(x)| \\
&= \|f\|_\infty \max_{a\le x\le b} \sum_{i=0}^n |\ell_i(x)|\,, \qquad\qquad \text{so} \\
\|P\|_\infty &\le \max_{a\le x\le b} \sum_{i=0}^n |\ell_i(x)|\,.
\end{aligned}$$

In fact, this upper bound *is* the value of $\|P\|_\infty$: Suppose that $\sum_{i=0}^n |\ell_i(x^*)| = \max_{a\le x\le b} \sum_{i=0}^n |\ell_i(x)|$. Then set $f$ to be a function satisfying $f(x_i) = \operatorname{sign} \ell_i(x^*)$, and $|f(x)| \le 1$ for all $x$. We can do this by using piecewise linear interpolation, for example. Then for this $f$, $\|f\|_\infty = 1$ and

$$\begin{aligned}
\|Pf\|_\infty &= \max_{a\le x\le b} \left| \sum_{i=0}^n f(x_i)\,\ell_i(x) \right| \ge \left| \sum_{i=0}^n f(x_i)\,\ell_i(x^*) \right| \\
&= \left| \sum_{i=0}^n \operatorname{sign} \ell_i(x^*)\cdot \ell_i(x^*) \right| = \sum_{i=0}^n \left|\ell_i(x^*)\right| \ge \|P\|_\infty\,\|f\|_\infty \,.
\end{aligned}$$

Since the inequalities between $\|P\|_\infty$ and $\max_{a\le x\le b} \sum_{i=0}^n |\ell_i(x)|$ go in both directions, they must be equal. We call $\|P\|_\infty = \max_{a\le x\le b} \sum_{i=0}^n |\ell_i(x)|$ the *Lebesgue number* for the interpolation method.

**Theorem 4.7** *(Lebesgue numbers) For any $f \in C[a,b]$ and interpolation operator $P$ on $C[a,b]$, the interpolation error*

$$\|f - Pf\|_\infty \le (1 + \|P\|_\infty) \inf_{q \in \mathcal{P}} \|f - q\|_\infty .$$

***Proof*** For any $q \in \mathcal{P}$,

$$\begin{aligned} \|f - Pf\|_\infty &= \|f - q + q - Pf\|_\infty \qquad \text{but } Pq = q \text{ since } q \in \mathcal{P}, \\ &= \|f - q + Pq - Pf\|_\infty \\ &\le \|f - q\|_\infty + \|P(q - f)\|_\infty \\ &\le \|f - q\|_\infty + \|P\|_\infty \|f - q\|_\infty \\ &= (1 + \|P\|_\infty) \|f - q\|_\infty , \end{aligned}$$

as we wanted. □

The important issue then is to estimate

$$\|P\|_\infty = \max_{a \le x \le b} \sum_{i=0}^{n} |\ell_i(x)|$$

for any interpolation scheme we wish to investigate. Since for any interpolation scheme that is exact for constant functions, $1 = \sum_{i=0}^{n} \ell_i(x)$ so $1 \le \sum_{i=0}^{n} |\ell_i(x)|$ and thus $\max_{a \le x \le b} \sum_{i=0}^{n} |\ell_i(x)| \ge 1$. This is not surprising as $P$ is a projection: $P = P^2$. For *any* operator norm $\|P\| = \left\|P^2\right\| \le \|P\|^2$ so either $\|P\| = 0$ (and $P = 0$) or $\|P\| \ge 1$.

We first consider equally spaced polynomial interpolation. We focus on the case of interpolation on [0, 1]. Later we will see that the Lebesgue number does not depend on the interval $[a, b]$. For now, we note that for standard polynomial interpolation, $\ell_i(x) = L_i(x)$, the Lagrange interpolation polynomials (4.1.3).

Figure 4.1.6 shows $\sum_{i=0}^{n} |L_i(x)|$ for equally spaced interpolation ($x_i = i/n$) for $n = 20$. Clearly the maximizing $x$ is between $x = 0$ and $x = 1/n$, and symmetrically, between $x = 1 - 1/n$ and $x = 1$. Most of this is due to $L_i(x)$ for $i \approx n/2$, not $i \approx 0$, or $i \approx n$. Figure 4.1.7 shows $|L_i(x)|$ for equally spaced interpolation with $n = 20$ and $i = 10$.

The Lebesgue number for equally spaced interpolation depends on the degree of the interpolation. How this changes with $n$ is shown in Figure 4.1.8.

Refined calculations give the asymptotic value of Lebesgue constants for equally spaced interpolation (denoted $\Lambda_n$) [231]:

$$\Lambda_n \sim \frac{2^{n+1}}{e\, n\, \ln n} \qquad \text{as } n \to \infty.$$

This exponential growth results in large interpolation errors. Even if the best approximation error goes to zero at an exponential rate as the polynomial degree $n$ goes to infinity, the exponential growth of the Lebesgue constants will result in exploding interpolation errors unless the exponential decay of the best approximation is even faster. The example of the Runge phenomenon (Section 4.1.1.13) shows that

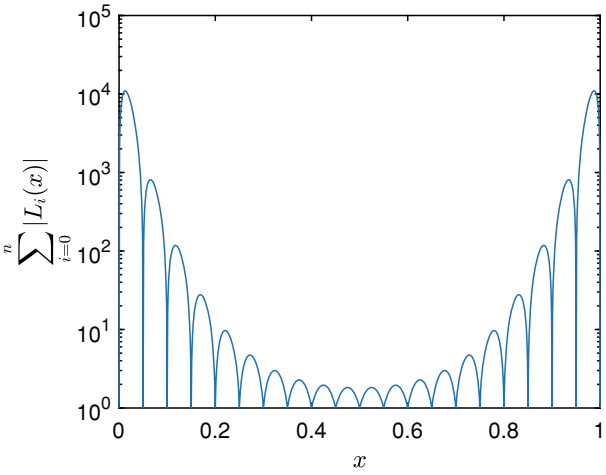



**Fig. 4.1.6** $\sum_{i=0}^{n} |L_i(x)|$ for equally spaced interpolation points, $n = 20$

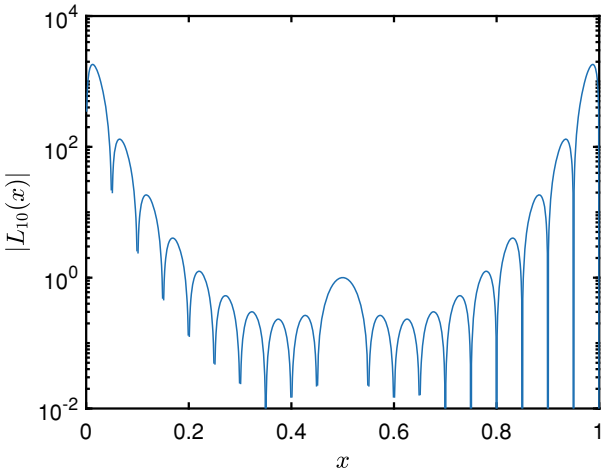



**Fig. 4.1.7** $|L_{10}(x)|$for equally spaced interpolation, $n = 20$

interpolation errors can grow exponentially even for functions that are infinitely differentiable.

Using a different interpolation scheme, with a different distribution of interpolation points, we can get drastically different Lebesgue constants. For Chebyshev points (Section 4.6.2),

$$x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos(\frac{(k+\frac{1}{2})\pi}{n+1}), \qquad k = 0, 1, 2, \ldots, n,$$

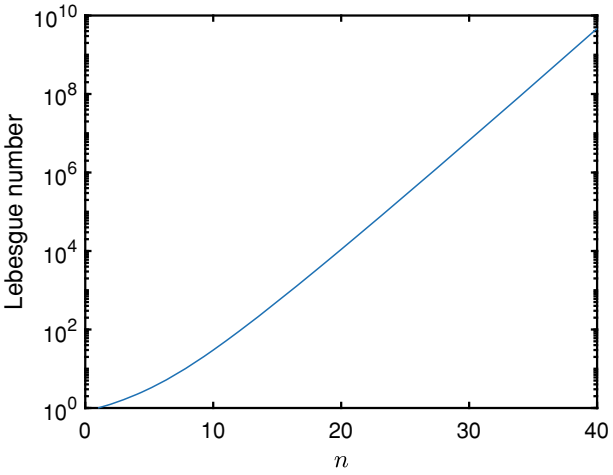



**Fig. 4.1.8** Lebesgue numbers for equally spaced polynomial interpolation

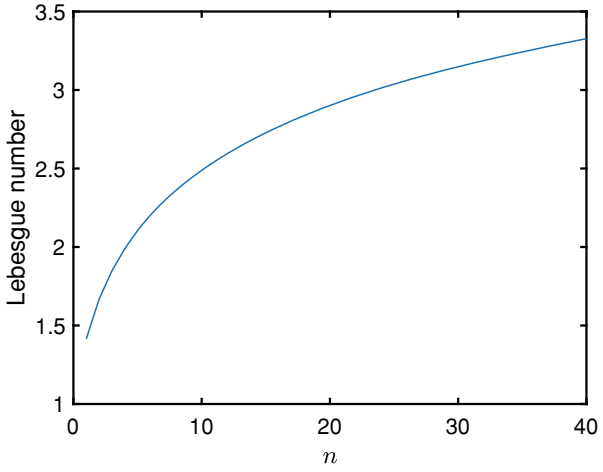



**Fig. 4.1.9** Lebesgue numbers for Chebyshev polynomial interpolation

we have the Lebesgue constants [214]:

$$\Lambda_n \sim \frac{2}{\pi} \ln n \qquad \text{as } n \to \infty.$$

Figure 4.1.9 shows the Lebesgue constants for polynomial interpolation with Chebyshev points.

The slow growth of the Lebesgue numbers for polynomial interpolation using Chebyshev points indicates the quality of these points for interpolation. Because of

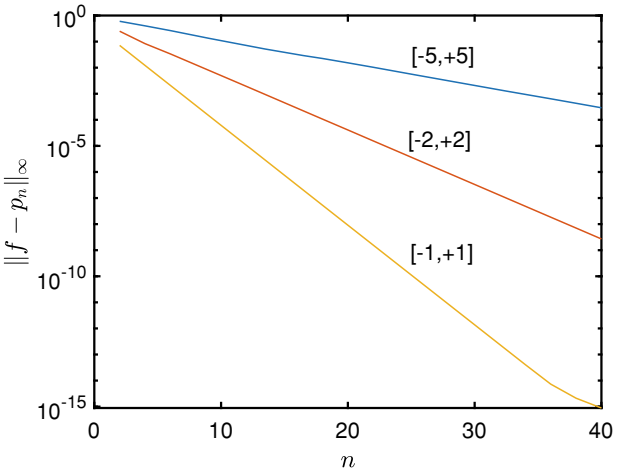



**Fig. 4.1.10** Maximum interpolation error for $f(x) = 1/(1 + x^2)$ on $[-5, +5]$, $[-2, +2]$, and $[-1, +1]$ using $n + 1$ Chebyshev points

this slow growth, the Runge phenomenon does not occur with Chebyshev points. Compare the maximum errors using Chebyshev points as shown in Figure 4.1.10 with those shown in Figure 4.1.5.

Instead of having the interpolation error increasing, it is decreasing at an exponential rate. As noted in Section 4.1.1.13, it is still better to reduce spacing before increasing $n$.

Lebesgue constants are also very useful in dealing with multivariate interpolation, which will be discussed in the following sections.

**Exercises.**

(1) Let $f(x) = e^{-x}/(1 + x^2)$. Compute the quadratic interpolant $p(x)$ (or some representation of it) for this function using interpolation points $x_0 = 0$, $x_1 = 1/2$, and $x_2 = 1$. Plot the difference $f(x) - p(x)$ for $0 \le x \le 1$. For the plot, compute the difference $f(x_k) - p(x_k)$ for $k = 0, 1, 2, \ldots, N$ for $x_k = k/N$ with $N = 100$.

(2) Show that if $x_0$, $x_1$, and $x_2$ are distinct, then $p(x) := x_0 L_0(x) + x_1 L_1(x) + x_2 L_2(x) = x$ for all $x$. Here $L_j(x)$ are the quadratic Lagrange interpolation polynomials for these interpolation points. [**Hint:** Let $f(x) = x$; then $p(x)$ is the quadratic interpolant of $f(x)$ at $x_0$, $x_1$, $x_2$. But $f$ is also a quadratic interpolant of this data; by uniqueness of the quadratic interpolant, $f(x) = p(x)$ for all $x$.]

(3) Let $f(x) = e^{-x}/(1 + x^2)$. For $n = 1, 2, 3$ and for $h = 2^{-m}, m = 1, 2, \ldots, 8$, compute the interpolating polynomial $p$ of $f(x_k)$ of degree $\le n$ where $x_k = a + k\,h$, $k = 0, 1, \ldots, n$, and $a = \frac{1}{2}$. Estimate the maximum interpolation error for each pair $(n, h)$: $\max_{a \le x \le a+nh} |f(x) - p(x)|$. Estimate this

maximum error by evaluating $f(x) - p(x)$ at 101 points equally spaced in the interval $[a,\ a + nh]$. Plot the maximum error against $h$ for each value of $n$. You should use logarithmic scaling for both $h$ and the maximum error. Estimate the slope of the plots. If $\epsilon_{n,m}$ is the maximum error for degree $n$ interpolation and $h = 2^{-m}$, then we compute $slope = (\log(\epsilon_{n,\ell}) - \log(\epsilon_{n,m}))/(\log(h_\ell) - \log(h_m))$ for suitable values of $\ell$ and $m$. These slopes should approximate the order of the interpolation error, which should be $n + 1$. Report the estimated slopes and comment on how close the slopes are to $n + 1$.

(4) Suppose that $x_k = a + \xi_k\, h$ and $y_\ell = b + \eta_j\, h$ for $k, \ell = 1, 2, \ldots, n$. Show how we can create an interpolating polynomial $p(x, y)$ of the data $p(x_k, y_\ell) = z_{k,\ell}$ where the degree of $x \mapsto p(x, y)$ and the degree of $y \mapsto p(x, y)$ are both $\leq n$. Re-use the functions for divided differences in one variable to implement your method. Compute the divided differences $D_{i,j}$ of $z_{i,j}$ with respect to $y$ for each $i$, and then the divided differences of $E_{i,j}$ of $D_{i,j}$ with respect to $x$ for each $j$. To evaluate $p(x, y)$ we evaluate first

$$D_j(x) = \sum_{i=0}^{m} E_{i,j} \prod_{k=0}^{i-1} (x - x_k), \qquad j = 0, 1, \ldots, n, \qquad \text{and then}$$

$$p(x, y) = \sum_{j=0}^{n} D_j(x) \prod_{\ell=0}^{j-1} (y - y_\ell).$$

How many arithmetic operations are needed to compute the divided differences, and how many to evaluate $p(x, y)$ for each $(x, y)$? Test your method for the function $f(x, y) = \exp(-x^2 - y^2) + \cos(\pi x)/(1 + x + y)$ for $a = b = 0$ with equally spaced interpolation points ($\xi_k = k$, $\eta_\ell = \ell$) for $n = 2$ and $h = 2^{-m}$ for $m = 1, 2, \ldots, 5$. What is the order of accuracy of this method?

(5) An alternative to the previous exercise is to use two-variable Lagrange interpolation polynomials:

$$L_{i,j}(x, y) = L_i(x)\, \widetilde{L}_j(y),$$

where $L_i(x)$ is the Lagrange interpolation polynomial in $x$, and $\widetilde{L}_j(y)$ is the Lagrange interpolation polynomial in $y$. Use this to create an algorithm for computing

$$p(x, y) = \sum_{i=0}^{m} \sum_{j=0}^{n} z_{i,j}\, L_{i,j}(x, y).$$

How many arithmetic operations are needed to evaluate $p(x, y)$ for each $(x, y)$? Test your code as described in the previous exercise.

(6) Extend Exercise 4 to functions of three variables. What are the problems of using this interpolation method for functions of $d$ variables if $d$ is large?

(7) Repeat Exercise 3 using Hermite interpolation. This does mean you will have to compute $f'(x)$ symbolically.

(8) The Runge phenomenon is explained in more detail in [86], which uses complex analysis to better show the exponential growth of the error. If the interpolation points are $x_0,\ x_1,\ \ldots,\ x_n \in [a, b]$ and $\Psi_n(t) = \prod_{k=0}^{n}(t - x_k)$, show using the Residue Theorem of complex analysis that for $t \in [a, b]$

$$f(t) - p_n(t) = \frac{1}{2\pi i}\int_C \frac{\Psi_n(t)}{\Psi_n(z)}\frac{f(z)}{z-t}dz,$$

where $C$ is a closed curve that goes once counter-clockwise around the interval $[a, b]$. Also show that if the interpolation points are equally spaced, then $(1/n)\ln|\Psi_n(z)| \to (1/(b-a))\int_a^b \ln|z-t|\,dt$ as $n \to \infty$ for $z \notin [a, b]$.

(9) The Chebyshev interpolation points (4.6.5) on the interval $(-1,\ +1)$ are given by $x_k = \cos((k + \frac{1}{2})\pi/(n+1))$ for $k = 0, 1, 2, \ldots, n$ are the roots of the Chebyshev polynomial (4.6.3) $T_{n+1}(\cos\theta) = \cos((n+1)\theta)$. Derive the weights $w_k$ for the barycentric interpolation Formula (4.1.14) for the Chebyshev points. [**Hint:** Use the fact that $\Psi_n(t) = \prod_{k=0}^{n}(t - x_k) = 2^{-n}\,T_{n+1}(t)$ and compute $\Psi'(x_j) = 2^{-n}\,T'_{n+1}(x_j)$ via the definition $T_{n+1}(\cos\theta) = \cos((n+1)\theta)$.]

(10) Derive the weights $w_k$ for the barycentric interpolation formula for equally spaced interpolation points $x_k = k/n$ for $k = 0, 1, 2, \ldots, n$. [**Hint:** Write $\Psi'(x_j)$ in terms of factorials.]

(11) Suppose that we have interpolation points $a \le x_0 < x_1 < \cdots < x_n \le b$ and $c \le y_0 < y_1 < \cdots < y_n \le d$; then we can interpolate over the interval $[a, b] \times [c, d]$ using $p(x_k, y_\ell) = f(x_k, y_\ell)$ for $k, \ell = 0, 1, 2 \ldots, n$ for given function $f$. Suppose $\Lambda_n$ is the Lebesgue number for the one-dimensional interpolation using $x_0, x_1, \ldots, x_n$, and $M_n$ the Lebesgue constant for the one-dimensional interpolation using $y_0, y_1, \ldots, y_n$. Show that $\Lambda_n M_n$ is the Lebesgue constant for the two-dimensional interpolation described above.

(12) Use the standard algorithm for computing divided differences in double precision on equally spaced points $a = x_0 < x_1 < x_2 < \cdots < x_n = b$ for $f(x) = \exp(-\sqrt{1+x^2})$ on the interval $[a, b] = [-4,\ +4]$ with $n = 80$. Close examination of the plot of the interpolant reveals erratic behavior near $x = +4$ which indicates roundoff error (see Figure 4.1.11). What do you get if you reverse the order of the interpolation points? What if the interpolation points are randomly ordered? Can you give a deterministic ordering of the interpolation points that avoids problems with roundoff error?

## 4.2 Interpolation—Splines

The Runge phenomenon shows how polynomial interpolation can fail. While the distribution of interpolation points over the interpolation interval can reduce these problems, it is not always possible to get data for these points. More reliable inter-

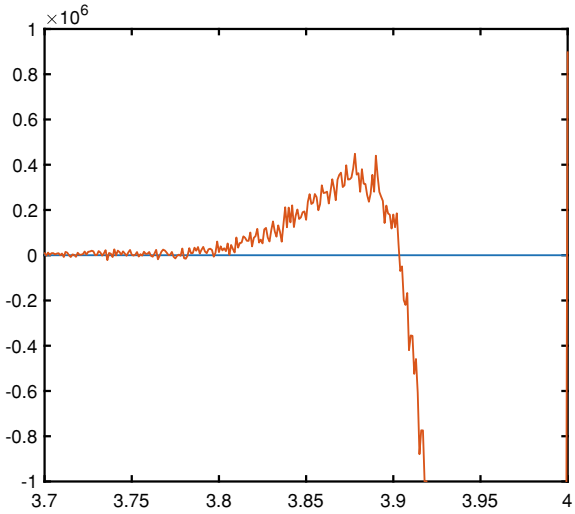



**Fig. 4.1.11** Roundoff error in interpolation

polation methods are desirable. However, there are trade-offs: the rapid decay of the maximum error *if the interpolation interval is small enough* no longer holds.

*Spline interpolation*, at least in one dimension, is *piecewise polynomial interpolation*. That is, given the interpolation points $x_0 < x_1 < x_2 < \ldots < x_n$ and data points $(x_i, y_i)$, $i = 0, 1, 2, \ldots, n$, on each piece $[x_i, x_{i+1}]$ the interpolant $p(x)$ is a polynomial. This can be done using standard polynomial interpolation by using a subset $\{x_{i-k}, x_{i-k+1}, \ldots, x_{i+\ell}\}$ of neighboring interpolation points to give an interpolating polynomial of degree $\leq k + \ell$ that is used on $[x_i, x_{i+1}]$. The piecewise polynomial created in this way is continuous: the polynomial used to interpolate on $[x_{i-1}, x_i]$ and the polynomial used for $[x_i, x_{i+1}]$ both interpolate the same value $y_i$ at $x_i$. However, the derivatives at $x_i$ usually do not match.

While piecewise polynomials with discontinuous derivatives can serve many purposes, the needs of some industries such as aerospace, vehicle manufacturing, and Computer-Aided Design (CAD), in general, motivated a search for more general interpolation method that gave better smoothness without the reliability issues of standard polynomial interpolation.

### *4.2.1 Cubic Splines*

The simplest kind of splines are *linear splines*. A linear spline is simply piecewise linear interpolation: given the points $(x_i, y_i)$ and $(x_{i+1}, y_{i+1})$ with $x_i < x_{i+1}$ we set $\ell(x) = y_i\, L_{i,0}(x) + y_{i+1}\, L_{i,1}(x)$ for $x_i \leq x \leq x_{i+1}$ where $L_{i,0}(x) = (x - x_{i+1})/(x_i - x_{i+1})$ and $L_{i,1}(x) = (x - x_i)/(x_{i+1} - x_i)$ are the corresponding Lagrange interpolating polynomials. There is another way of looking at linear splines: linear splines minimize

$$\int_{x_0}^{x_n} \ell'(x)^2\,dx \qquad \text{subject to } \ell(x_i) = y_{i,} \quad \text{for } i = 0, 1, 2, \ldots, n. \tag{4.2.1}$$

Linear splines $\ell$ are continuous, but the derivative $\ell'$ is generally discontinuous at each interpolation point $x_i$. Standard polynomial interpolation can be applied on each piece $[x_i, x_{i+1}]$; if $y_j = f(x_j)$, the polynomial interpolation error formula (4.1.7) can applied to give:

$$f(x) - \ell(x) = \frac{1}{2!} f''(c_{i,x})(x - x_i)(x - x_{i+1}) = \mathcal{O}((x_{i+1} - x_i)^2).$$

*Cubic splines* [71] are piecewise cubic functions $s$ that are continuous and have continuous first and second derivatives. A cubic spline interpolant $s$ for data points $(x_i, y_i)$, $i = 0, 1, 2, \ldots, n$ with $x_i < x_{i+1}$, is a cubic spline where $s(x)$ is a cubic polynomial for $x_i \le x \le x_{i+1}$. We can represent $s(x) = s_i(x)$ for $x_i \le x \le x_{i+1}$ for $i = 0, 1, \ldots, n-1$, where $s_i(x)$ is a cubic polynomial. We can represent

$$s_i(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i. \tag{4.2.2}$$

The problem then is to determine the unknown coefficients $a_i, b_i, c_i, d_i$ for $i = 0, 1, 2, \ldots, n-1$. This is a total of $4n$ unknowns to find. The equations to be satisfied are

$$s_i(x_i) = y_i, \qquad s_i(x_{i+1}) = y_{i+1} \qquad \text{for } i = 0, 1, 2, \ldots, n-1, \tag{4.2.3}$$

$$s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}), \qquad \text{for } i = 0, 1, 2, \ldots, n-2, \tag{4.2.4}$$

$$s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}), \qquad \text{for } i = 0, 1, 2, \ldots, n-2. \tag{4.2.5}$$

Equation (4.2.3) represents the interpolation conditions. Note that $s_i(x_{i+1}) = y_{i+1} = s_{i+1}(x_{i+1})$, so the interpolation conditions ensure that $s(x)$ is continuous. Equations (4.2.4), (4.2.5) ensure continuity of the first and second derivatives at the "joints" $x_j$, $j = 1, 2 \ldots, n-1$, where adjacent pieces meet.

The total number of equations is therefore $2n + 2(n-1) = 4n - 2$. That is, there are two more unknowns than equations. We can specify a unique interpolant with two additional conditions, provided the linear system is invertible. There are many ways of adding two additional equations. The most commonly used are

- *Natural spline*: $s''(x_0) = s''(x_n) = 0$.
- *Clamped spline*: $s'(x_0) = y_0'$ and $s'(x_n) = y_n'$ for given $y_0'$ and $y_n'$.
- *Not-a-knot spline*: $s'''(x)$ continuous at $x = x_1$ and $x = x_{n-1}$.
- *Periodic spline*: provided $y_0 = y_n$ we set $s'(x_0) = s'(x_n)$ and $s''(x_0) = s''(x_n)$.

The natural spline is, in fact, the solution $s$ to the optimization problem

$$\min_s \int_{x_0}^{x_n} s''(x)^2\,dx \qquad \text{subject to } s(x_i) = y_i, \quad i = 0, 1, 2, \ldots, n, \tag{4.2.6}$$

where the minimum is taken over all $s$ where $s''$ is integrable. Clamped splines are the minimizers of (4.2.6) with the additional constraints that $s'(x_0) = y_0'$ and $s'(x_n) = y_n'$. Periodic splines are the minimizers of (4.2.6) with the additional constraints that $s$ is periodic with period $x_n - x_0$.

#### 4.2.1.1 Computing Cubic Splines

Each of the four kinds of splines identified above can be solved via linear systems. The task here is to explicitly give a system that can be efficiently and accurately solved. The approach taken here follows [13, 241]. Since the spline $s(x)$ is piecewise cubic with continuous first and second derivatives, then $s''(x)$ is continuous and piecewise linear. Let $M_i = s''(x_i)$, $i = 0, 1, 2, \ldots, n$. Then for $x_i \le x \le x_{i+1}$ we have

$$s''(x) = s_i''(x) = M_i \frac{x - x_{i+1}}{x_i - x_{i+1}} + M_{i+1} \frac{x - x_i}{x_{i+1} - x_i},$$

using linear Lagrange interpolating polynomials. This choice of variables ensures that $s''$ is continuous. Let $h_i = x_{i+1} - x_i$ for $i = 0, 1, 2, \ldots, n-1$. Integrating twice we get

$$s_i(x) = \frac{M_i}{6h_i}(x_{i+1} - x)^3 + \frac{M_{i+1}}{6h_i}(x - x_i)^3 + C_i(x_{i+1} - x) + D_i(x - x_i)$$

with two constants of integration $C_i$ and $D_i$. The interpolation Equation (4.2.3) for $s_i$ becomes

$$\begin{aligned} y_i = s_i(x_i) &= \frac{M_i}{6h_i}h_i^3 + \frac{M_{i+1}}{6h_i}0^3 + C_i h_i + D_i 0 = \frac{1}{6}M_i h_i^2 + C_i h_i, \\ y_{i+1} = s_i(x_{i+1}) &= \frac{M_i}{6h_i}0^3 + \frac{M_{i+1}}{6h_i}h_i^3 + C_i 0 + D_i h_i = \frac{1}{6}M_{i+1}h_i^2 + D_i h_i. \end{aligned}$$

These equations allow us to solve for $C_i$ and $D_i$:

$$C_i = (y_i/h_i) - \frac{1}{6}M_i h_i, \qquad D_i = (y_{i+1}/h_i) - \frac{1}{6}M_{i+1}h_i.$$

We still have to ensure continuity of the first derivative of $s(x)$: $s_{i-1}'(x_i) = s_i'(x_i)$ for $i = 1, 2, \ldots, n-1$. The derivative $s_i'(x)$ is

$$\begin{aligned} s_i'(x) &= -\frac{M_i}{2h_i}(x_{i+1} - x)^2 + \frac{M_{i+1}}{2h_i}(x - x_i)^3 + D_i - C_i \\ &= -\frac{M_i}{2h_i}(x_{i+1} - x)^2 + \frac{M_{i+1}}{2h_i}(x - x_i)^3 + \frac{y_{i+1} - y_i}{h_i} - \frac{1}{6}(M_{i+1} - M_i)h_i. \end{aligned}$$

Thus

$$s_i'(x_i) = -\frac{M_i}{2h_i}h_i^2 + \frac{y_{i+1}-y_i}{h_i} - \frac{1}{6}(M_{i+1}-M_i)h_i, \qquad \text{and}$$
$$s_{i-1}'(x_i) = +\frac{M_i}{2h_{i-1}}h_{i-1}^2 + \frac{y_i-y_{i-1}}{h_{i-1}} - \frac{1}{6}(M_i-M_{i-1})h_{i-1}.$$

Equating these derivatives gives

$$\frac{1}{6}h_{i-1}M_{i-1} + \frac{1}{3}(h_{i-1}+h_i)M_i + \frac{1}{6}h_iM_{i+1} = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}} \tag{4.2.7}$$

for $i = 1, \ldots, n-1$. We are still missing the two additional equations, which depend on the type of cubic spline.

- Natural spline: $M_0 = M_n = 0$.
- Clamped spline: $s_0'(x_0) = y_0'$ and $s_{n-1}'(x_n) = y_n'$ so

$$\frac{y_1-y_0}{h_0} - y_0' = \frac{1}{2}M_0h_0 + \frac{1}{6}(M_1-M_0)h_0 = h_0\left(\frac{1}{3}M_0 + \frac{1}{6}M_1\right),$$
$$y_n' - \frac{y_n-y_{n-1}}{h_{n-1}} = \frac{1}{2}M_nh_{n-1} - \frac{1}{6}(M_n-M_{n-1})h_{n-1} = h_{n-1}\left(\frac{1}{6}M_{n-1} + \frac{1}{3}M_n\right).$$

- Not-a-knot spline: $(M_1-M_0)/h_0=(M_2-M_1)/h_1$ and $(M_{n-1}-M_{n-2})/h_{n-2} = (M_n-M_{n-1})/h_{n-1}$.
- Periodic spline: $M_0 = M_n$ and

$$\frac{y_1-y_0}{h_0} - \frac{y_n-y_{n-1}}{h_{n-1}} = \frac{1}{6}h_{n-1}M_{n-1} + \frac{1}{3}(h_{n-1}+h_0)M_0 + \frac{1}{6}h_0M_1,$$

assuming $y_n = y_0$.

The equations to solve are tridiagonal (see Section 2.3.1) for the natural and clamped splines. The equations to solve for the not-a-knot and periodic splines are rank-2 modifications of tridiagonal matrices for which we can apply the Sherman–Morrison (2.1.16) or Sherman–Morrison–Woodbury (2.1.17) formulas.

To see that the linear systems are invertible we start with the natural spline equations:

$$\begin{bmatrix} \frac{1}{3}(h_0+h_1) & \frac{1}{6}h_1 & & & \\ \frac{1}{6}h_1 & \frac{1}{3}(h_1+h_2) & \frac{1}{6}h_2 & & \\ & \frac{1}{6}h_2 & \frac{1}{3}(h_2+h_3) & \ddots & \\ & & \ddots & \ddots & \frac{1}{6}h_{n-2} \\ & & & \frac{1}{6}h_{n-2} & \frac{1}{3}(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ \vdots \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \cdots \\ b_{n-1} \end{bmatrix}.$$

This matrix is symmetric and also diagonally dominant as $\frac{1}{3}(h_k + h_{k+1}) > \frac{1}{6}h_k + \frac{1}{6}h_{k+1}$. Since the matrix is diagonally dominant, it is invertible, and LU factorization does not need partial pivoting (see 2.1.4.4).

#### 4.2.1.2 Error Estimates for Cubic Splines

We can get optimal $\mathcal{O}(h^4)$ error estimates for clamped cubic spline interpolation. In fact, we can also get error estimates for the $k$th derivatives up to $k = 3$ of order $\mathcal{O}(h^{4-k})$, which is also optimal. There is a small catch, which is that we need a bound on $\max_i \max(h_{i-1}/h_i,\ h_i/h_{i-1})$ and on $\max_i h_i/\min_i h_i$. Note that similar results can be obtained for the "not-a-knot" and periodic splines. Natural splines still have good behavior, thanks to the rapid decay of errors from the endpoints to the interior.

**Theorem 4.8** *If $f$ has continuous fourth-order derivative and $s$ is the clamped cubic spline interpolant of $s(x_i) = f(x_i)$, $i = 0, 1, 2, \ldots, n$ with $a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$ then*

$$\max_{a\le x\le b} |f(x) - s(x)| = \mathcal{O}(L\,K^{1/2}\,h^4), \qquad \textit{and}$$
$$\max_{a\le x\le b} \left|f^{(j)}(x) - s^{(j)}(x)\right| = \mathcal{O}(L\,K^{1/2}\,h^{4-j}), \qquad \textit{for } j = 1, 2, 3,$$

*as $h \to 0$ where $L = \max_{a\le x\le b} \left|f^{(4)}(x)\right|$, $K = \max_i \max(h_{i-1}/h_i,\ h_i/h_{i-1})$, and $h = \max_i h_i$. The hidden constants in the "$\mathcal{O}$" are absolute constants except for $j = 3$ where there is an additional factor of $h/\min_i h_i$.*

***Proof*** Let us assume that $y_i = f(x_i)$ and $y_i' = f'(x_i)$, etc. To get the error estimates started, we need to show that $M_i = f''(x_i) + \mathcal{O}(h^2) = y_i'' + \mathcal{O}(h^2)$ with $h = \max_i h_i$. The starting point for this is the linear system we have generated. Let us write the clamped cubic spline equations (4.2.7) as $A\boldsymbol{M} = \boldsymbol{b}$:

(4.2.8)
$$\begin{bmatrix} \frac{1}{3}h_0 & \frac{1}{6}h_0 & & & & & \\ \frac{1}{6}h_0 & \frac{1}{3}(h_0+h_1) & \frac{1}{6}h_1 & & & & \\ & \frac{1}{6}h_1 & \frac{1}{3}(h_1+h_2) & \ddots & & & \\ & & \ddots & \ddots & \frac{1}{6}h_{n-2} & & \\ & & & \frac{1}{6}h_{n-2} & \frac{1}{3}(h_{n-2}+h_{n-1}) & \frac{1}{6}h_{n-1} & \\ & & & & \frac{1}{6}h_{n-1} & \frac{1}{3}h_{n-1} \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ M_2 \\ \vdots \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ \vdots \\ b_{n-1} \\ b_n \end{bmatrix},$$

where $b_i = (y_{i+1} - y_i)/h_i - (y_i - y_{i-1})/h_{i-1}$ for $i = 1, 2, \ldots, n-1$, $b_0 = (y_1 - y_0)/h_0 - y_0'$, and $b_n = y_n' - (y_n - y_{n-1})/h_{n-1}$.

The first step is to show that the inverse of a scaled version of our matrix is bounded independently of $h > 0$. If we multiply (4.2.7) by $1/\sqrt{h_{i-1}h_i}$ we get

$$\frac{1}{6}\sqrt{\frac{h_{i-1}}{h_i}}M_{i-1}+\frac{1}{3}\left(\sqrt{\frac{h_{i-1}}{h_i}}+\sqrt{\frac{h_i}{h_{i-1}}}\right)M_i+\frac{1}{6}\sqrt{\frac{h_i}{h_{i-1}}}M_{i+1}=\frac{b_i}{\sqrt{h_{i-1}h_i}}$$

for $i = 1, 2, \ldots, n-1$. For the first and last rows of (4.2.8), we can multiply by $1/h_0$ and $1/h_{n-1}$, respectively. The scaled matrix $A'$ is diagonally dominant and the diagonal entries satisfy

$$\frac{1}{3}\left(\sqrt{\frac{h_{i-1}}{h_i}}+\sqrt{\frac{h_i}{h_{i-1}}}\right)\geq\frac{2}{3}.$$

Write $A' = D' - B$ where $D'$ is the diagonal of $A'$ so that

$$\begin{aligned}\left(A'\right)^{-1}&=\left(D'-B\right)^{-1}=\left(D'\left[I-\left(D'\right)^{-1}B\right]\right)^{-1}\\&=\left[I-\left(D'\right)^{-1}B\right]^{-1}\left(D'\right)^{-1}\\&=\left[I+\left(D'\right)^{-1}B+\left(\left(D'\right)^{-1}B\right)^2+\left(\left(D'\right)^{-1}B\right)^3+\cdots\right]\left(D'\right)^{-1}.\end{aligned}$$

But $\left\|\left(D'\right)^{-1}B\right\|_\infty\leq 1/2$, and so $\left\|\left(A'\right)^{-1}\right\|_\infty\leq 2\left\|\left(D'\right)^{-1}\right\|_\infty\leq 3$.

Now if we let $F_i = f''(x_i) = y_i''$, then to estimate $\|\boldsymbol{M}-\boldsymbol{F}\|_\infty$, we first compute $\|A\,(\boldsymbol{M}-\boldsymbol{F})\|_\infty$. Estimating $\|A\,(\boldsymbol{M}-\boldsymbol{F})\|_\infty$ is an exercise in Taylor series. Writing $f^{(k)}(x_i)$ as $y_i^{(k)}$, we see that we can write the Taylor series as

$$\begin{aligned}y_{i+1}&=y_i+h_iy_i'+\frac{1}{2}h_i^2y_i''+\frac{1}{6}h_i^3y_i'''+\frac{1}{24}h_i^4\eta_i,\\y_{i-1}&=y_i-h_{i-1}y_i'+\frac{1}{2}h_{i-1}^2y_i''-\frac{1}{6}h_{i-1}^3y_i'''+\frac{1}{24}h_{i-1}^4\widetilde{\eta}_i,\\y_{i+1}''&=y_i''+h_iy_i'''+\frac{1}{2}h_i^2\mu_i,\\y_{i-1}''&=y_i''-h_{i-1}y_i'''+\frac{1}{2}h_{i-1}^2\widetilde{\mu}_i,\end{aligned}$$

where $\eta_i$, $\widetilde{\eta}_i$, $\mu_i$, $\widetilde{\mu}_i$ are fourth derivatives of $f$ evaluated at certain points in the interval $(x_{i-1},\ x_{i+1})$. Then noting that $\boldsymbol{\gamma} := A\,(\boldsymbol{M}-\boldsymbol{F}) = A\boldsymbol{M}-A\boldsymbol{F} = \boldsymbol{b}-A\boldsymbol{F}$, we have for $i = 1,\ 2,\ \ldots,\ n-1$,

$$\begin{aligned}\gamma_i&=\frac{y_{i+1}-y_i}{h_i}-\frac{y_i-y_{i-1}}{h_{i-1}}-\left[\frac{h_{i-1}}{6}y_{i-1}''+\frac{h_{i-1}+h_i}{3}y_i''+\frac{h_i}{6}y_{i+1}''\right]\\&=\frac{1}{2}h_iy_i''+\frac{1}{6}h_i^2y_i'''+\frac{1}{24}h_i^3\eta_i+\frac{1}{2}h_{i-1}y_i''-\frac{1}{6}h_{i-1}^2y_i'''+\frac{1}{24}h_{i-1}^3\widetilde{\eta}_i\\&\quad-\left[\frac{h_{i-1}}{6}y_{i-1}''+\frac{h_{i-1}+h_i}{3}y_i''+\frac{h_i}{6}y_{i+1}''\right]\end{aligned}$$

$$= \frac{1}{2}(h_i + h_{i-1})\, y_i'' + \frac{1}{6}\left(h_i^2 - h_{i-1}^2\right) y_i''' + \mathcal{O}\left(L(h_i^3 + h_{i-1}^3)\right)$$
$$- \frac{1}{2}(h_i + h_{i-1})\, y_i'' - \frac{1}{6}\left(h_i^2 - h_{i-1}^2\right) y_i''' + \mathcal{O}\left(L(h_i^3 + h_{i-1}^3)\right)$$
$$= \mathcal{O}\left(L(h_i^3 + h_{i-1}^3)\right), \tag{4.2.9}$$

where $L := \max_x \left|f^{(4)}(x)\right|$. After scaling from $A$ to $A'$, we change the linear system to $A'(\boldsymbol{M} - \boldsymbol{F}) = \boldsymbol{\gamma}'$ with

$$\left|\gamma_i'\right| \le \text{const}\, L\, \frac{h_{i-1}^3 + h_i^3}{\sqrt{h_{i-1} h_i}}$$
$$\le \text{const}\, L\, \left(h_{i-1}^2 + h_i^2\right) \max\left(\frac{h_{i-1}}{h_i}, \frac{h_i}{h_{i-1}}\right)^{1/2}$$

for all $i$. Then using the bound on $\left(A'\right)^{-1}$, we obtain

$$\|\boldsymbol{M} - \boldsymbol{F}\|_\infty \le \text{const}\, L\, h^2\, K^{1/2}$$

where $K = \max_i \max(h_{i-1}/h_i,\ h_i/h_{i-1})$. That is, $M_i - f''(x_i) = \mathcal{O}(L\, K^{1/2}\, h^2)$.

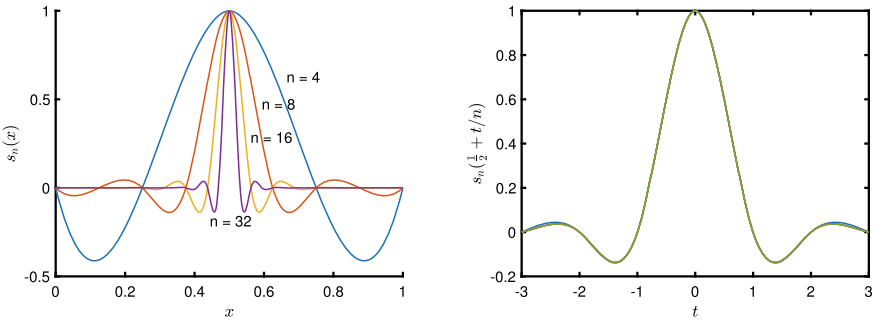
Since $f^{(4)}$ is continuous and $s''(x)$ is the piecewise linear interpolant of $M_i \approx f''(x_i)$, the error in $s''(x)$ can be estimated by the error in piecewise interpolation of $f''(x)$, which is $\mathcal{O}(L\, h^2)$ plus the error due to the approximation $M_i \approx f''(x_i)$. Thus the error in $s''(x)$ is $M_i - f''(x_i) = \mathcal{O}(L\, K^{1/2}\, h^2)$.

The error in $s'''(x) = (M_{i+1} - M_i)/h_i$ for $x_i \le x \le x_{i+1}$ can be estimated by the error in $f'''(x) \approx \left(f''(x_{i+1}) - f''(x_i)\right)/h_i$ plus bounds for the errors in $M_i \approx f''(x_i)$ over $h_i$, which gives a bound of $\mathcal{O}(L\, K^{1/2}\, h\ (h/\min_i h_i))$.

Since $s(x_i) = f(x_i)$, for all $i$, by Rolle's theorem, there is a point $\xi_i \in (x_i,\ x_{i+1})$ where $s'(\xi_i) = f'(\xi_i)$. Integrating from $\xi_i$ to the remainder of the interval $\left[x_i,\ x_{i+1}\right]$ using the error bound for $s''$ shows that the error in $s'$ is only $\mathcal{O}(L\, K^{1/2}\, h^3)$. Finally, integrating this error from $x_i$ to $x \in \left[x_i,\ x_{i+1}\right]$ shows that the error in $s(x)$ is $\mathcal{O}(L\, K^{1/2}\, h^4)$. □

It should be noted that this proof indicates that it is possible to accurately interpolate less smooth functions by adapting the spacing $h_i = x_{i+1} - x_i$ according to the size of $f^{(4)}(x)$ for $x_{i-1} \le x \le x_{i+1}$. The estimate of $\gamma_i$ (4.2.9), where $A(\boldsymbol{M} - \boldsymbol{F}) = \boldsymbol{\gamma}$, can be controlled by $\mathcal{O}(\max_{x_{i-1} \le x \le x_{i+1}} \left|f^{(4)}(x)\right| (h_i^3 + h_{i-1}^3))$. Scaling by $1/\sqrt{h_i h_{i-1}}$ increases this bound on $\gamma_i'$ to $\mathcal{O}(\max_{x_{i-1} \le x \le x_{i+1}} \left|f^{(4)}(x)\right| K^{1/2}$ $(h_i^2 + h_{i-1}^2))$. However, by reducing $h_i \approx h^* \left|f^{(4)}(x_i)\right|^{-1/2}$ gives $\gamma_i' = \mathcal{O}(K^{1/2}(h^*)^2)$. The size of $K$ can be controlled by making sure that the spacing does not change dramatically between adjacent interpolation points. The error $\|\boldsymbol{M} - \boldsymbol{F}\|_\infty$ in $M_i \approx F_i = f''(x_i)$ is then $\mathcal{O}((h^*)^2)$, and corresponding error bounds can be found for $s''(x)$ and the lower derivatives of $s$.

One of the advantages of cubic spline interpolation is that the error due to the perturbation of one data point decays exponentially rapidly as you move away from




(A) Not-a-knot splines $n = 4, 8, 16, 32$

(B) Zoomed not-a-knot splines, $n = 8, 10, 12, 14, 16$

**Fig. 4.2.1** Decay of perturbations for equally spaced cubic spline interpolation

the perturbed data point. Note that the decay rate is independent of $h$ as a rate *per interpolation point*. Figure 4.2.1 shows not-a-knot cubic spline interpolants for the data points $(x_i, y_i)$, $i = 0, 1, 2, \ldots, n$, with $x_i = i\,h$ and $y_i = 0$ except for $y_{n/2} = 1$ (taking $n$ even). The perturbation therefore occurs at $x = 1/2$. The claim that the perturbations decay at a roughly constant rate *per interpolation point* can be seen in Figure 4.2.1(b).

The rate of decay can be determined for constant $h$ by using (4.2.7) with zero right-hand side:

$$\begin{aligned} &\frac{1}{6}hM_{i-1} + \frac{1}{3}(h+h)M_i + \frac{1}{6}hM_{i+1} = 0, \qquad \text{so} \\ &\frac{1}{6}\ M_{i-1} + \frac{2}{3}\qquad\quad M_i + \frac{1}{6}\ M_{i+1} = 0. \end{aligned}$$

Solving this linear recurrence relation we get $M_j = a_1 r_1^j + a_2 r_2^j$ where $r_1$ and $r_2$ are the roots of the characteristic equation $\frac{1}{6} + \frac{2}{3}r + \frac{1}{6}r^2 = 0$; these roots are $-2 \pm \sqrt{3}$. Note that $r_1 \cdot r_2 = 1$, so we can write $M_i = a_1 r_1^j + a_2 r_1^{-j}$. Around a perturbed interpolation value, the perturbations decay with a factor of roughly $\left|-2+\sqrt{3}\right| \approx 0.268$ per interpolation point.

The reliability of spline interpolation can be confirmed by estimation of the Lebesgue constant for equally spaced not-a-knot spline interpolation. The functions $L_i(x)$ are the interpolants of $L_i(x_j) = 1$ if $i = j$ and $L_i(x_j) = 0$ if $i \neq j$. These Lebesgue constants are shown in Figure 4.2.2.

As apparent in Figure 4.2.2, the Lebesgue constants do not grow rapidly with $n$. In fact, they appear to be bounded with a bound a little less than two. Using $n = 1000$, the Lebesgue constant computed for not-a-knot splines is about 1.965. This combined with the exponential decay of perturbations makes cubic spline interpolation very robust.

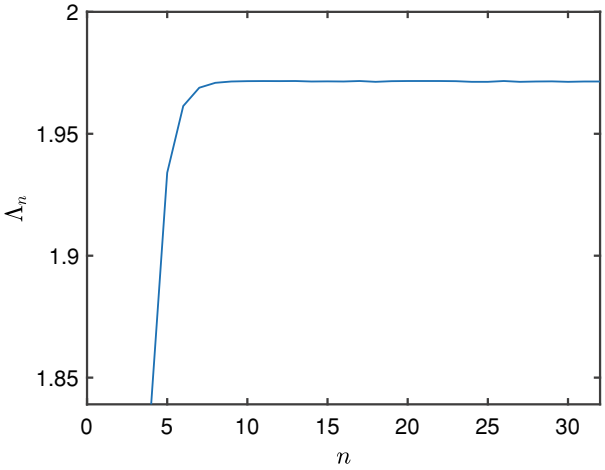



**Fig. 4.2.2** Lebesgue constants for not-a-knot spline interpolation with $n \geq 4$ and equally spaced points

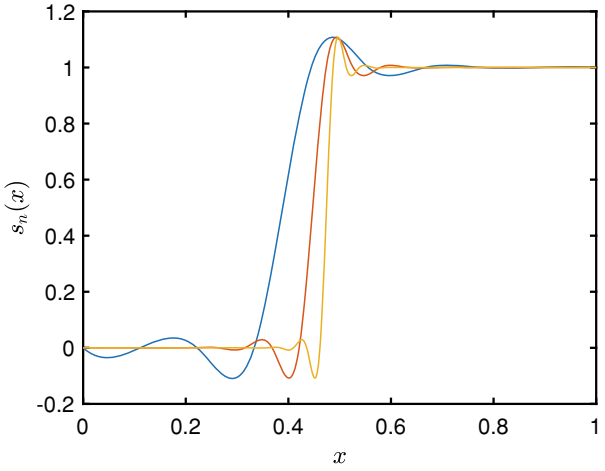



**Fig. 4.2.3** Overshoot of not-a-knot spline interpolation $n = 9,\ 19,\ 39$

Notwithstanding the excellent character of cubic spline interpolation, it still has some oscillation when interpolating discontinuous data. For example, for a step function interpolated by not-a-knot cubic splines, we obtain the results shown in Figure 4.2.3 for $n = 9,\ 19,\ 39$. As we can see, the overshoot does not go to zero as $n \to \infty$.

### 4.2.2 Higher Order Splines in One Variable

There is a natural generalization of cubic splines in one variable based on the minimization principles (4.2.1) for piecewise linear interpolation and (4.2.6) for cubic splines. We can ask to find

$$\min_s \int_{x_0}^{x_n} s'''(x)^2\,dx \qquad \text{subject to } s(x_i) = y_i, \quad i = 0, 1, 2, \ldots, n. \tag{4.2.10}$$

The minimizer $s$ of (4.2.10) is piecewise quintic (fifth order) polynomial that has continuous fourth derivatives (not just third derivatives). Combining these conditions with the interpolation property gives $6n - 4$ equations for $6n$ unknown coefficients of $s(x) = a_i x^5 + b_i x^4 + c_i x^3 + d_i x^2 + e_i x + f_i$ for $x_i \le x \le x_{i+1}$. What four extra conditions are imposed give different types of quintic splines. Natural quintic splines have the properties that $s'(x) = s''(x) = 0$ at $x = x_0,\ x_n$; clamped quintic splines have the properties that $s'(x_0) = y_0'$, $s''(x_0) = y_0''$, $s'(x_n) = y_n'$, and $s''(x_n) = y_n''$ are all specified. Computing one of these quintic splines given the interpolation data involves solving a penta-diagonal (5-diagonal) symmetric positive-definite linear system.

Quintic splines have the advantage of slightly better smoothness and less oscillation in the interpolants. They have the disadvantage of greater computational cost to implement.

**Exercises.**

(1) Use not-a-knot cubic splines to interpolate $f(x) = e^{-x}/(1 + x)$ over $[0, 1]$ using $n + 1$ equally spaced interpolation points with $n = 5,\ 10,\ 20,\ 40,\ 100$. Estimate the maximum error between $f$ and the spline interpolants using 1001 points equally spaced over $[0, 1]$. Plot the maximum error against $n$. Estimate the exponent $\alpha$ where the maximum error is asymptotically $C\,h^\alpha$. Does this confirm the theoretical error estimate of $\mathcal{O}(h^4)$?

(2) Numerically estimate the Lebesgue constant of not-a-knot spline interpolation by finding $\max_{0\le x\le 1} \sum_{k=0}^{n} |\ell_k(x)|$ where $\ell_k$ is the not-a-knot spline function interpolating $\ell_k(x_j) = 1$ if $j = k$ and zero if $j \ne k$. Use equally spaced interpolation points $x_j = j/n$ for $j = 0, 1, 2, \ldots, n$. Do this for $n = 5,\ 10,\ 20,\ 40,\ 100$.

(3) To see the exponential decay of perturbations, compute the not-a-knot spline interpolant of the data $y_j = 0$ for $0 \le j \le N$ except that $y_{N/2} = 1$ assuming $N$ even; also set $x_j = j,\ j = 0, 1, 2, \ldots, N$. Do this for $N = 100$. Estimate the exponential rate of decay of the spline interpolant $s(x_j)$ as $|j - N/2|$ increases. Repeat this with $x_j = j/N$.

(4) Compute not-a-knot, clamped, and natural spline interpolants of $f(x) = e^{-x}/(1 + x)$ over $[0, 1]$ using $n + 1$ equally spaced interpolation points with $n = 5,\ 10,\ 20,\ 40,\ 100$. Plot the errors for the different interpolants. Which has smallest maximum error? Where are the differences between the different versions of cubic spline interpolants?

(5) Show that natural splines minimize $\int_{x_0}^{x_n} \left[s''(x)\right]^2 \, dx$ subject to $s(x_j) = y_j$ for $j = 0, 1, 2, \ldots, n$. [**Hint:** First show that between consecutive interpolation points, $s''''(x) = 0$, so the function must be piecewise cubic. Do this by using

$$\int_{x_0}^{x_n} \left[s''(x)\right]^2 \, dx \le \int_{x_0}^{x_n} \left[s''(x) + t\,\eta''(x)\right]^2 \, dx$$

for all $t \in \mathbb{R}$ and $\eta(x)$ smooth and $\eta(x_j) = 0$ for all $j$. Use integration by parts on each piece $[x_j, x_{j+1}]$ to put the derivatives on $s$.]

(6) Show that clamped splines minimize $\int_{x_0}^{x_n} \left[s''(x)\right]^2 \, dx$ subject to $s(x_j) = y_j$ for $j = 0, 1, 2, \ldots, n$, and the clamping conditions $s'(x_0) = y_0'$ and $s'(x_n) = y_n'$.

(7) Use equally spaced interpolation points and not-a-knot splines to interpolate $f(x) = \sqrt{x}$ on $[0, 1]$. Use $n + 1$ interpolation points for $n = 5,\ 10,\ 20,\ 40,\ 100$. Plot the maximum error against the spacing $h = 1/n$ in a log–log plot. What relationship do you see between $h$ and the maximum error?

(8) For the task of Exercise 7, instead of using equally spaced interpolation points, we can try using a graded mesh to deal with the singularity of $\sqrt{x}$ at $x = 0$, use interpolation points $x_k = hk^\gamma / N^{\gamma-1}$ for $k = 0, 1, 2, \ldots, N$. Use $\gamma = 1,\ 1\frac{1}{2},\ 2,\ 2\frac{1}{2},\ 3$ and $N = 2^\ell$, $\ell = 1, 2, \ldots, 10$. Plot the maximum error against $N$ for each value of $\gamma$ used. Estimate the exponents $\alpha$ where the maximum error appears to be $\approx$ constant $N^{-\alpha}$ for each value of $\gamma$ used.

(9) Do two-dimensional spline interpolation as follows: Given interpolation points $\widehat{x}_i$, $i = 0, 1, \ldots, M$, in the $x$-axis and $\widehat{y}_j$, $j = 0, 1, 2, \ldots, N$, and values to interpolate $z_{ij}$, we want a function $s(x_i, y_j) = z_{ij}$. For each $j$, find a vector representation $\boldsymbol{r}_j$ of the splines along the $x$-axis with $y = \widehat{y}_j$. There must be a universal function spline$(x;\ \boldsymbol{r},\ \widehat{\boldsymbol{x}})$ that computes the value of the spline represented by $\boldsymbol{r}$ at the point $x$ with interpolation points $\widehat{\boldsymbol{x}}$. Apply spline interpolation to create the vector-valued spline function $\boldsymbol{r}(y)$ where $\boldsymbol{r}(y_j) = \boldsymbol{r}_j$. Then $s(x, y) = \text{spline}(x;\ \boldsymbol{r}(y),\ \widehat{\boldsymbol{x}})$. Implement this approach in your favorite programming language. Test your implementation for interpolating the function $f(x, y) = \exp(-x^2 - xy - \frac{1}{2}y^2)$ over the rectangle $[-2,\ +2] \times [-2,\ +2]$ with $M = N = 2^\ell$, $\ell = 2, 3, \ldots, 6$. [**Note:** It is important that your representation is a *linear* one. That is, spline$(x;\ a_1\boldsymbol{r}_1 + a_2\boldsymbol{r}_2,\ \widehat{\boldsymbol{x}}) = a_1\text{spline}(x;\ \boldsymbol{r}_1,\ \widehat{\boldsymbol{x}}) + a_2\text{spline}(x;\ \boldsymbol{r}_2,\ \widehat{\boldsymbol{x}})$ for any $a_1,\ a_2$, and $\boldsymbol{r}_1,\ \boldsymbol{r}_2$.]

(10) Show that fifth-order natural spline interpolants for interpolation points $a = x_0 < x_1 < \cdots < x_{N-1} < x_N = b$ can be defined by minimizing $\int_a^b \left[s'''(x)\right]^2 \, dx$ subject to the condition that $s(x_i) = y_i$ for $i = 0, 1, 2, \ldots, N$. Show that these fifth-order spline functions are piecewise polynomials of degree $\le 5$, and are continuous with continuous derivatives up to the fourth order.

## 4.3 Interpolation—Triangles and Triangulations

### *4.3.1 Interpolation over Triangles*

In one dimension, the basic shapes are usually very simple: intervals. In two dimensions, there is a much greater choice, and in three dimensions, the set of basic shapes is even larger.

In two dimensions, we focus on triangles. Polygons can be decomposed into triangles. Domains with curved boundaries can be approximated by unions of non-overlapping triangles. The triangles in the union should be "non-overlapping" at least in the sense that intersections of different triangles in the union are either vertices or edges. In three dimensions, we focus on tetrahedra, and simplices in four and higher dimensions, where similar methods and behavior apply.

For many calculations, it is convenient to use *barycentric coordinates* to represent points in a triangle:

$$\begin{aligned} &\boldsymbol{x} = \lambda_1 \boldsymbol{v}_1 + \lambda_2 \boldsymbol{v}_2 + \lambda_3 \boldsymbol{v}_3 \qquad \text{where} \\ &0 \le \lambda_1, \lambda_2, \lambda_3 \text{ and } \lambda_1 + \lambda_2 + \lambda_3 = 1, \end{aligned} \tag{4.3.1}$$

and $\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3$ are the vertices of the triangle. Note that the triangle $T$ with these vertices is the set of all convex combinations of $\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3$; we write $T = \text{co}\,\{\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3\}$. Furthermore, each point in $T$ can be represented uniquely in the form of (4.3.1). The barycentric coordinates for a given point $\boldsymbol{x} \in T$ are written as $\lambda_1(\boldsymbol{x}),\ \lambda_2(\boldsymbol{x}),\ \lambda_3(\boldsymbol{x})$. Note that the vector of barycentric coordinates $\boldsymbol{\lambda}(\boldsymbol{x})$ is an *affine function* of $\boldsymbol{x}$: $\boldsymbol{\lambda}(\boldsymbol{x}) = A\boldsymbol{x} + \boldsymbol{b}$ for some matrix $A$ and vector $\boldsymbol{b}$.

Given a function $f : T \to \mathbb{R}$ we have the linear interpolant given by

$$p(\boldsymbol{x}) = f(\boldsymbol{v}_1)\,\lambda_1(\boldsymbol{x}) + f(\boldsymbol{v}_2)\,\lambda_2(\boldsymbol{x}) + f(\boldsymbol{v}_3)\,\lambda_3(\boldsymbol{x}). \tag{4.3.2}$$

Computing the barycentric coordinates can be done using some linear algebra: since $\boldsymbol{x} = [\boldsymbol{v}_1 \mid \boldsymbol{v}_2 \mid \boldsymbol{v}_3]\boldsymbol{\lambda}$ and $1 = [1 \mid 1 \mid 1]\boldsymbol{\lambda}$ we can combine them into

$$\begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{bmatrix} \boldsymbol{\lambda}, \qquad \text{so}$$

$$\boldsymbol{\lambda} = \begin{bmatrix} 1 & 1 & 1 \\ \boldsymbol{v}_1 & \boldsymbol{v}_2 & \boldsymbol{v}_3 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ \boldsymbol{x} \end{bmatrix}.$$

Every linear polynomial in two variables $(x, y)$ has the form $a_1 + a_2 x + a_3 y$; the space of linear polynomials in two variables has dimension three.

The space of quadratic polynomials in two variables has dimension six: $a_1 + a_2 x + a_3 y + a_4 x^2 + a_5 xy + a_6 y^2$. So a quadratic interpolation method will require six data values. These are typically taken to be the points shown in the middle triangle of Figure 4.3.1. In general, if $\mathcal{P}_{k,d}$ is the space of polynomials of degree $\le k$ in $d$

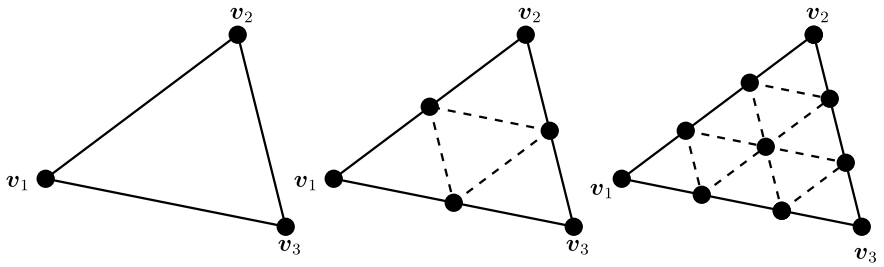



**Fig. 4.3.1** Triangles showing Lagrange interpolation points

variables, then

$$\dim \mathcal{P}_{k,d} = \binom{k+d}{k} = \binom{k+d}{d}. \tag{4.3.3}$$

For quadratic interpolation over a triangle, we use the vertices $\boldsymbol{v}_i$ and the midpoints $\boldsymbol{v}_{ij} = (\boldsymbol{v}_i + \boldsymbol{v}_j)/2$ for the evaluation points. Then the quadratic interpolant can be expressed in terms of barycentric coordinates as

$$p_2(\boldsymbol{x}) = \sum_{i=1}^{3} f(\boldsymbol{v}_i)\, \lambda_i(2\lambda_i - 1) + \sum_{i,j=1;\, i<j}^{3} f(\boldsymbol{v}_{ij})\, 4\lambda_i\lambda_j. \tag{4.3.4}$$

Cubic interpolation uses the vertices $\boldsymbol{v}_i$, points on the edges $\boldsymbol{v}_{ij} = (2\boldsymbol{v}_i + \boldsymbol{v}_j)/3$ (so that $\boldsymbol{v}_{ij} \neq \boldsymbol{v}_{ji}$) and $\boldsymbol{v}_{123} = (\boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_3)/3$ (which is the centroid of the triangle). The cubic interpolant can be expressed as

(4.3.5)

$$p_3(\boldsymbol{x}) = \frac{9}{2}\sum_{i=1}^{3} f(\boldsymbol{v}_i)\, \lambda_i(\lambda_i - \frac{1}{3})(\lambda_i - \frac{2}{3}) + \frac{27}{2}\sum_{i,j=1:i\neq j}^{3} f(\boldsymbol{v}_{ij})(\lambda_i - \frac{1}{3})\lambda_i\lambda_j + \\ + 27\, f(\boldsymbol{v}_{123})\, \lambda_1\lambda_2\lambda_3.$$

This can be continued to higher degree interpolation on a triangle.

In general, we can create a polynomial interpolant of degree $\leq k$ on a triangle, using interpolation points where each barycentric coordinate $\lambda_i$ is a multiple of $1/k$. This gives $\binom{k+2}{2}$ interpolation points as desired. This scheme for interpolation over triangles is called *Lagrange interpolation*. This can be extended to tetrahedra in three dimensions, and to simplices in even higher dimensions.

Finding a set of interpolation points in two or more dimensions is a more complex issue than for one dimension. In one dimension, as long as there are $k + 1$ distinct points, there is one and only one interpolating polynomial of degree $\leq k$. However, in two or more dimensions, having $\dim \mathcal{P}_{k,d}$ distinct points does not guarantee existence

or uniqueness of an interpolating polynomial. Consider, for example, having all the interpolating points on a circle in two dimensions: $(x-a)^2+(y-b)^2=c^2$. There are clearly infinitely many points on this circle. Even though, for example, $\dim \mathcal{P}_{2,2}=6$, if we choose six distinct points on this circle, then uniqueness fails as any multiple of $(x-a)^2+(y-b)^2-c^2$ can be added to a given interpolant and we still have an interpolant. Also, existence fails as the dimension of all quadratic polynomials of two variables restricted to this circle has dimension no more than five. That is, if the values of five of the six interpolation points are known, then the value at the sixth point must be a linear combination of the values at the first five. Even if not all six points are exactly on a circle (or ellipse, or hyperbola), if they are close to being so, then the interpolation problem becomes ill-conditioned: small perturbations are amplified greatly.

For determining if a set of interpolation points is suitable for a space of polynomials $\mathcal{P}$, such as $\mathcal{P}_{k,d}$ of all polynomials of degree $\leq k$ in $d$ variables, we can use the following theorem.

**Theorem 4.9** *A set $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ of distinct points is a set of acceptable interpolation points for the vector space $\mathcal{P}$ of interpolation functions if and only if $N=\dim \mathcal{P}$ and there is no non-zero $p\in\mathcal{P}$ where $p(\boldsymbol{x}_i)=0$ for all $i$.*

***Proof*** Let $\{\phi_1, \phi_2, \ldots, \phi_N\}$ be a basis for $\mathcal{P}$. Consider the linear transformation $T:\mathbb{R}^N\to\mathbb{R}^N$ given by $(T\boldsymbol{c})_i=\sum_{j=1}^N c_j\,\phi_j(\boldsymbol{x}_i)$. The set $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is a set of acceptable interpolation points if and only if $T$ is invertible. Since the matrix of $T$ with respect to any basis is $N\times N$, $T$ is invertible if and only if the only $\boldsymbol{c}$ where $T\boldsymbol{c}=\boldsymbol{0}$ is $\boldsymbol{c}=\boldsymbol{0}$. That is, the only $p=\sum_{j=1}^N c_j\,\phi_j\in\mathcal{P}$ where $p(\boldsymbol{x}_i)=0$ for all $i$ is when $p=0$, as we wanted. □

The contrapositive of Theorem 4.9 indicates that if there is a $p\in\mathcal{P}$ that is zero at all interpolation points, then the interpolation points are not acceptable. For example, for quadratic interpolation over two dimensions, for example, the vertices of a regular hexagon do not form an acceptable set of interpolation points, as they all lie on a circle. Fortunately, we can guarantee the existence of acceptable interpolation points.

**Theorem 4.10** *Suppose that $\mathcal{P}$ is a vector space of analytic functions[1] (which includes all polynomials) with $0<\dim\mathcal{P}<\infty$, then any $D$ with positive volume in $\mathbb{R}^d$ has a set of acceptable interpolation points.*

***Proof*** Let $\{\phi_1, \phi_2, \ldots, \phi_N\}$ be a basis for $\mathcal{P}$. Then $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is a set of acceptable interpolation points if and only if the matrix $A(\boldsymbol{x}_1, \boldsymbol{x}_2\ldots., \boldsymbol{x}_N)=[\phi_i(\boldsymbol{x}_j)]_{i,j=1}^N$ is invertible, or equivalently, $\det A(\boldsymbol{x}_1, \boldsymbol{x}_2\ldots., \boldsymbol{x}_N)\neq 0$. Since the $\phi_i$ are analytic functions, $\det A(\boldsymbol{x}_1, \boldsymbol{x}_2\ldots., \boldsymbol{x}_N)$ is an analytic function of $\left[\boldsymbol{x}_1^T, \boldsymbol{x}_2^T, \ldots, \boldsymbol{x}_N^T\right]^T$. The zero set of an analytic function [178] has zero volume. Therefore, as $D^N$ has

[1] A function $f$ is *analytic* on $D$ if for every $\boldsymbol{a}\in D$ the Taylor series of $f$ using derivatives at $\boldsymbol{a}$ converges for all $\boldsymbol{x}$ within a positive distance of $\boldsymbol{a}$, and $f$ is equal to its Taylor series where the Taylor series converges.

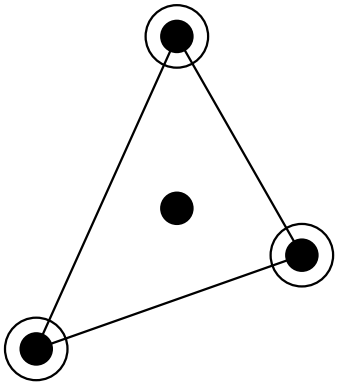


**Fig. 4.3.2** Cubic Hermite interpolation on a triangle

positive volume $\text{vol}_{Nd}(D^N) = \text{vol}_d(D)^N > 0$, there must be points in $D^N$ where $\det A(\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots, \boldsymbol{x}_N) \neq 0$ and so $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is a set of acceptable interpolation points. Furthermore, if $\det A(\boldsymbol{x}'_1, \boldsymbol{x}'_2 \ldots, \boldsymbol{x}'_N) = 0$ there must be points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ with $\boldsymbol{x}_i$ arbitrarily close to $\boldsymbol{x}'_i$ for which $\det A(\boldsymbol{x}_1, \boldsymbol{x}_2 \ldots, \boldsymbol{x}_N) \neq 0$ and so $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is a set of acceptable interpolation points. □

It should be noted that simply having existence and uniqueness of interpolants does not mean that a given set of interpolation points is a good set of such points. The quality of a set of interpolation points should be measured in terms of quantities such as the Lebesgue number (see Section 4.1.2) for an interpolation scheme.

#### 4.3.1.1 Hermite Triangle Element

We can create a kind of cubic Hermite interpolant on a triangle, interpolating first derivatives as well as values. This is illustrated in Figure 4.3.2. Note that "●" indicates that the *value* is interpolated at this point, while "◉" indicates that the *value and the gradient* are interpolated at this point. The point in the interior of the triangle at which the value is interpolated is the centroid of the triangle.

For general Hermite interpolants we do not restrict the order of the derivatives used, although this is usually much less than the degree of the polynomial interpolants. We do assume that there is an interpolation operator $\mathcal{I}\colon C^\ell(K) \to C^\ell(K)$ where $C^\ell(K)$ is the set of all functions $f\colon K \to \mathbb{R}$ where all derivatives of order $\leq \ell$ are continuous. For Hermite interpolation as described above (Figure 4.3.2), we take $\ell = 1$, while Lagrange interpolation schemes (Figure 4.3.1) have $\ell = 0$.

The Hermite cubic triangle interpolation system uses the values and gradients at the vertices together with the value at the centroid to uniquely specify the cubic interpolant. Each vertex contributes three interpolation conditions (the value plus two derivatives), so the centroid value provides the additional condition needed as

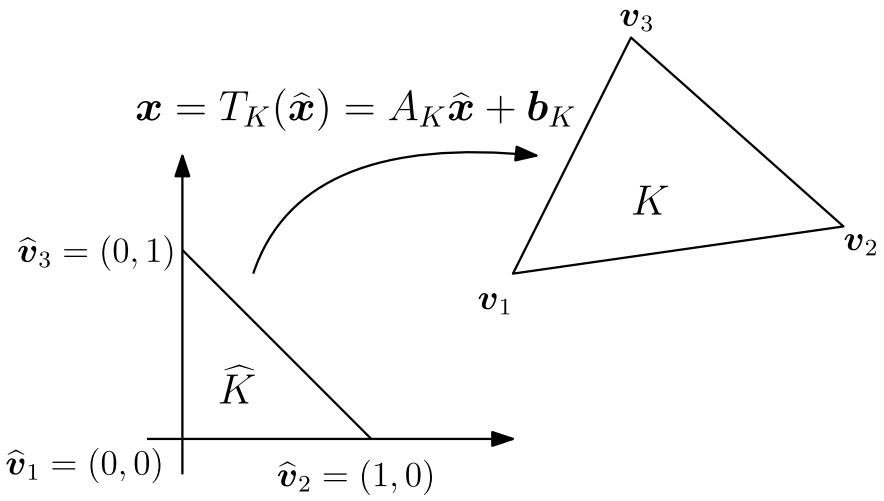



**Fig. 4.3.3** Affine transform from reference triangle $\widehat{K}$ to triangle $K$

$\dim \mathcal{P}_{3,2} = 10$. These basis functions can be represented in terms of barycentric coordinates. For example, the basis function that is one at the centroid with value zero and zero gradient at the vertices is given by $27\,\lambda_1\lambda_2\lambda_3$ in terms of barycentric coordinates.

#### 4.3.1.2 Reference Triangles

A way to prove properties about interpolation methods over triangles is to start with a *reference triangle* or reference element $\widehat{K}$, and then transfer properties from the reference triangle $K$ to a given triangle (or other shape) via an affine function or transformation: $\boldsymbol{T}_K(\widehat{\boldsymbol{x}}) = A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K$ with $A_K$ an invertible matrix. For example, we can take the reference element $\widehat{K}$ to be the triangle with vertices $\widehat{\boldsymbol{v}}_1 = (0,0)$, $\widehat{\boldsymbol{v}}_2 = (1,0)$, and $\widehat{\boldsymbol{v}}_3 = (0,1)$. For a triangle $K$ with vertices $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$, we can set $\boldsymbol{b}_K = \boldsymbol{v}_1$ and $A_K = [\boldsymbol{v}_2 - \boldsymbol{v}_1,\ \boldsymbol{v}_3 - \boldsymbol{v}_1]$. This is illustrated in Figure 4.3.3.

Any function $f\colon K \to \mathbb{R}$ can be represented by a function $\widehat{f}\colon \widehat{K} \to \mathbb{R}$ given by

$$\widehat{f}(\widehat{\boldsymbol{x}}) = f(A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K). \tag{4.3.6}$$

It is easy to see that $f$ is a polynomial of degree $k$ if and only if $\widehat{f}$ is a polynomial of degree $k$. Note that $\boldsymbol{x} = \lambda_1\boldsymbol{v}_1 + \lambda_2\boldsymbol{v}_2 + \lambda_3\boldsymbol{v}_3$ is a representation of $\boldsymbol{x} \in K$ by barycentric coordinates, and $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$, then the barycentric coordinates of $\widehat{\boldsymbol{x}}$ in $\widehat{K}$ are also $(\lambda_1, \lambda_2, \lambda_3)$. So if $f$ is the linear interpolant of data values at the vertices of $K$ ($f(\boldsymbol{v}_i) = y_i$), then $\widehat{f}$ is the linear interpolant of those same values at the vertices of $\widehat{K}$ ($\widehat{f}(\widehat{\boldsymbol{v}}_i) = y_i$). The Lagrange interpolation points are, in fact, constant in terms of the barycentric coordinates. The vertices have barycentric coordinates

$$(1,0,0),\ (0,1,0),\ (0,0,1).$$

The barycentric coordinates of the midpoints for quadratic Lagrange interpolation are

$$(\frac{1}{2},\frac{1}{2},0),\ (\frac{1}{2},0,\frac{1}{2}),\ (0,\frac{1}{2},\frac{1}{2}).$$

The barycentric coordinates for degree $d$ Lagrange interpolation are

$$\left(\frac{i}{d},\ \frac{j}{d},\ \frac{d-i-j}{d}\right), \qquad i,j=0,1,\ldots,d,\ i+j\le d.$$

The centroid of any triangle has barycentric coordinates $(\frac{1}{3},\frac{1}{3},\frac{1}{3})$.

The barycentric coordinates can be represented for the given reference triangle which can be explicitly represented in terms of $\widehat{\boldsymbol{x}}=(\widehat{x},\widehat{y})$:

$$\begin{aligned}\lambda_1&=1-\widehat{x}-\widehat{y},\\ \lambda_2&=\widehat{x},\\ \lambda_3&=\widehat{y}.\end{aligned}$$

While function values on the original and reference elements can be related through (4.3.6), gradients, and Hessian matrices are a little more complex:

$$A_K^T\nabla f(\boldsymbol{x})=\nabla\widehat{f}(\widehat{\boldsymbol{x}}), \tag{4.3.7}$$

$$A_K^T\,\mathrm{Hess}\,f(\boldsymbol{x})\,A_K=\mathrm{Hess}\,\widehat{f}(\widehat{\boldsymbol{x}}), \tag{4.3.8}$$

where $\boldsymbol{x}=A_K\widehat{\boldsymbol{x}}+\boldsymbol{b}_K=\boldsymbol{T}_K(\widehat{\boldsymbol{x}})$. Integrals are also transformed:

$$\int_K f(\boldsymbol{x})\,d\boldsymbol{x}=|\det A_K|\int_{\widehat{K}}\widehat{f}(\widehat{\boldsymbol{x}})\,d\widehat{\boldsymbol{x}}. \tag{4.3.9}$$

In computational practice, many quantities can be pre-computed on the reference triangle, and the corresponding quantity on the original triangle can be computed quickly.

Some important quantities, such as Lebesgue numbers for polynomial interpolation, are invariant under affine transformations (see Section 4.1.2). Interpolation schemes on a reference triangle $\widehat{K}$ are transformed to interpolation schemes on a given triangle $K$ by an affine transformation $\boldsymbol{T}_K$; if $\widehat{\mathcal{P}}$ is the family of interpolation functions on $\widehat{K}$ then the interpolation functions $\mathcal{P}$ on $K$ are given by $p=\widehat{p}\circ\boldsymbol{T}_K^{-1}$ for $\widehat{p}\in\widehat{\mathcal{P}}$. Since polynomials under affine transformations are still polynomials of the same degree, if $\widehat{\mathcal{P}}=\mathcal{P}_{k,d}$ then $\mathcal{P}=\mathcal{P}_{k,d}$ as well. The interpolation points $\widehat{\boldsymbol{x}}_i\in\widehat{K}$ are transformed to interpolation points $\boldsymbol{x}_i=\boldsymbol{T}_K(\widehat{\boldsymbol{x}}_i)\in K$.

**Table 4.3.1** Lebesgue numbers of Lagrange interpolation of degree $k$ in a triangle

| $k$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| $\Lambda_{k,2}$ | 1 | 1.67 | 2.26 | 3.47 | 5.23 | 8.48 | 14.33 |

**Theorem 4.11** *If $\widehat{\Lambda}$ is the Lebesgue number for interpolation on $\widehat{K}$ with interpolation points $\widehat{\boldsymbol{x}}_i$, $i = 1, 2, \ldots, N$, using interpolation functions from $\widehat{\mathcal{P}}$, and $\Lambda$ is the Lebesgue number of interpolation on $K$ with interpolation points $\boldsymbol{x}_i = \boldsymbol{T}_K(\widehat{\boldsymbol{x}}_i)$ using interpolation functions $\mathcal{P} = \widehat{\mathcal{P}} \circ \boldsymbol{T}_K^{-1}$, then $\widehat{\Lambda} = \Lambda$.*

***Proof*** The Lagrange interpolation functions on $\widehat{K}$ are given by $\widehat{L}_i \in \widehat{\mathcal{P}}$ where $\widehat{L}_i(\widehat{\boldsymbol{x}}_j) = 1$ if $i = j$ and zero if $i \neq j$. On the other hand $L_i = \widehat{L}_i \circ \boldsymbol{T}_K^{-1} \in \mathcal{P}$ and $L_i(\boldsymbol{x}_j) = L_i(\boldsymbol{T}_K(\widehat{\boldsymbol{x}}_j)) = \widehat{L}_i(\boldsymbol{T}_K^{-1}(\boldsymbol{T}_K(\widehat{\boldsymbol{x}}_j))) = \widehat{L}_i(\widehat{\boldsymbol{x}}_j) = 1$ if $i = j$ and zero if $i \neq j$. Thus $L_i = \widehat{L}_i \circ \boldsymbol{T}_K^{-1}$ are the Lagrange interpolation functions on $K$. Therefore,

$$\Lambda = \max_{\boldsymbol{x} \in K} \sum_{i=1}^{N} |L_i(\boldsymbol{x})| = \max_{\widehat{\boldsymbol{x}} \in \widehat{K}} \sum_{i=1}^{N} |L_i(\boldsymbol{T}_K(\widehat{\boldsymbol{x}}))| = \max_{\widehat{\boldsymbol{x}} \in \widehat{K}} \sum_{i=1}^{N} \left|\widehat{L}_i(\widehat{\boldsymbol{x}})\right| = \widehat{\Lambda},$$

as we wanted. □

Lebesgue numbers for Lagrange interpolation on a triangle for different degrees $k$ are given in [26] and shown in Table 4.3.1. Note that the Lebesgue number $\Lambda_{k,2}$ increases with $k$, but not so dramatically that seventh-order interpolation is not useful.

Just as in one dimension, equally spaced interpolation is not optimal when the degree is high. Bos [26] gives alternative interpolation points for triangles that have better Lebesgue numbers.

#### 4.3.1.3 ⚠Error Estimates

As in the one-dimensional case, the error estimates will depend on the size of the triangles. We can measure the size of each triangle or other shape by means of the *diameter*

$$h_K = \operatorname{diam}(K) = \max_{\boldsymbol{x}, \boldsymbol{y} \in K} \|\boldsymbol{x} - \boldsymbol{y}\|_2 . \tag{4.3.10}$$

In two and higher dimensions, however, this is not the only quantity that is important. Triangles can be long and thin, or relatively "chunky". For convex $K$, one way to measure this is to look at the radius of the largest ball that can fit in $K$:

$$\rho_K = \sup \{\, \rho \mid \text{there is a ball } B(\boldsymbol{x}, \rho) \subset K \text{ for some } \boldsymbol{x} \in K \,\} . \tag{4.3.11}$$

Brenner and Scott [31, p. 99] define a *chunkiness parameter* $\gamma_K := h_K/\rho_K \geq 1$. Note that smaller $\gamma_K$ means "chunkier" $K$. Thin but long triangles have $\rho_K$ small in comparison to $h_K$ making $\gamma_K$ large. Another measure of "chunkiness" is $\widetilde{\gamma}_K := h_K/\mathrm{vol}_d(K)^{1/d}$ for $K \subset \mathbb{R}^d$. Here "$\mathrm{vol}_d(R)$" is the $d$-dimensional volume of a region $R$; for $d = 2$ this is just the area of $R$, and for $d = 1$ it is just the total length of $R$.

For measuring the size and smoothness of functions on a region $\Omega \subset \mathbb{R}^d$ we use $W^{m,p}(\Omega)$, or Sobolev, norms and semi-norms; $m$ indicates the degree of differentiability, and $1 \leq p \leq \infty$ the exponent used: in terms of multi-indexes,

$$\|f\|_{W^{m,p}(\Omega)} = \left[\sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|\leq m} \int_\Omega \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}\right|^p d\boldsymbol{x}\right]^{1/p}, \tag{4.3.12}$$

$$|f|_{W^{m,p}(\Omega)} = \left[\sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|= m} \int_\Omega \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}\right|^p d\boldsymbol{x}\right]^{1/p}. \tag{4.3.13}$$

Note that

$$\|f\|_{W^{m,p}(\Omega)} = \left[\sum_{k=0}^{m} |f|^p_{W^{k,p}(\Omega)}\right]^{1/p}.$$

If $p = \infty$ we use

$$\|f\|_{W^{m,\infty}(\Omega)} = \max_{\boldsymbol{x}\in\Omega} \max_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|\leq m} \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x})\right|, \tag{4.3.14}$$

$$|f|_{W^{m,\infty}(\Omega)} = \max_{\boldsymbol{x}\in\Omega} \max_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|= m} \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x})\right|. \tag{4.3.15}$$

It should be noted that the $W^{m,p}(\Omega)$ semi-norm is equivalent to the semi-norm

$$f \mapsto \left[\int_\Omega \left\|D^m f(\boldsymbol{x})\right\|^p d\boldsymbol{x}\right]^{1/p}$$

in the sense of (1.5.2).

We want to bound the interpolation error $f - \mathcal{I}f$. This error is zero if $f$ is itself an interpolant, that is, $\mathcal{I}f = f$. Each interpolation scheme has a set of possible interpolants $\mathcal{P} = \mathrm{range}(\mathcal{I})$. Linear Lagrange interpolation is exact for all linear functions, quadratic Lagrange interpolation is exact for all quadratic functions, while Hermite interpolation is exact for all cubic functions. We suppose that this interpolation operator is exact for all polynomials of degree $\leq k$. As $k$ becomes larger, the interpolation operator is generally more accurate. As noted with respect to the Runge phenomenon (Section 4.1.1.13), it is more important to reduce spacing ($h_K$) than it is to increase the degree ($k$). We assume that $\mathcal{P}_k \subseteq \mathcal{P}$ where $\mathcal{P}_k$ is the set of polynomials of $\boldsymbol{x}$ of degree $\leq k$.

To obtain these estimates, we start with estimates of the interpolation error $\widehat{f} - \widehat{\mathcal{I}}\widehat{f}$ on the reference triangle $\widehat{K}$. By pulling the interpolation data from the original triangle $K$ back to the reference triangle $\widehat{K}$, doing the interpolation on $\widehat{K}$, and pushing the resulting interpolant back to $K$, we can get bounds for interpolation error $f - \mathcal{I}f$ on any given triangle $K$. We need to consider $\widehat{f}$ for which we can do the interpolation. To guarantee that we can take the derivatives up to order $\ell$ at an interpolation point, we need $\|\widehat{f}\|_{W^{m,p}(\widehat{K})}$ finite where $m - \ell > d/p$ (see [213, Sec. 6.4.6], for example), or $m \geq \ell$ if $p = \infty$.

Because $\widehat{\mathcal{I}}$ is exact on $\mathcal{P} \supseteq \mathcal{P}_k$, we can use the *Bramble–Hilbert lemma* [31, Thm. 4.3.8, p. 102] to give a bound

$$\left|\widehat{f} - \widehat{\mathcal{I}}\widehat{f}\right|_{W^{s,p}(\widehat{K})} \leq C(s, m, p, \widehat{\mathcal{I}}) \sum_{j=k+1}^{m} \left|\widehat{f}\right|_{W^{j,p}(\widehat{K})} \qquad \text{provided } m \geq \max(s,\ k+1).$$

Now we use the affine scaling property to transfer this result to a triangle (or tetrahedron or simplex) $K \subset \mathbb{R}^d$. Suppose that $\boldsymbol{T}_K : \widehat{K} \to K$ is the affine transformation $\boldsymbol{T}_K(\widehat{\boldsymbol{x}}) = A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K$. If $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$ then $\widehat{\boldsymbol{x}} = \boldsymbol{T}_K^{-1}(\boldsymbol{x}) = A_K^{-1}(\boldsymbol{x} - \boldsymbol{b}) = A_K^{-1}\boldsymbol{x} - A_K^{-1}\boldsymbol{b}$. Given a function $f : K \to \mathbb{R}$ we have a corresponding function $\widehat{f} : \widehat{K} \to \mathbb{R}$ where $\widehat{f}(\widehat{\boldsymbol{x}}) = f(\boldsymbol{x})$ for $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$. That is, $\widehat{f}(\widehat{\boldsymbol{x}}) = f(\boldsymbol{T}_K(\widehat{\boldsymbol{x}})) = f(A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K)$. Then the Sobolev semi-norms

$$|f|_{W^{j,p}(K)} = \left[\int_K \sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|=j} \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x})\right|^p d\boldsymbol{x}\right]^{1/p}.$$

Note that if $g(\boldsymbol{x}) = \widehat{g}(\widehat{\boldsymbol{x}})$, then

$$\begin{aligned}\frac{\partial g}{\partial x_j}(\boldsymbol{x}) &= \sum_\ell \frac{\partial}{\partial \widehat{x}_\ell} g(A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K)\,\frac{\partial \widehat{x}_\ell}{\partial x_j} \\ &= \sum_\ell (A_K^{-1})_{\ell j}\,\frac{\partial \widehat{g}}{\partial \widehat{x}_\ell}(\widehat{\boldsymbol{x}}),\end{aligned}$$

so there is a constant $C_1(|\boldsymbol{\alpha}|, j, d)$ where

$$\left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x})\right| \leq C_1(|\boldsymbol{\alpha}|, j, d)\ \left\|A_K^{-1}\right\|^{|\boldsymbol{\alpha}|} \max_{\boldsymbol{\beta}:|\boldsymbol{\beta}|=|\boldsymbol{\alpha}|} \left|\frac{\partial^{|\boldsymbol{\beta}|} \widehat{f}}{\partial \widehat{\boldsymbol{x}}^{\boldsymbol{\beta}}}(\widehat{\boldsymbol{x}})\right|.$$

This gives the bound on the Sobolev semi-norm for $j \leq k$,

$$|f|_{W^{j,p}(K)} \leq C_2(j, d, p)\left\|A_K^{-1}\right\|^j |\det A_K|^{1/p}\ \left|\widehat{f}\right|_{W^{j,p}(\widehat{K})}. \tag{4.3.16}$$

Applying the argument to $T_K^{-1} : K \to \widehat{K}$ gives

$$|\widehat{f}|_{W^{j,p}(\widehat{K})} \le C_2(j,d,p)\,\|A_K\|^j \left|\det A_K^{-1}\right|^{1/p} |f|_{W^{j,p}(K)}. \tag{4.3.17}$$

The interpolation estimate for $\widehat{K}$ from the Bramble–Hilbert lemma implies that for $r \le k$,

$$|\widehat{f} - \widehat{\mathcal{I}}\widehat{f}|_{W^{r,p}(\widehat{K})} \le C_3(r,m,p)\,|\widehat{f}|_{W^{k+1,p}(\widehat{K})}.$$

We can transform these bounds on $\widehat{K}$ to $K$. Note that the interpolation operator $\mathcal{I}_K$ on $K$ is given by $\mathcal{I}_K f(\boldsymbol{x}) = \widehat{\mathcal{I}}\widehat{f}(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = T_K(\widehat{\boldsymbol{x}})$. So

$$\begin{aligned}
|f - \mathcal{I}_K f|_{W^{r,p}(K)} &\le C_2(r,d,p)\left\|A_K^{-1}\right\|^r |\det A_K|^{1/p}\ |\widehat{f} - \widehat{\mathcal{I}}\widehat{f}|_{W^{r,p}(\widehat{K})} \\
&\le C_2(r,d,p)\left\|A_K^{-1}\right\|^r |\det A_K|^{1/p}\ C_3(r,m,p)\,|\widehat{f}|_{W^{k+1,p}(\widehat{K})} \\
&\le C_2(r,d,p)\left\|A_K^{-1}\right\|^r |\det A_K|^{1/p}\ C_3(r,m,p) \\
&\qquad \sum_{j=k+1}^{m} C_2(j,d,p)\,\|A_K\|^j \left|\det A_K^{-1}\right|^{1/p} |f|_{W^{j,p}(K)} \\
&\le C_4(r,k,m,p,d) \sum_{j=k+1}^{m} \left\|A_K^{-1}\right\|^r \|A_K\|^j\, |f|_{W^{j,p}(K)} \\
&\le C_4(r,k,m,p,d)\,\kappa(A_K)^r \sum_{j=k+1}^{m} \|A_K\|^{j-r}\, |f|_{W^{j,p}(K)}.
\end{aligned}$$

Here $\kappa(A) = \|A\|\left\|A^{-1}\right\|$ is the condition number of $A$.

Let $h_K$ be the diameter of $K$, which is the length of the longest edge of the triangle $K$. In two dimensions, $A_K = [\boldsymbol{v}_2 - \boldsymbol{v}_1,\ \boldsymbol{v}_3 - \boldsymbol{v}_1]$, where $\boldsymbol{v}_1,\ \boldsymbol{v}_2,\ \boldsymbol{v}_3$ are the vertices of $K$, so

$$\|A_K\|_2 \le \|A_K\|_F = \sqrt{\|\boldsymbol{v}_2 - \boldsymbol{v}_1\|_2^2 + \|\boldsymbol{v}_3 - \boldsymbol{v}_1\|_2^2} \le \sqrt{2}\,h_K.$$

For general $d$, $\|A_K\|_2 \le \sqrt{d}\,h_K$. However, the condition number $\kappa_2(A_K)$ can be arbitrarily large. To obtain a bound on the condition number, we need an additional condition that the triangle $K$ is "well shaped", at least according to the "chunkiness" parameters $\gamma_K$ or $\widetilde{\gamma}_K$ mentioned above. In two dimensions, "chunkiness" can be defined in terms of the minimum angle of the triangle $K$.

In higher dimensions, "chunkiness" is perhaps most commonly defined in terms of having an upper bound on the ratio $h_K/\rho_K$ where $\rho_K$ is the radius of the largest inscribed sphere of $K$.

However "chunkiness" or "well-shaped" is defined, it ensures a bound $\kappa(A_K) \le \kappa_{\max}$. Then we have a bound

$$|f - \mathcal{I}f|_{W^{r,p}(K)} \le C_5(r,k,m,p,d)\,\kappa_{\max}^r \sum_{j=k+1}^{m} h_K^{j-r}\, |f|_{W^{j,p}(K)}. \tag{4.3.18}$$

Clearly then, for smooth $f$, the interpolation error in the Sobolev norm of $W^{r,p}(K)$ is

$$\|f - \mathcal{I}f\|_{W^{r,p}(K)} = \mathcal{O}(h_K^{k+1-r}) \qquad \text{as } h_K \to 0$$

for well-shaped triangles. Note, however, that in the case of $r = 0$, the factor $\kappa(A_K)^r = 1$ no matter the value of $\kappa(A_K)$. So, if we are concerned only with approximating function *values*, then the requirement of the triangle to be well shaped is unnecessary. But if we wish the *derivatives* of the interpolant $\mathcal{I}_K f$ to approximate *derivatives* of $f$, then the condition of being well shaped becomes necessary. As we will see in Section 6.3.2.2 on the finite element method for partial differential equations, approximating derivatives is exactly what is needed.

### 4.3.2 Interpolation over Triangulations

Triangulations give a way of dividing a region up into triangles in two dimensions, or tetrahedra in three dimensions, or simplices in higher dimensions. If we can interpolate using polynomials over each triangle, then we can create a piecewise polynomial interpolant over the entire region. However, we want the interpolant on each triangle to be consistent with the interpolant on the neighboring triangles so that the combined interpolant is at least continuous over the triangulated region (Figure 4.3.4).

A triangulation is not simply a union of non-overlapping triangles. The triangles must meet each other in a specific way: if $T_1$ and $T_2$ are two triangles then $T_1 \cap T_2$ is either

- empty,
- a common vertex of $T_1$ and $T_2$, or
- a common edge of $T_1$ and $T_2$.

Note that the common edge must be an entire edge, not a partial edge, as shown in Figure 4.3.5.

Simply interpolating in each triangle does not guarantee that the interpolant is continuous on each common edge. We want the interpolant on each side of a common edge to be the same so that the overall interpolant is continuous.

Consider piecewise linear interpolation; if the two values on a common edge are identical, then the interpolants on each triangles sharing the edge will match on that edge. For a pair of triangles that meet at a vertex, the values of the interpolants on the different triangles must also match.

Since the values at interpolation points can be treated as independent quantities, these matching conditions imply that each vertex must be an interpolation point, and each edge must have two interpolation points. Piecewise linear interpolation on a triangulation then requires three interpolation points on each triangle, which must therefore be the vertices of each triangle.

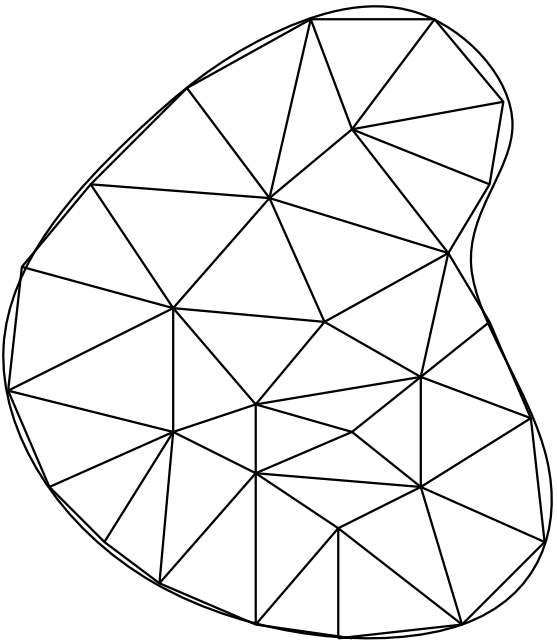


**Fig. 4.3.4** An example of a triangulation

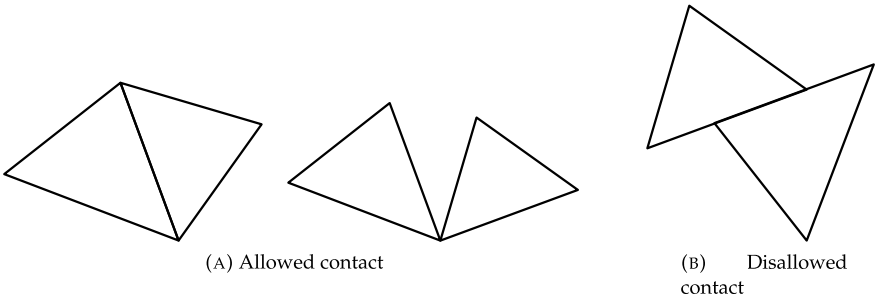
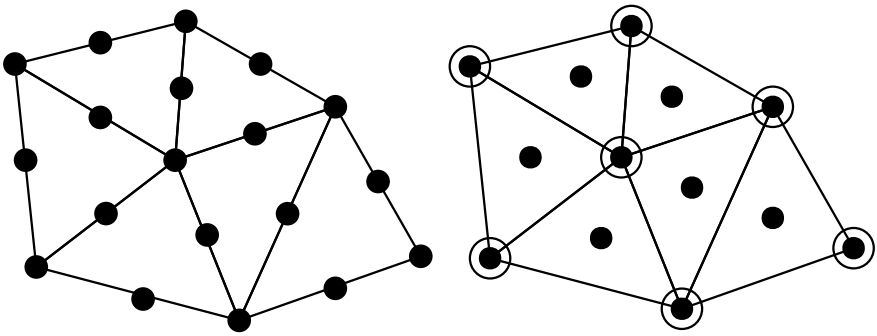



**Fig. 4.3.5** Allowed and disallowed contact between triangles in a triangulations

Guaranteeing continuity across triangles is substantially harder in two or more dimensions than one dimension, because in one dimension the "join" between two pieces is a point. As long as that point is an interpolation point, continuity is guaranteed across the pieces. However, two triangles typically meet at an edge. Two tetrahedra typically meet at a face. The values of the interpolants from each triangle must match, not just at the interpolation points, but also at every point of the common edge. For tetrahedra, the interpolants must match at every point of the common face.

To ensure that this works on triangles we require the following principles:

(1) vertices must be interpolation points;



(A) Quadratic interpolation on a triangulation

(B) Hermite interpolation on a triangulation

**Fig. 4.3.6** Interpolation on a triangulation

(2) the interpolation values on an edge uniquely determine the interpolant on that edge.

Note that the second item implies that value of the interpolant on a triangle restricted to an edge must *not* depend on the interpolation values at any point *not* on that edge. For tetrahedra and higher dimensional simplices, the condition can be expressed more succinctly: for an $m$-dimensional simplex, the interpolation values on each $k$-dimensional face must uniquely specify the interpolant on that face. This implies that the interpolant on a face cannot depend on any interpolation value for any interpolation point *not* on that face.

Piecewise quadratic interpolation can also be guaranteed to be continuous using the Lagrange nodal points shown in Figure 4.3.1 (middle figure). For triangles with a common vertex, we just need vertices to be interpolation points. For triangles with a common edge, the interpolants for each triangle must be quadratic on the edge, and so we need three interpolation points on each edge to ensure that these quadratics match. This requires six interpolation points: the three vertices plus one point on each edge. The edge midpoints are usually chosen. This means that the orientations of the triangles sharing an edge do not matter.

This gives continuous piecewise quadratic interpolation, but the first derivatives are not continuous. More specifically, while the tangential derivatives along the edges match because the values on the shared edge match, the normal derivatives generally do not.

Hermite interpolation across a triangulation is illustrated by Figure 4.3.6.

#### 4.3.2.1 Creating Interpolants with Continuous First Derivatives

Creating interpolation systems that guarantee continuous first derivatives across triangles in a triangulation is a surprisingly tricky thing to do. Unlike in one dimension, increasing the degree of the polynomial inside the triangle also increases the number

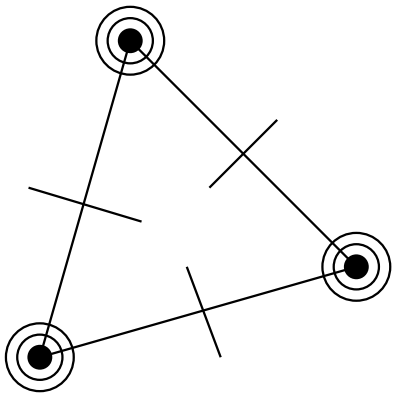


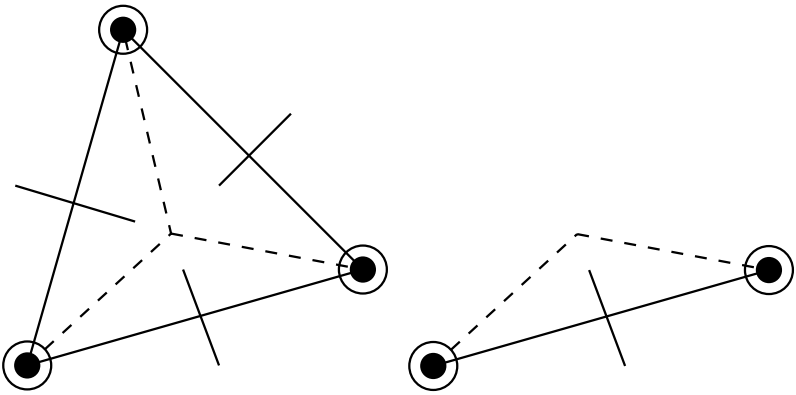
**Fig. 4.3.7** Argyris element (degree 5)

of conditions needed to match first derivatives across the boundary. The simplest is the *Argyris element* illustrated in Figure 4.3.7.

In Figure 4.3.7, the dot means interpolation of the value at that point, the small circle around a point means interpolating the first derivatives at that point, and the larger circle means interpolating the second derivatives at that point. The short perpendicular line segments indicate interpolation of the normal derivative to the edge at the intersection point of the edge and the line segment. The interpolating polynomials have degree 5. The dimension of the space of degree 5 polynomials of two variables is $\binom{5+2}{2} = 21$. This exactly matches the number of independent values to be interpolated. At each vertex we interpolate the function value, two first derivatives ($\partial f/\partial x$, $\partial f/\partial y$), and three second derivatives ($\partial^2 f/\partial x^2$, $\partial^2 f/\partial x\,\partial y$, $\partial^2 f/\partial y^2$) giving six values interpolated for each vertex. This gives 18 interpolated values for the vertices, plus three more the normal derivative values at the midpoints of each edge gives a total of 21 values to interpolate.

To see that Argyris element interpolants are continuous across edges, we note that on an edge the interpolating polynomial must have matching values, first and second derivatives, at the ends of the edge. The derivatives are, of course, scalar derivatives as along the edge we should consider tangential derivatives. This gives six values interpolated by a degree 5 polynomial in one variable. These six values are sufficient to uniquely specify the degree 5 polynomial on the edge. Since these six interpolated values are the same on both sides of the edge in question, the values of the interpolant must match across a common edge of two Argyris triangles.

But to have continuous first derivatives across the boundary, we also need the normal derivatives to match on each sides of the edge. Each edge is straight, so if $p(\boldsymbol{x})$ is degree 5 polynomial in two variables, on each edge $\partial p/\partial n(\boldsymbol{x}) = \boldsymbol{n}^T \nabla p(\boldsymbol{x})$ is a polynomial of degree $5 - 1 = 4$, since $\boldsymbol{n}$ is constant on each edge. The normal derivative $\partial p/\partial n$ is interpolated at each end of an edge, as is the first tangential derivative of $\partial p/\partial n$ at each end. Furthermore, since the Argyris element interpolates the normal derivative at the midpoint of each edge, we have five values to interpolate



(A) HCT element (piecewise cubic)

(B) HCT element piece

**Fig. 4.3.8** Hsieh–Clough–Tocher (HCT) element

on the edge for $\partial p/\partial n$. This means that $\partial p/\partial n$ is a uniquely specified polynomial of degree 4. Tangential derivatives of $p(\boldsymbol{x})$ along the edge are uniquely specified as the values of $p(\boldsymbol{x})$ on the edge are uniquely specified by the values and derivatives interpolated on that edge. Thus, the gradient $\nabla p(\boldsymbol{x})$ is uniquely specified on an edge by the values and derivatives interpolated on that edge.

Combining these arguments, we see that the degree 5 Argyris element gives a piecewise polynomial interpolant on a triangulation that has continuous first derivatives.

An alternative to Argyris-type elements are so-called *macro-elements*, such as the *Hsieh–Clough–Tocher* (HCT) element illustrated in Figure 4.3.8.

Macro-elements do not give a polynomial interpolant over the interval, but rather a piecewise polynomial interpolant where the pieces are sub-triangles of the original triangle. The pieces of the HCT triangle are indicated by the dashed lines in Figure 4.3.8. The advantage is the degree of the polynomial in each piece can be less than if we were imposing the same conditions on a single polynomial.

The HCT element is a piecewise cubic macro-element. The cubic polynomial on each piece has 10 coefficients, giving a total of 30 coefficients to specify the piecewise cubic interpolant. The cubic polynomial on each piece has to satisfy the values and first derivatives $(\partial p/\partial x, \partial p/\partial y)$ at *two* vertices of the triangle. This gives six conditions for each piece, giving a total of 18 conditions overall. Each external edge has the normal derivative of the interpolant specified at the midpoint. This gives a total of 21 conditions on the interpolating cubics. The additional nine conditions come from the continuity conditions on the internal edges. From continuity of the interpolant values and the first derivatives of the interpolant, we need matching values and first derivatives at the centroid. This means we have two sets of three equations to ensure these match. The internal edges go from the centroid to the vertices of the triangle; along each internal edge the values and tangential derivatives of the

interpolant match at the endpoints. Since the interpolant on each side of this internal edge is a cubic polynomial, we see that both of these interpolants match on the internal edge. This further implies that the tangential derivatives match on an internal edge. To complete the construction, we need to use the remaining three degrees of freedom to match the normal derivatives on the internal edges.

The normal derivative on an internal edge is a quadratic polynomial. These match at the endpoints of that internal edge as the gradients match at the endpoints of an internal edge. Since the normal derivative is a quadratic polynomial along an internal edge, we just need matching normal derivatives at one point at (say) the midpoint of the internal edge. This gives the extra three conditions needed to uniquely specify the cubic polynomials on each piece.

If we try matching the external edges of an HCT element across element boundaries, we note that the values and tangential derivatives match at the endpoints of the external edge. Since the cubic polynomial interpolants on each side of an external edge are cubic, they must match along the entire external edge. Because the first derivatives must match at the endpoints of an external edge, the normal derivatives must match as well. In addition, the normal derivatives must match at the midpoints of the external edge, by construction of the HCT element.

Thus, the HCT element can also be used to create piecewise cubic interpolants that are continuously differentiable over triangulations.

### 4.3.3 ⚠ Approximation Error over Triangulations

If we have a smooth function $f$ and a triangulation $\mathcal{T}_h$ we can interpolate $f$ over $\Omega_h = \bigcup_{K \in \mathcal{T}_h} K \subset \mathbb{R}^d$, the union of triangles of $\mathcal{T}_h$. We can define the interpolation operator $\mathcal{I}_h f(\boldsymbol{x}) = \widehat{\mathcal{I}} \widehat{f}(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = T_K(\widehat{\boldsymbol{x}})$ and $\boldsymbol{x} \in K$, for $K$ is a triangle in $\mathcal{T}_h$. There are many ways of defining $\widehat{\mathcal{I}}$ as we have seen: Lagrange interpolation, Hermite elements, Argyris and Hsieh–Clough–Tocher elements. Each of these interpolants $\widehat{\mathcal{I}} \widehat{f}$ is a piecewise polynomial of a certain degree. Provided the image of $\widehat{\mathcal{I}}$ contains all polynomials of degree $\leq k$ for some $k$, and the triangulation is "well shaped", the order of approximation is given by

$$|f - \mathcal{I}_h f|_{W^{r,p}(\Omega_h)} = \mathcal{O}(h^{k+1-r}) \ \|f\|_{W^{m,p}(\Omega_h)}$$

provided $m \geq \max(r,\ k+1)$ and $m > \ell + d/p$ (or $m \geq \ell$ if $p = \infty$) where $\mathcal{I}_h$ uses the values and derivatives of $f$ up to derivatives of order $\ell$.

The condition that $m > \ell + d/p$ is necessary to ensure that the function and derivative values up to order $\ell$ of $f$ can be bounded in terms of $\|f\|_{W^{m,p}(\Omega_h)}$.

If we have a function $f \in W^{k+1,p}(\Omega)$ then we cannot necessarily even define the interpolant $\mathcal{I}_h f$. However, we can approximate $f$ by an interpolant $\mathcal{I}_h g$ for some $g$. We can construct a suitable function $g$ as a convolution of an extension of $f$ to $\mathbb{R}^d$ with a smooth kernel function.

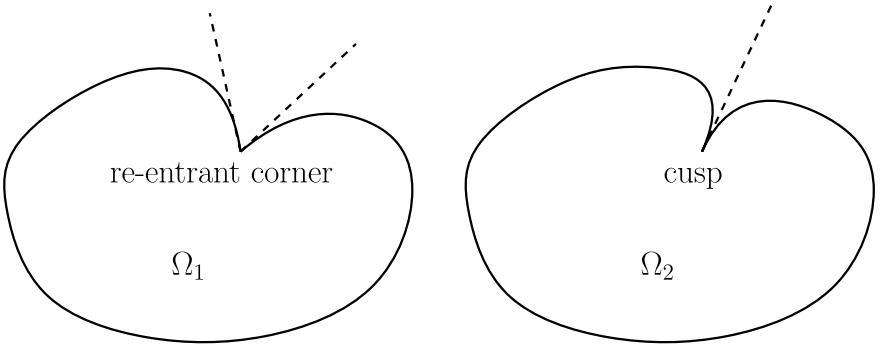



**Fig. 4.3.9** Corner vs. cusp

The extension of $f$ beyond $\Omega$ can be achieved through an extension operator $\mathcal{E}\colon W^{m,p}(\Omega) \to W^{m,p}(\mathbb{R}^d)$ provided $\Omega \subset \mathbb{R}^d$ has a boundary that is locally the graph of a Lipschitz continuous function [31, 237]. This condition on the boundary means that re-entrant corners are allowed, but not "cusps". The distinction is illustrated in Figure 4.3.9.

The extension operator $\mathcal{E}$ is a bounded operator in the sense that there is a constant $C$ where $\|\mathcal{E}f\|_{W^{m,p}(\mathbb{R}^d)} \le C\ \|f\|_{W^{m,p}(\Omega)}$ for all $f$. Note that the extension $\mathcal{E}f(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$, so this constant $C$ must be at least one. Since $\mathcal{E}f(\boldsymbol{x}) = f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \Omega$, $\mathcal{E}f$ cannot be smoother than $f$. To handle this we consider the convolution $\mathcal{E}f * \psi_s$ where $\psi_s$ is a smooth function (derivatives of all orders are continuous) that has compact support (that is, $\psi_s(\boldsymbol{z}) = 0$ for $\|\boldsymbol{z}\|_2 \ge R(s)$). We use the parameter $s > 0$ as a scaling parameter. More specifically,

$$\psi_s(\boldsymbol{z}) = s^{-d}\,\psi(\boldsymbol{z}/s).$$

The $R(s)$ above can then be given as $R(1)\,s$. We also assume that $\int_{\mathbb{R}^d} \psi(\boldsymbol{z})\,d\boldsymbol{z} = 1$. Then $\int_{\mathbb{R}^d} \psi_s(\boldsymbol{z})\,d\boldsymbol{z} = 1$ for any $s \neq 0$. The other condition that we need for the $\psi$ function is that it has zero moments of order up to $k$:

$$\int_{\mathbb{R}^d} \boldsymbol{z}^{\boldsymbol{\alpha}}\,\psi(\boldsymbol{z})\,d\boldsymbol{z} = 0 \qquad \text{for all } \boldsymbol{\alpha} \text{ where } 0 < |\boldsymbol{\alpha}| \le k. \tag{4.3.19}$$

This zero moments property ensures that the convolutions $\mathcal{E}f * \psi_s$ converge rapidly to $\mathcal{E}f$ as $s \downarrow 0$. This is straightforward to show for $p = 2$ via Fourier transforms: $W^{k+1,p}(\mathbb{R}^d) = W^{k+1,2}(\mathbb{R}^d)$. The moment conditions (4.3.19) and $\int_{\mathbb{R}^d} \psi(\boldsymbol{z})\,d\boldsymbol{z} = 1$ imply that

$$\mathcal{F}\psi(\boldsymbol{\zeta}) = 1 + \mathcal{O}(\|\boldsymbol{\zeta}\|_2^{k+1}) \qquad \text{as } \boldsymbol{\zeta} \to \mathbf{0}.$$

The reason is that

$$0 = \mathcal{F}\left[z \mapsto z^{\alpha}\,\psi(z)\right](\zeta) = i^{|\alpha|}\frac{\partial^{|\alpha|}}{\partial \zeta^{\alpha}}\mathcal{F}\psi(\zeta) \qquad \text{for } 0 < |\alpha| \le k.$$

As $\mathcal{F}\psi(\zeta)$ is analytic in $\zeta$, we can use multivariate Taylor series, and $\mathcal{F}\psi(\mathbf{0}) = \int_{\mathbb{R}^d} \psi(z)\,dz = 1$. Then provided $\mathcal{E}f \in W^{k+1,2}(\mathbb{R}^d) = H^{k+1}(\mathbb{R}^d)$ and $r \le k+1$,

$$\begin{aligned}
|\mathcal{E}f - \mathcal{E}f * \psi_s|^2_{W^{r,2}(\mathbb{R}^d)} &= (2\pi)^{-d}\int_{\mathbb{R}^d} \|\xi\|_2^{2r}\,|\mathcal{F}[\mathcal{E}f](\xi) - \mathcal{F}[\mathcal{E}f * \psi_s](\xi)|^2\,d\xi \\
&= (2\pi)^{-d}\int_{\mathbb{R}^d} \|\xi\|_2^{2r}\,|1 - \mathcal{F}\psi_s(\xi)|^2\,|\mathcal{F}[\mathcal{E}f](\xi)|^2\,d\xi \\
&\le \max_{\xi} \|\xi\|_2^{2r-2(k+1)}\,|1 - \mathcal{F}\psi_s(\xi)|^2 \times \\
&\qquad (2\pi)^{-d}\int_{\mathbb{R}^d} \|\xi\|_2^{2(k+1)}\,|\mathcal{F}[\mathcal{E}f](\xi)|^2\,d\xi \\
&= \max_{\xi} \|\xi\|_2^{2(r-k-1)}\,|1 - \mathcal{F}\psi(s\xi)|^2\;|\mathcal{E}f|^2_{W^{k+1,2}(\mathbb{R}^d)}\,.
\end{aligned}$$

Setting $\eta = s\xi$, we see that

$$\begin{aligned}
\max_{\xi} \|\xi\|_2^{2(r-k-1)}\,|1 - \mathcal{F}\psi(s\xi)|^2 &= \max_{\eta} \|\eta/s\|_2^{2(r-k-1)}\,|1 - \mathcal{F}\psi(\eta)|^2 \\
&= s^{2(k+1-r)} \max_{\eta} \|\eta\|_2^{2(r-k-1)}\,|1 - \mathcal{F}\psi(\eta)|^2\,.
\end{aligned}$$

We need $\mathcal{F}\psi(\eta) = 1 + \mathcal{O}(\|\eta\|_2^{k+1})$ to ensure that the maximum exists for $0 \le r \le k$. Summarizing,

$$|\mathcal{E}f - \mathcal{E}f * \psi_s|_{W^{r,2}(\mathbb{R}^d)} \le \mathcal{O}(s^{k+1-r})\;|\mathcal{E}f|_{W^{k+1,2}(\mathbb{R}^d)} = \mathcal{O}(s^{k+1-r})\;\|f\|_{W^{k+1,2}(\Omega)}\,,$$

as $s \downarrow 0$. On the other hand, if $r \ge k+1$ we have

$$|\mathcal{E}f * \psi_s|_{W^{r,2}(\mathbb{R}^d)} = \mathcal{O}(s^{k+1-r})\;\|f\|_{W^{k+1,2}(\Omega)}\,.$$

This bound can be extended to other $p$ from one to $\infty$ [237].

We can apply the interpolation operator $\mathcal{I}_h$ to $\mathcal{E}f * \psi_s$ as it belongs to $W^{m,p}(\mathbb{R}^d)$ and thus $W^{m,p}(\Omega)$ where $m > \ell + d/p$. This gives an approximation to $f$ over $\Omega$, and we can estimate the error in the approximation:

$$\begin{aligned}
|f - \mathcal{I}_h(\mathcal{E}f * \psi_s)|_{W^{r,p}(\Omega)} &\le |f - \mathcal{E}f * \psi_s|_{W^{r,p}(\Omega)} + |\mathcal{E}f * \psi_s - \mathcal{I}_h(\mathcal{E}f * \psi_s)|_{W^{r,p}(\Omega)} \\
&\le \mathcal{O}(s^{k+1-r})\;\|f\|_{W^{k+1,p}(\Omega)} + \mathcal{O}(1)\sum_{j=k+1}^{m} h^{j-r}\,|\mathcal{E}f * \psi_s|_{W^{j,p}(\Omega)} \\
&= \mathcal{O}(s^{k+1-r})\;\|f\|_{W^{k+1,p}(\Omega)} + \mathcal{O}(1)\sum_{j=k+1}^{m} h^{j-r}s^{k+1-j}\,\|f\|_{W^{k+1,p}(\Omega)}\,.
\end{aligned}$$

If we take $s = h$, or even just within a constant factor, we get

$$|f - \mathcal{I}_h(\mathcal{E}f * \psi_s)|_{W^{r,p}(\Omega)} = \mathcal{O}(h^{k+1-r}) \ \|f\|_{W^{k+1,p}(\Omega)} \,. \tag{4.3.20}$$

This estimate assumes that the triangulation is "well shaped" with $\max_{K\in\mathcal{T}_h} \kappa(A_K) \leq \kappa_{\max}$ independent of $h$. If $r = 0$, this "well-shaped" condition on the triangulation $\mathcal{T}_h$ can be dropped: we simply need to ensure that the maximum diameter of the triangles in $\mathcal{T}_h$ is small to obtain small errors. These approximation error estimates are asymptotically optimal; we cannot expect to obtain asymptotically better bounds given the smoothness of the function being approximated and the degree of the piecewise polynomial approximation.

### 4.3.4 Creating Triangulations

Creating triangulations is an important, but often neglected task, in obtaining piecewise polynomial approximations or solving partial differential equations. Rather than leave this issue unaddressed, we give a simple yet practical algorithm that gives "well-shaped" triangulations suitable for two- and three-dimensional problems by Persson and Strang [201]. This method involves some novel tools adapted from computational geometry, such as *Voronoi diagrams* and the associated *Delaunay triangulations* [70, Chaps. 7 & 9]. The Voronoi diagram and Delaunay triangulation for a set of $N$ points in $\mathbb{R}^2$ can be computed in $\mathcal{O}(N \log N)$ time, while for $N$ points in $\mathbb{R}^3$ these can be computed in $\mathcal{O}(N^2)$ time.

The Voronoi diagram of a set of points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\} \subset \mathbb{R}^d$ is the collection of regions

$$V_j = \left\{ \boldsymbol{x} \mid \left\|\boldsymbol{x} - \boldsymbol{x}_j\right\|_2 < \min_{\ell:\, \ell\neq j} \|\boldsymbol{x} - \boldsymbol{x}_\ell\|_2 \right\}. \tag{4.3.21}$$

That is, $V_j$ is the set of points in $\mathbb{R}^d$ closer to $\boldsymbol{x}_j$ than any other point in $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. Because we use the Euclidean or 2-norm for measuring distance, the boundaries are piecewise linear: if $\|\boldsymbol{x} - \boldsymbol{a}\|_2 = \|\boldsymbol{x} - \boldsymbol{b}\|_2$ then squaring and subtracting gives $2(\boldsymbol{b} - \boldsymbol{a})^T\boldsymbol{x} = \|\boldsymbol{b}\|_2^2 - \|\boldsymbol{a}\|_2^2 = (\boldsymbol{b} - \boldsymbol{a})^T(\boldsymbol{b} + \boldsymbol{a})$; it is the line perpendicular to $\boldsymbol{b} - \boldsymbol{a}$ that intersects the line segment joining $\boldsymbol{a}$ and $\boldsymbol{b}$ at its midpoint.

The Delaunay triangulation of $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is a graph where the vertices are the points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, and there are edges $\boldsymbol{x}_j \sim \boldsymbol{x}_\ell$ if $V_j$ and $V_\ell$ have a common edge (in two dimensions) or face (in three dimensions). In two dimensions, the Delaunay triangulation can be thought of as the "dual graph" to the Voronoi diagram (or perhaps, the boundaries of the $V_j$'s): the Voronoi regions $V_j$ becomes the vertices of the Delaunay triangulation while the edges of the Voronoi diagrams correspond to the edges of the Delaunay triangulation, except that the edges $\boldsymbol{x}_j \sim \boldsymbol{x}_\ell$ of the Delaunay triangulation are perpendicular to the common edge of $V_j$ and $V_\ell$.

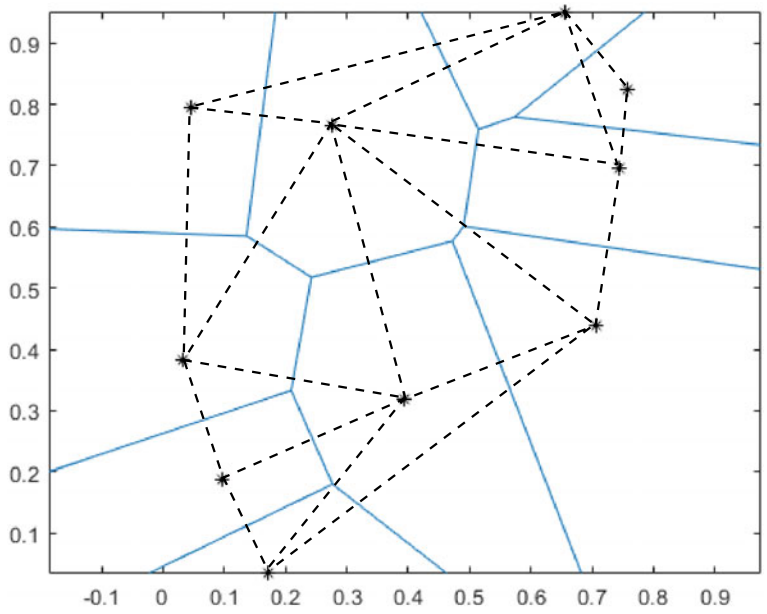



**Fig. 4.3.10** Voronoi diagram and Delaunay triangulation for a set of points; the dashed lines show the Delaunay triangulation

Efficient algorithms for computing Voronoi regions and Delaunay triangulations can be found in [70, Chap. 9]. An example of a Voronoi diagram and its associated Delaunay triangulation is shown in Figure 4.3.10.

Some of the common borders between the regions of a Voronoi diagram are outside the figure.

As can be seen from Figure 4.3.10, there is no guarantee that the triangles of a Delaunay triangulation are "well shaped".

The algorithm of Persson and Strang uses two functions: a signed "distance" function $D : \mathbb{R}^d \to \mathbb{R}$ and a relative spacing function $H : \mathbb{R}^d \to \mathbb{R}$. The region $\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^d \mid D(\boldsymbol{x}) < 0 \right\}$. The diameter of a triangle $K$ in the generated triangulation $\mathcal{T}$ should have diameter $h_K \leq h_0 \, H(\boldsymbol{x})$ for $\boldsymbol{x} \in K$. The value of $h_0$ represents the overall size of the triangles in the eventual triangulation.

The Persson and Strang algorithm begins by creating a uniformly spaced grid of points $\boldsymbol{x}_j$ over a bounding box $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d]$. The generated points $\boldsymbol{x}_j$ where $D(\boldsymbol{x}_j) > \epsilon_g$ for a specified $\epsilon_g > 0$ are removed. The Delaunay triangulation of the remaining points is then constructed. Any triangle that has a centroid outside $\Omega$ is removed.

Once a set of points roughly defining the region $\Omega$ have been created, they are adjusted to both better approximate the boundary of $\Omega$ and to make the mesh "well shaped". To make the triangulation "well shaped", each length of the edges of each triangle is moved to reduce the discrepancy. This is done by creating a "force" that

is applied to each point. This force is evaluated by first evaluating $H(\boldsymbol{x})$ for every $\boldsymbol{x}$ the midpoint of an edge of the triangulation.

To make this more formal, for any edge $e$ of the triangulation, $\ell_e$ is the length of edge $e$ ($\ell_e = \|\boldsymbol{x}_j - \boldsymbol{x}_k\|_2$ where $\boldsymbol{x}_j$ and $\boldsymbol{x}_k$ are the endpoints of edge $e$), and $\boldsymbol{x}_e = (\boldsymbol{x}_j + \boldsymbol{x}_k)/2$ be the midpoint of edge $e$. Let $h_e = H(\boldsymbol{x}_e)$. The target length for edge $e$ is $\widehat{\ell}_e = h_e F_{\text{scale}} \left[\sum_f \ell_f^2 / \sum_f h_f^2\right]^{1/2}$, the sum over $f$ ranging over all edges of the triangulation. The factor $F_{\text{scale}}$ is chosen to be about 1.2 for two-dimensional triangulations; the important issue is that $F_{\text{scale}} > 1$. The "force" applied to edge $e$ has magnitude $F_e = \max(\widehat{\ell}_e - \ell_e, 0)$, and in vector terms is $\boldsymbol{F}_e = F_e(\boldsymbol{x}_j - \boldsymbol{x}_k)/\ell_e$ so that the force acts along the direction of the edge. These forces on edges are turned into forces on vertices: the force on vertex $\boldsymbol{x}_j$ is then

$$\boldsymbol{F}_j = \sum_{e:\text{start}(e)=j} \boldsymbol{F}_e - \sum_{e:\text{end}(e)=j} \boldsymbol{F}_e.$$

The positions are updated $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_j + \Delta t\, \boldsymbol{F}_j$. This, of course, does not correspond to the true meaning of force which would involve updating momentum, but rather it is used to update positions. The value used for $\Delta t$ is 0.2.

Certain points $\boldsymbol{x}_j$ can be fixed, so that they are not updated using these "forces". It is particularly helpful to fix the corners of $\Omega$, for example.

To make the points give a better approximation to $\Omega$, points outside $\Omega$ are moved closer to the boundary. This is done essentially by using an under-determined Newton method $\boldsymbol{x} \leftarrow \boldsymbol{x} - D(\boldsymbol{x})\,\nabla D(\boldsymbol{x}) / \|\nabla D(\boldsymbol{x})\|_2^2$, using a finite difference approximation to $\nabla D(\boldsymbol{x})$.

Once the vertices $\boldsymbol{x}_j$ are updated, the Delaunay triangulation for the new set of points is computed, and the update process given above is repeated. This happens until the change between the previous and updated $\boldsymbol{x}_j$'s falls below a threshold.

The signed distance function $D(\boldsymbol{x})$ has to be created to represent $\Omega = \{\boldsymbol{x} \mid D(\boldsymbol{x}) < 0\}$. A circle of center $\boldsymbol{a}$ and radius $r$ can be represented by $D(\boldsymbol{x}) = (\boldsymbol{x} - \boldsymbol{a})^T(\boldsymbol{x} - \boldsymbol{a}) - r^2$. A rectangle $\Omega = (a, b) \times (c, d)$ can be represented by $D(x, y) = \max(|x - (a + b)/2| - (b - a)/2,\ |y - (c + d)/2| - (d - c)/2)$. We can also combine functions to represent combined regions: if $D_1$ represents $\Omega_1$ while $D_2$ represents $\Omega_2$. Then

$$\begin{aligned} D(\boldsymbol{x}) &= \max(D_1(\boldsymbol{x}),\ D_2(\boldsymbol{x})) \quad \text{represents } \Omega_1 \cap \Omega_2, \\ D(\boldsymbol{x}) &= \min(D_1(\boldsymbol{x}),\ D_2(\boldsymbol{x})) \quad \text{represents } \Omega_1 \cup \Omega_2, \\ D(\boldsymbol{x}) &= \max(D_1(\boldsymbol{x}),\ -D_2(\boldsymbol{x})) \quad \text{represents } \Omega_1 \backslash \Omega_2. \end{aligned}$$

While these functions are not smooth, they are effective for the Persson–Strang algorithm. Furthermore, the places where the function $D(\boldsymbol{x})$ is not smooth on the boundary are usually at a corner point of $\Omega$. Often these points are better represented as fixed points. Even for points on the boundary where $D(\boldsymbol{x})$ is not smooth, the

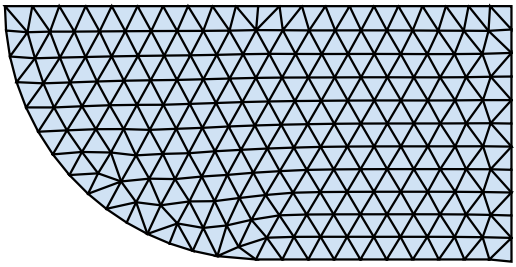


**Fig. 4.3.11** Triangulation from Persson–Strang algorithm

updates can still converge rapidly, although the theory supporting this argument is beyond the scope of this text.

To give an example of the results of the Persson–Strang triangulation algorithm, Figure 4.3.11 shows the results for the set

$$\Omega = \left\{ (x, y) \mid y \geq 0 \,\&\, (x^2 + y^2 \leq 1 \text{ or } 0 \leq x \leq 1) \right\} \quad \text{represented by}$$
$$D(x, y) = \max(\min(x^2 + y^2 - 1,\ y(y-1)),\ x(x+1)).$$

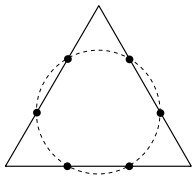
The Persson–Strang algorithm is far from perfect. It can go into infinite loops for several reasons. One is due to the fact that the triangulation is a discrete object computed from points, and so there are discontinuities where small changes in the point positions result in large discrete changes in the Delaunay triangulation. This introduces the possibility of oscillation between two triangulations: one triangulation results in a small change to the point positions, resulting in another triangulation. Then the new triangulation results in reversed point positions, resulting in the original triangulation. The combination of non-smooth but Lipschitz $D(\boldsymbol{x})$ functions combined with finite difference approximations to $\nabla D(\boldsymbol{x})$ can result in bad performance in the resulting algorithm. Also, three-dimensional triangulations generated can have elements that are not "well shaped".

These problems can be fixed using a more sophisticated, and more complex algorithm: locking the triangulation while point positions "settle down", and using exact values for $\nabla D(\boldsymbol{x})$ or a suitable generalization of the gradient can be used.

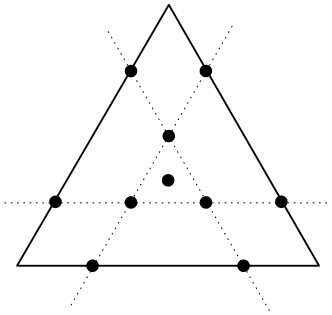
In spite of these issues, the Persson–Strang algorithm is a good starting point for investigating triangulation methods.

**Exercises.**

(1) The six points in this triangle are not sufficient to interpolate a quadratic function. Why?



(2) An interpolation method for cubic interpolation in $\mathbb{R}^2$ requires 10 function values, but this is not sufficient. Show that the points below in a triangle are sufficient for cubic interpolation.

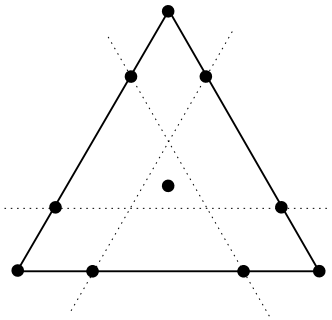


The center point is at the centroid. The dotted lines are at a distance of $1/4$ of the edge length from the nearest corner of the triangle.

(3) Give a Lagrange basis for cubic polynomials with the interpolation scheme of Exercise 2. That is, if $\boldsymbol{x}_j$, $j = 1, 2, \ldots, 10$ are the interpolation points, find cubic polynomials $\ell_j(\boldsymbol{x})$, $j = 1, 2, \ldots, 10$, where $\ell_j(\boldsymbol{x}_k) = 1$ if $j = k$ and zero otherwise. It may be simpler to first write the $\ell_j(\boldsymbol{x})$ in terms of barycentric coordinates $(\lambda_1, \lambda_2, \lambda_3)$ with $0 \leq \lambda_i$ for all $i$ and $\sum_{i=1}^{3} \lambda_i = 1$. Symmetry can be used to reduce the amount of work to do this.

(4) The interpolation scheme in Exercise 2 can be used to interpolate on a single triangle, but should not be used to generate a piecewise cubic interpolation on a triangulation. Explain why.

(5) Here is a modification of the interpolation in Exercise 2. Explain why this interpolation scheme can be used to generate a piecewise cubic interpolation on a triangulation.



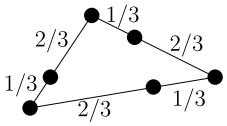
(6) Estimate the Lebesgue constant for the interpolation method in Exercise 2 over the triangle. Compare the number obtained against the Lebesgue constant for interpolation using the Lagrange points with barycentric coordinates $(j/3,\ k/3,\ \ell/3)$ where $j, k, \ell \geq 0$ and $j + k + \ell = 3$.

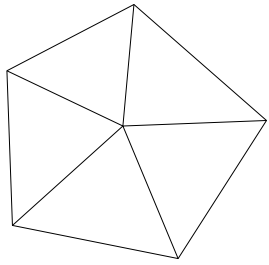
(7) Use the Persson–Strang triangulation method to obtain a triangulation of a unit disk $\Omega = \left\{ (x, y) \mid x^2 + y^2 \leq 1 \right\}$ with a target triangle diameter of 0.2, 0.1, and 0.05. Use these triangulations to obtain piecewise quadratic interpolants using the Lagrange points (see Figure 4.3.1) of $f(x, y) = e^{x+y} \sin(\pi x) - 1/(1 + x^2 + y^2)$. Plot the maximum error against the target triangle diameter.

(8) The Hermite interpolation scheme on triangles (see Figure 4.3.2) requires first derivative information. Give a basis for this interpolation scheme on a reference triangle $\widehat{K} = \{ (x, y) \mid 0 \leq x, y \,\&\, x + y \leq 1 \}$. For a general triangle $K$ with vertices $\boldsymbol{v}_1$, $\boldsymbol{v}_2$, $\boldsymbol{v}_3$ give formulas for the Hermite scheme on $K$ using the affine transformation $T_K(\widehat{\boldsymbol{x}}) = A_K \widehat{\boldsymbol{x}} + \boldsymbol{b}_K$. [**Note:** $p(\boldsymbol{x}) = \widehat{p}(A_K^{-1}(\boldsymbol{x} - \boldsymbol{b}_K))$ so $\nabla p(\boldsymbol{x}) = A_K^{-T} \nabla \widehat{p}(A_K^{-1}(\boldsymbol{x} - \boldsymbol{b}_K))$.]

(9) Give a basis for the Hsieh–Clough–Tocher or HCT element (Figure 4.3.8) on the reference triangle $\widehat{K} = \{ (x, y) \mid 0 \leq x, y \,\&\, x + y \leq 1 \}$. Recall that the HCT element has continuous first derivatives and is piecewise cubic. The central point is the centroid of the triangle. Use symmetry where possible to reduce the amount of work needed.

(10) Partly symmetric elements like the element below can cause trouble for triangulations. Show that it is not possible to use this interpolation scheme to obtain continuous interpolation in a pentagon of triangles as shown below.



Note that the barycentric coordinates of the interpolation points are $(1, 0, 0)$, $(0, 1, 0)$, $(0, 0, 1)$, $(\frac{2}{3}, \frac{1}{3}, 0)$, $(0, \frac{2}{3}, \frac{1}{3})$, and $(\frac{1}{3}, 0, \frac{2}{3})$.

## 4.4 Interpolation—Radial Basis Functions

*Radial basis functions* are functions of the form $\boldsymbol{x} \mapsto \varphi(\|\boldsymbol{x} - \boldsymbol{y}\|_2)$ for some $\boldsymbol{y}$. As we have seen, interpolation over two or three dimensions can be done using triangulations and interpolating over each triangle in a triangulation. However, trying to produce smooth interpolants is difficult: complex and high order methods are needed even to obtain continuous first order derivative interpolants in two dimensions. For dimensions higher than three, using triangulations with simplices becomes even more expensive: the $d$-dimensional hypercube $[0, 1]^d$ must be decomposed into at least $d!$ many $d$-dimensional simplices, with vertices at the vertices of the hypercube.

Also, the amount of data needed to perform such an interpolation becomes extremely large: to perform even a piecewise linear interpolant over the hypercube $[0,1]^d$ requires $2^d$ data points.

Radial basis functions provide a way of performing high-dimensional interpolation without requiring vast amounts of data or huge computational cost. These ideas are developed in several books, such as Buhmann [37], Fasshauer [88], and Wendland [258]. Suppose we have a set of $n$ data points $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, n$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \mathbb{R}$. We look for an interpolant of the data of the form

$$p(\boldsymbol{x}) = \sum_{j=1}^{n} c_j \, \varphi(\|\boldsymbol{x} - \boldsymbol{x}_j\|_2). \tag{4.4.1}$$

To find the values of the $c_i$'s, we need to solve the equations $p(\boldsymbol{x}_i) = y_i$ for $i = 1, 2, \ldots, n$. That is, we need to solve the equations

$$y_i = \sum_{j=1}^{n} c_j \, \varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) \qquad \text{for } i = 1, 2, \ldots, n. \tag{4.4.2}$$

This is clearly a square linear system of equations, and so can be solved, provided the matrix $A := \left[\varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) \mid i, j = 1, 2, \ldots, n\right]$ is invertible.

Some functions $\varphi\colon [0,\infty) \to \mathbb{R}$ are not suitable; constant functions are clearly a bad choice. Scaling can be an important issue here: if $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \ll 1$ for all $i$ and $j$, then $\varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2) \approx \varphi(0)$. While the function $\varphi$ is not constant, it appears to be nearly constant, and so numerical troubles can be expected in this case.

The placement of the interpolation points is also important. If $\|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \ll 1$ for *some* $i$ and $j$, then

$$\|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2 - \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2 \le \|\boldsymbol{x}_j - \boldsymbol{x}_k\|_2 \le \|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2 + \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2$$

and so $\varphi(\|\boldsymbol{x}_i - \boldsymbol{x}_k\|_2) \approx \varphi(\|\boldsymbol{x}_j - \boldsymbol{x}_k\|_2)$ for all $k$. Thus, columns $i$ and $j$ of $A$ will be nearly linearly dependent, making $A$ ill-conditioned.

Bearing these things in mind, what can we say about the choice of $\varphi$ that will make it possible to ensure that the matrix $A$ above is invertible? Can we find some bounds on the size of $A^{-1}$? An example where the theory is fairly straightforward is $\varphi(r) = \exp(-\alpha r^2)$ for a constant $\alpha > 0$. As discussed in Buhmann [37] this is in some ways the starting point for all other radial basis function methods.

One way to show that $A$ is invertible provided the interpolation points $\boldsymbol{x}_j$, $j = 1, 2, \ldots, n$, are distinct is to show that the matrix $A$ is positive definite. Clearly $A$ is symmetric as $a_{j\ell} = \varphi(\|\boldsymbol{x}_j - \boldsymbol{x}_\ell\|_2) = \varphi(\|\boldsymbol{x}_\ell - \boldsymbol{x}_j\|_2) = a_{\ell j}$. Let $\psi(\boldsymbol{z}) = \varphi(\|\boldsymbol{z}\|_2)$, so $p(\boldsymbol{x}) = \sum_{j=1}^{n} c_j \, \psi(\boldsymbol{x} - \boldsymbol{x}_j)$. Note that $p$ can be written as a convolution

$$p(\boldsymbol{x}) = \left[\psi * \sum_{j=1}^{n} c_j\, \delta(\cdot - \boldsymbol{x}_j)\right](\boldsymbol{x}) = \int_{\mathbb{R}^d} \psi(\boldsymbol{x} - \boldsymbol{z}) \sum_{j=1}^{n} c_j\, \delta(\boldsymbol{z} - \boldsymbol{x}_j)\, d\boldsymbol{z}$$
$$= \sum_{j=1}^{n} c_j\, \psi(\boldsymbol{x} - \boldsymbol{x}_j),$$

where $\delta$ is the Dirac-$\delta$ function. Using the convolution property of Fourier transforms (A.2.8),

$$\mathcal{F}p(\boldsymbol{\xi}) = \mathcal{F}\psi(\boldsymbol{\xi})\, \mathcal{F}\left[\sum_{j=1}^{n} c_j\, \delta(\cdot - \boldsymbol{x}_j)\right]$$
$$= \mathcal{F}\psi(\boldsymbol{\xi}) \sum_{j=1}^{n} c_j\, e^{-i\boldsymbol{\xi}^T \boldsymbol{x}_j}.$$

To show that the matrix $A$ is positive definite, we need to look at

$$\boldsymbol{c}^T A\boldsymbol{c} = \sum_{j,\ell=1}^{n} c_j a_{j\ell} c_\ell = \sum_{j,\ell=1}^{n} c_j \psi(\boldsymbol{x}_j - \boldsymbol{x}_\ell) c_\ell$$
$$= \sum_{j=1}^{n} c_j\, p(\boldsymbol{x}_j) = \left(\sum_{j=1}^{n} c_j\, \delta(\cdot - \boldsymbol{x}_j),\ p\right) = (q,\ p),$$

where $q = \sum_{j=1}^{n} c_j\, \delta(\cdot - \boldsymbol{x}_j)$ and $(f, g) = \int_{\mathbb{R}^d} \overline{f(\boldsymbol{z})}\, g(\boldsymbol{z})\, d\boldsymbol{z}$ is the usual inner product for functions. Note that $\mathcal{F}q(\boldsymbol{\xi}) = \sum_{j=1}^{n} c_j\, e^{-i\boldsymbol{\xi}^T \boldsymbol{x}_j}$. The inner product property of the Fourier transform (A.2.6) is that $(f,\ g) = (2\pi)^{-d}\, (\mathcal{F}f,\ \mathcal{F}g)$, so

$$\boldsymbol{c}^T A\boldsymbol{c} = (2\pi)^{-d}\ (\mathcal{F}q,\ \mathcal{F}p)$$
$$= (2\pi)^{-d} \int_{\mathbb{R}^d} \overline{\mathcal{F}q(\boldsymbol{\xi})}\, \mathcal{F}\psi(\boldsymbol{\xi})\, \mathcal{F}q(\boldsymbol{\xi})\, d\boldsymbol{\xi}$$
$$= (2\pi)^{-d} \int_{\mathbb{R}^d} \mathcal{F}\psi(\boldsymbol{\xi})\ |\mathcal{F}q(\boldsymbol{\xi})|^2\ d\boldsymbol{\xi}.$$

Now

$$\mathcal{F}\psi(\boldsymbol{\xi}) = \left(\frac{\pi}{\alpha}\right)^{d/2} \exp(-\|\boldsymbol{\xi}\|_2^2 / (4\alpha)) > 0$$

for all $\boldsymbol{\xi}$. Thus if $\boldsymbol{c} \neq \boldsymbol{0}$, $\boldsymbol{c}^T A\boldsymbol{c} > 0$ and so $A$ is positive definite, and therefore invertible.

We can estimate the Lebesgue numbers for this interpolation method: the corresponding Lagrange functions are $L_k(\boldsymbol{x}) = \sum_{j=1}^{n} c_j^{(k)}\, \psi(\boldsymbol{x} - \boldsymbol{x}_j)$ where $\boldsymbol{c}^{(k)}$ solves

the linear system $A\boldsymbol{c}^{(k)} = \boldsymbol{e}_k$ where $\boldsymbol{e}_k$ is the $k$th unit basis vector. So the Lebesgue number $\Lambda$ can be bounded as

$$\Lambda = \max_{\boldsymbol{x}} \sum_{k=1}^{n} |L_k(\boldsymbol{x})| \le \max_{\boldsymbol{x}} \sum_{k=1}^{n} \sum_{j=1}^{n} \left| c_j^{(k)} \right| \, |\psi(\boldsymbol{x} - \boldsymbol{x}_j)| \le \sum_{k=1}^{n} \left\| \boldsymbol{c}^{(k)} \right\|_1 ,$$

since $|\psi(\boldsymbol{z})| \le 1$ for all $\boldsymbol{z}$. Note that $\boldsymbol{c}^{(k)}$ is the $k$th column of $A^{-1}$ and so $\Lambda \le \sum_{j,\ell=1}^{n} \left| (A^{-1})_{j\ell} \right|$.

Lebesgue numbers are not the end of the story about how well this works to approximate functions. To start with, if $\alpha \to \infty$ in $\varphi(r) = \exp(-\alpha r^2)$, then the matrix $A$ goes to the $n \times n$ identity matrix. This gives $\Lambda \approx 1$ which seems ideal, but as shown in Theorem 4.7, we also need good approximation properties for our family of interpolants to get small interpolation error.

However, large $\alpha > 0$ does not give good interpolants or approximations as $\varphi(\left\| \boldsymbol{x} - \boldsymbol{x}_j \right\|_2) = \exp(-\alpha \left\| \boldsymbol{x} - \boldsymbol{x}_j \right\|_2^2)$ drops off rapidly as $\left\| \boldsymbol{x} - \boldsymbol{x}_j \right\|_2$ increases for large $\alpha$. We need to reduce $\alpha$ so that the influence of an interpolation point $\boldsymbol{x}_j$ extends significantly beyond the closest distinct interpolation point. On the other hand, taking $\alpha \downarrow 0$ gives $\left\| A^{-1} \right\| \to \infty$. Finding the optimal value of $\alpha$ depends on the spacing of the points $\boldsymbol{x}_i$ and their placement in $\mathbb{R}^d$.

**Exercises.**

(1) Show that for distinct points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n \in \mathbb{R}^d$ the interpolation matrix $A(\alpha)$ with entries $a_{ij}(\alpha) = \exp(-\alpha \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|_2^2)$ goes to the identity matrix as $\alpha \to +\infty$, but goes to $\boldsymbol{e}\boldsymbol{e}^T$ as $\alpha \downarrow 0$ where $\boldsymbol{e} \in \mathbb{R}^n$ is the vector of ones.

(2) Expand the previous exercise by using the Taylor series $a_{ij}(\alpha)$ for small $\alpha > 0$: $a_{ij}(\alpha) = 1 - \alpha \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|_2^2 + \mathcal{O}(\alpha^2)$. Show that $\left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|_2^2 = (\boldsymbol{b}\boldsymbol{e}^T + \boldsymbol{e}\boldsymbol{b}^T - 2X^T X)_{ij}$ where $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N]$, $\boldsymbol{e} = [1, 1, \ldots, 1]^T$, and $\boldsymbol{b} = [\boldsymbol{x}_1^T \boldsymbol{x}_1, \boldsymbol{x}_2^T \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N^T \boldsymbol{x}_N]^T$. Give a first-order approximation to $A(\alpha)$ for $\alpha \approx 0$. Give an asymptotic approximation for $A(\alpha)^{-1}$ in terms of $X$ and $\boldsymbol{b}$.

(3) Use the radial basis function $\varphi(r) = \exp(-\alpha r^2)$ to interpolate $f(x, y) = e^{x+y} \sin(\pi x) - 1/(1 + x^2 + y^2)$ over the unit circle $\Omega = \left\{ (x, y) \mid x^2 + y^2 \le 1 \right\}$. Use grid points $\boldsymbol{x} = (i, j)\, h$ for $i, j \in \mathbb{Z}$ in the unit circle with $h = 0.1$ for the interpolant. Plot the maximum error of the radial basis interpolant against $\alpha$. Also plot the norm $\left\| A^{-1} \right\|_\infty$ against $\alpha$ where $a_{ij} = \exp(-\alpha \left\| \boldsymbol{x}_i - \boldsymbol{x}_j \right\|_2^2)$. What is the optimal $\alpha$ for the maximum error of the interpolant?

(4) Repeat the previous exercise with interpolation points sampled from a uniform distribution t circle: set $\boldsymbol{x}_k = (x_k, y_k)$ with both $x_k$ and $y_k$ sampled from the uniform distribution over $[-1, +1]$, rejecting any $\boldsymbol{x}_k$ outside the unit circle. Use 314 interpolation points.

(5) Consider finding the least squares approximation $\sum_{k \in \mathbb{Z}} c_k \varphi(x - k) \approx 1$ for all $x$. Assuming the solution is integer-translation invariant, set $c_k = c_0$ for all $k \in \mathbb{Z}$. The problem then becomes: minimize $\int_{-1/2}^{+1/2} (1 - c_0 \sum_{k \in \mathbb{Z}} \varphi(x - k))^2 dx$. Show that the value of $c_0$ that minimizes this integral is

$$c_0 = \frac{\int_{-1/2}^{+1/2} \sum_{k\in\mathbb{Z}} \varphi(x-k)\,dx}{\int_{-1/2}^{+1/2} \left(\sum_{k\in\mathbb{Z}} \varphi(x-k)\right)^2 dx}.$$

Compute this integral numerically for $\varphi(z) = \exp(-\alpha z^2)$ for $\alpha = 2^m$, $m = -5, -4, \ldots, +4, +5$. For each value of $\alpha$ listed, compute numerically $\int_{-1/2}^{+1/2} (1 - c_0 \sum_{k\in\mathbb{Z}} \varphi(x-k))^2 dx$. What value(s) of $\alpha$ minimize the least squares error for approximating a constant function?

(6) Show that if

$$B(x) = \begin{cases} 1 + 3\,|x|^3/4 - 3x^2/2, & 0 \le |x| \le 1, \\ (2 - |x|)^3/4, & 1 \le |x| \le 2, \\ 0, & 2 \le |x|\,, \end{cases}$$

then $\mathcal{F}B(\xi) = (3/2)\sin^4(\xi/2)/(\xi/2)^4$ and so $\mathcal{F}B(\xi) > 0$ for all $\xi \in \mathbb{R}$. Show that $\varphi(x) = B(\alpha x)$ for $\alpha > 0$ can be used for radial basis function for interpolation over $\mathbb{R}$. In higher dimensions, show that $p(\boldsymbol{x}) := \sum_{k=1}^{N} c_k \varphi(\boldsymbol{x} - \boldsymbol{x}_k)$ can be used to interpolate arbitrary values $y_k = p(\boldsymbol{x}_k)$ provided all the interpolation points $\boldsymbol{x}_k \in \mathbb{R}^d$ are distinct if $\varphi(\boldsymbol{z}) = \prod_{j=1}^{d} B(\alpha z_j)$. [**Hint:** Show that $\mathcal{F}\varphi(\boldsymbol{\xi}) > 0$ for all $\boldsymbol{\xi} \in \mathbb{R}^d$.] Note that $B$ is a cubic $B$-spline function.

## 4.5 Approximating Functions by Polynomials

How well can we approximate functions by polynomials? It depends, to some extent, by how we measure the size of the difference between two functions $f, p\colon D \to \mathbb{R}$. The most severe measure commonly used is the maximum error:

$$\|f - p\|_\infty = \max_{\boldsymbol{x}\in D} |f(\boldsymbol{x}) - p(\boldsymbol{x})|\,.$$

Other measures include the least squares measure

$$\|f - p\|_2 = \left[\int_D (f(\boldsymbol{x}) - p(\boldsymbol{x}))^2\,d\boldsymbol{x}\right]^{1/2}.$$

How well can we approximate $f$ by a polynomial of degree $\le m$? Can we make the error arbitrarily small (by taking $m \to \infty$)?

### 4.5.1 Weierstrass' Theorem

How well can continuous functions be approximated by polynomials?

Weierstrass discovered the answer: as well as we please.

**Theorem 4.12** *(Weierstrass approximation theorem) For any continuous function* $f : [a, b] \to \mathbb{R}$ *and* $\epsilon > 0$ *there is a polynomial* $p$ *where*

$$\max_{a \le x \le b} |f(x) - p(x)| \le \epsilon.$$

There are many different proofs of this result. One is given in Ralston and Rabinowitz [211] using Bernstein polynomials on [0, 1]. Note that if we prove the result on [0, 1], then we can use the fact that if $\widetilde{p}(t)$ is a polynomial approximation to $t \mapsto f(a + (b - a)t)$ on [0, 1], then $p(x) = \widetilde{p}((x - a)/(b - a))$ approximates $f$ by the same amount on $[a, b]$:

$$\begin{aligned} \max_{a \le x \le b} |f(x) - p(x)| &= \max_{0 \le t \le 1} \left| f(a + t(b - a)) - \widetilde{p}\left(\frac{\{a + t(b - a)\} - a}{b - a}\right) \right| \\ &= \max_{0 \le t \le 1} |f(a + t(b - a)) - \widetilde{p}(t)| . \end{aligned}$$

The proof in [211] uses *Bernstein polynomials* given by

$$\phi_{k,n}(t) = \binom{n}{k} t^k (1 - t)^{n-k}$$

and the *quasi-interpolant*

$$p_n(t) = \sum_{k=0}^{n} f\left(\frac{k}{n}\right) \phi_{k,n}(t).$$

The accuracy of the approximation depends on how smooth or rough the function $f$ is. One measure of roughness is the *modulus of continuity* of a function $f : [a, b] \to \mathbb{R}$ given by

$$\omega(\delta) = \sup \{ |f(x) - f(y)| : x, y \in [a, b] \text{ and } |x - y| \le \delta \} . \tag{4.5.1}$$

Note that the proof in [211] gives a bound on $\|f - p_n\|_\infty = \mathcal{O}(\omega(n^{-1/2}))$ as $n \to \infty$. This is, in fact, rather pessimistic. If $f$ is twice differentiable, then the error in the Bernstein quasi-interpolant is, in fact, $\|f - p_n\|_\infty = \mathcal{O}(n^{-1})$, compared to the estimate obtained above, which is $\|f - p_n\|_\infty = \mathcal{O}(n^{-1/2})$ as $n \to \infty$.

If we allow ourselves to use other polynomials than Bernstein quasi-interpolants, we can obtain much better approximation estimates—Jackson's theorem (Theorem 4.14) shows how to do this very well. But Weierstrass' result is a valuable start in one dimension.

In more than one dimension, there is a very powerful generalization that applies not just to polynomial approximation, but also trigonometric polynomial and other kinds of approximation methods. That theorem is the *Stone–Weierstrass theorem*, and it is taught more often in topology than numerical analysis. Note that an *algebra* $\mathcal{A}$ is understood to be a vector space over $\mathbb{R}$ that has a multiplication operation that is consistent with the vector space structure.

**Theorem 4.13** *(Stone–Weierstrass theorem) Suppose that $\mathcal{A}$ is an algebra of continuous functions $D \to \mathbb{R}$ with $D$ a compact set that contains a non-zero constant function and separates points (that is, for any $\boldsymbol{x} \neq \boldsymbol{y} \in D$ there is a function $s$ in $\mathcal{A}$ where $s(\boldsymbol{x}) \neq s(\boldsymbol{y})$). Then for any continuous function $f : D \to \mathbb{R}$ and $\epsilon > 0$, there is a $p \in \mathcal{A}$ where $\|f - p\|_\infty \leq \epsilon$.*

Since $\mathcal{A}$ is a subalgebra of all continuous functions $D \to \mathbb{R}$, this amounts to requiring that if $f, g \in \mathcal{A}$, then $f \cdot g \in \mathcal{A}$. Examples of such algebras for $D \subset \mathbb{R}^n$ include polynomials, trigonometric polynomials ($x \mapsto \cos(kx)$ or $x \mapsto \sin(kx)$ for $k$ an integer), and linear combinations of exponentials $x \mapsto e^{\alpha x}$ with $\alpha \in \mathbb{R}$. It can include algebras that are not natural, such as polynomials on $[0, 1]$ with only even powers.

Unlike our proof of Weierstrass' theorem, most proofs of the Stone–Weierstrass theorem are heavily topological and give essentially no information about how rapidly the approximations approach a given function in terms of the "order" or "degree" of the approximating function. Nevertheless, the Stone–Weierstrass theorem is a powerful theorem that can justify different approximation schemes.

For a proof of the Stone–Weierstrass theorem, see [242] or [184].

### 4.5.2 Jackson's Theorem

In 1911 Dunham Jackson, an American mathematician writing a PhD thesis at Göttingen, Germany, proved theorems connecting the smoothness of a function, and the accuracy of trigonometric polynomial approximations and ordinary polynomial approximations. These results and more were published in his book [133]. Sergei Bernstein proved a converse to Jackson's main theorem in 1912, showing that the asymptotics of the error of the best polynomial approximation as the degree goes to infinity imply a certain degree of smoothness of the function.

**Theorem 4.14** *(Jackson's theorem) If $f : [a, b] \to \mathbb{R}$ is $r$ times differentiable, then there is a constant $C$ where for each positive integer $n$ there is a trigonometric polynomial $p_n$ of degree $n$ such that*

$$\max_{a \leq x \leq b} |f(x) - p_n(x)| \leq \frac{C\, \omega(f^{(r)}, 1/n)}{n^r}, \tag{4.5.2}$$

*where $\omega$ is the modulus of continuity.*

Jackson proved this result via Fourier series: start with

$$f(\frac{a+b}{2} + \frac{b-a}{2}\cos\theta) = \sum_{k=0}^{\infty} a_k \cos(k\theta).$$

Note that $\cos(k\theta)$ is a polynomial in $\cos\theta$; in fact, $\cos(k\theta) = T_k(\cos\theta)$ where $T_k$ is the *Chebyshev polynomial* of degree $k$ (see (4.6.3)). We can approximate the infinite sum by a finite sum. But simply truncating the sum does not necessarily give good approximations for smooth $f$. Instead, we first pick a *cutoff function* $\varphi\colon [0,\infty) \to \mathbb{R}$ where $\varphi(u) = 1$ for $u \le \frac{1}{2}$, $\varphi(u) = 0$ for $u \ge 1$, and is smooth and non-negative. Then we can set

$$p_n(\frac{a+b}{2} + \frac{b-a}{2}\cos\theta) = \sum_{k=0}^{n} \varphi(\frac{k}{n+1})\, a_k \cos(k\theta)$$

$$= \int_0^{2\pi} \psi_n(\theta-\theta')\, f(\frac{a+b}{2} + \frac{b-a}{2}\cos\theta')\, d\theta' \qquad \text{where} \tag{4.5.3}$$

$$\psi_n(\theta) = \frac{1}{2\pi}\varphi(0) + \frac{1}{\pi}\sum_{k=1}^{n} \varphi(\frac{k}{n+1}) \cos(k\theta). \tag{4.5.4}$$

Because $\int_0^{2\pi} \psi_n(\theta)\, d\theta = 1$ and yet $\psi_n(\theta) \to 0$ rapidly as $n \to \infty$ for $\theta$ not a multiple of $2\pi$, the polynomials $p_n$ converge to $f$ at a rapid rate that is determined by the smoothness of $f$.

### 4.5.3 Approximating Functions on Rectangles and Cubes

Creating approximations to functions on rectangles $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d] \subset \mathbb{R}^d$ can be done using polynomials based on one-dimensional methods. To simplify the discussion, we will scale and shift each interval $[a_j, b_j]$ to the interval $[-1, +1]$ so that we focus on the hypercube $[-1, +1]^d$. We can use *tensor product interpolation* to create polynomial interpolants: let $-1 \le \widehat{x}_0 < \widehat{x}_1 < \cdots < \widehat{x}_n \le +1$ be given one-dimensional interpolation points. Then for a function $f\colon [-1, +1]^d \to \mathbb{R}$ we interpolate the values $y_{i_1,i_2,\ldots,i_d} = f(\widehat{x}_{i_1}, \widehat{x}_{i_2}, \ldots, \widehat{x}_{i_d})$. For notational convenience we let $\boldsymbol{i} = (i_1, i_2, \ldots, i_d)$ and $\widehat{\boldsymbol{x}}_{\boldsymbol{i}} = [\widehat{x}_{i_1}, \widehat{x}_{i_2}, \ldots, \widehat{x}_{i_d}]^T$. We can use Lagrange interpolation polynomials to define the interpolant:

$$p(x_1, x_2, \ldots, x_n) = \sum_{i_1,i_2,\ldots,i_d=1}^{n} y_{i_1,i_2,\ldots,i_d} \prod_{j=1}^{d} L_{i_j}(x_j), \qquad \text{where}$$

$$L_i(x) = \prod_{\ell=1;\, \ell\neq i}^{n} \frac{x - \widehat{x}_\ell}{\widehat{x}_i - \widehat{x}_\ell}.$$

Note that the degree of $p(x_1, \ldots, x_d)$ in $x_j$ is $\leq n$, so that the total degree is $\leq d\,n$. The Jackson theorems can be extended to this case using

$$p_n(\cos\theta_1, \cos\theta_2, \ldots, \cos\theta_d) = \int_{[0,2\pi]^d} \prod_{j=1}^{d} \psi_n(\theta_j - \theta'_j)\, f(\cos\theta'_1, \cos\theta'_2, \ldots, \cos\theta'_d)\, d\theta'_1\, d\theta'_2 \cdots d\theta'_d,$$

with an error $\mathcal{O}(n^{-r-\alpha})$ provided $\partial^r f/\partial x_j^r$ is Hölder continuous of exponent $\alpha \in (0, 1]$ for $j' = 1, 2, \ldots, d$. Again, $p_n(x_1, \ldots, x_d)$ has degree $\leq n$ in each $x_j$ giving a total degree of $\leq d\,n$. Note that the hidden constant in "$\mathcal{O}$" can depend exponentially on the dimension $d$.

## 4.6 Seeking the Best—Minimax Approximation

Seeking the best approximation for a function $f : D \to \mathbb{R}$ from a family of functions can be a difficult undertaking, but there are ways of doing this. This can be written as an optimization problem: Given $f$ find $p \in \mathcal{F}$ where $\mathcal{F}$ is a given family of functions that minimizes $\|f - p\|$ according to a specific norm $\|\cdot\|$. For minimax approximation, we take our norm to be the maximum norm:

$$\|f - p\|_\infty = \max_{\boldsymbol{x} \in D} |f(\boldsymbol{x}) - p(\boldsymbol{x})|\,.$$

We start by looking for ways to identify when we have found a minimizing $p \in \mathcal{F}$. For the case where $\mathcal{F}$ is the set of polynomials of degree $\leq n$ in one variable and $D = [a, b]$, then the answer is found in the Chebyshev equi-oscillation theorem.

### *4.6.1 Chebyshev's Equi-oscillation Theorem*

The derivation of Chebyshev's equi-oscillation theorem starts with the recognition that any norm function and specifically the max-norm function $\|\cdot\|_\infty$ is a convex function. The second starting point is that the family $\mathcal{F}$ of all polynomials of degree $\leq n$ is a vector space.

We will have two equi-oscillation theorems: a general one, applicable to general domains and families of approximating functions, and another specialized to the case of polynomial approximations of degree $\leq n$ on an interval $[a.b]$.

First we note that any norm is convex: for $0 \leq \theta \leq 1$,

$$\|\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}\| \leq \|\theta\boldsymbol{x}\| + \|(1-\theta)\boldsymbol{y}\| = \theta\,\|\boldsymbol{x}\| + (1-\theta)\,\|\boldsymbol{y}\|\,.$$

The max-norm over a compact domain $D \subset \mathbb{R}^d$ is given by

$$\|g\|_{D,\infty} = \max_{\boldsymbol{x}\in D} |g(\boldsymbol{x})| = \max_{\boldsymbol{x}\in D,\, \sigma\in\{\pm 1\}} \sigma\, g(\boldsymbol{x}).$$

We drop the subscript "$D$" from $\|\cdot\|_{D,\infty}$ when it is clear from the context what $D$ should be.

This section uses material from Section 8.2.

Since our task amounts to minimizing a convex function, we use Lemma 8.11 to show that directional derivatives $\varphi'(\boldsymbol{x};\boldsymbol{d})$ exist for convex functions, and Theorem 8.12 to characterize a minimizer in terms of directional derivatives: $\varphi'(\boldsymbol{x};\boldsymbol{d}) \geq 0$ for all $\boldsymbol{d}$.

Our objective function

$$p \mapsto \|f - p\|_\infty = \max_{\boldsymbol{x}\in D,\, \sigma\in\{\pm 1\}} \sigma\,(f(\boldsymbol{x}) - p(\boldsymbol{x}))$$

can be written in the form $\max_{\boldsymbol{y}\in Y} \psi(p,\boldsymbol{y})$; for our problem, $Y = \{\,(\boldsymbol{x},\sigma) \mid \boldsymbol{x}\in D,\ \sigma\in\{\pm 1\}\,\}$. The set $Y$ is a closed and bounded set in $\mathbb{R}^{m+1}$, so the maximum over $Y$ exists. The function $\psi(p,(\boldsymbol{x},\sigma)) = \sigma\,(f(\boldsymbol{x}) - p(\boldsymbol{x}))$ is a smooth function of $p$, or equivalently, of the coefficients of $p$. Here is a generalization of Danskin's theorem that helps us compute these directional derivatives:

**Theorem 4.15** *Let $\psi(z,\boldsymbol{y})$ be a function where $\psi(z,\boldsymbol{y})$ and $\nabla_z\psi(z,\boldsymbol{y})$ are both continuous in $(z,\boldsymbol{y})$. If $Y$ is a closed and bounded subset of $\mathbb{R}^m$, for $\boldsymbol{z}\in\mathbb{R}^n$ we define*

$$\varphi(z) = \max_{\boldsymbol{y}\in Y} \psi(z,\boldsymbol{y}).$$

*Then for any $\boldsymbol{d}\in\mathbb{R}^n$, the directional derivative $\varphi'(z;\boldsymbol{d})$ exists and is given by*

$$\varphi'(z;\boldsymbol{d}) = \max_{\boldsymbol{y}\in Y^*(z)} \nabla_z\psi(z,\boldsymbol{y})^T\boldsymbol{d},$$

*where $Y^*(z) = \{\,\boldsymbol{y}\in Y \mid \psi(z,\boldsymbol{y}) = \varphi(z)\,\}$, the set of all $\boldsymbol{y}$'s attaining the maximum for the given $\boldsymbol{z}$*

***Proof*** Consider the quotient $(\varphi(\boldsymbol{z}+h\boldsymbol{d}) - \varphi(\boldsymbol{z}))/h$ for $h>0$. For each $h>0$, $\varphi(\boldsymbol{z}+h\boldsymbol{d}) = \psi(\boldsymbol{z}+h\boldsymbol{d},\boldsymbol{y}(h))$ for some $\boldsymbol{y}(h)\in Y$. Since $Y$ is compact there is a convergent subsequence $\boldsymbol{y}(h_k)\to\boldsymbol{y}^*\in Y$ and $h_k\to 0$ as $k\to\infty$. Now

$$\begin{aligned}&\frac{\varphi(\boldsymbol{z}+h_k\boldsymbol{d}) - \varphi(\boldsymbol{z})}{h_k}\\ &\quad= \frac{\psi(\boldsymbol{z}+h_k\boldsymbol{d},\boldsymbol{y}(h_k)) - \psi(\boldsymbol{z},\boldsymbol{y}^*)}{h_k},\\ &\qquad\text{for some } \boldsymbol{y}^*\in Y \text{ where } \varphi(\boldsymbol{z}) = \psi(\boldsymbol{z},\boldsymbol{y}^*) \geq \psi(\boldsymbol{z},\boldsymbol{y}) \text{ for all } \boldsymbol{y}\in Y\end{aligned}$$

$$= \frac{\psi(z + h_k \boldsymbol{d}, \boldsymbol{y}(h_k)) - \psi(z, \boldsymbol{y}(h_k))}{h_k} + \underbrace{\frac{\psi(z, \boldsymbol{y}(h_k)) - \psi(z, \boldsymbol{y}^*)}{h_k}}_{\leq 0}$$

$$\leq \nabla_z \psi(z + c_k h_k \boldsymbol{d}, \boldsymbol{y}(h_k))^T \boldsymbol{d} \qquad \text{for some } 0 < c_k < 1.$$

Note that by continuity of $\varphi$ and $\psi$, $\varphi(z + h_k \boldsymbol{d}) = \psi(z + h_k \boldsymbol{d}, \boldsymbol{y}(h_k)) \to \varphi(z) = \psi(z, \boldsymbol{y}^*)$, so $\boldsymbol{y}^* \in Y^*(z)$. By continuity of $\nabla_z \psi$,

$$\limsup_{k \to \infty} \frac{\varphi(z + h_k \boldsymbol{d}) - \varphi(z)}{h_k} \leq \nabla_z \psi(z, \boldsymbol{y}^*)^T \boldsymbol{d}$$

$$\leq \max_{\boldsymbol{y} \in Y^*(z)} \nabla_z \psi(z, \boldsymbol{y})^T \boldsymbol{d}.$$

On the other hand, for any $\boldsymbol{y} \in Y^*(z)$, for $h > 0$,

$$\frac{\varphi(z + h\boldsymbol{d}) - \varphi(z)}{h} \geq \frac{\psi(z + h\boldsymbol{d}, \boldsymbol{y}) - \psi(z, \boldsymbol{y})}{h}$$

$$\to \nabla_z \psi(z, \boldsymbol{y})^T \boldsymbol{d} \qquad \text{as } h \downarrow 0.$$

So

$$\liminf_{h \downarrow 0} \frac{\varphi(z + h\boldsymbol{d}) - \varphi(z)}{h} \geq \max_{\boldsymbol{y} \in Y^*(z)} \nabla_z \psi(z, \boldsymbol{y})^T \boldsymbol{d}.$$

Thus, since $\limsup_{k \to \infty} (\varphi(z + h_k \boldsymbol{d}) - \varphi(z))/h_k \leq \nabla_z \psi(z, \boldsymbol{y}^*)^T \boldsymbol{d}$,

$$\lim_{k \to \infty} \frac{\varphi(z + h_k \boldsymbol{d}) - \varphi(z)}{h_k} = \max_{\boldsymbol{y} \in Y^*(z)} \nabla_z \psi(z, \boldsymbol{y})^T \boldsymbol{d}.$$

Since there is no subsequence for which the limit can be different, the directional derivative exists and is given by

$$\varphi'(z; \boldsymbol{d}) = \max_{\boldsymbol{y} \in Y^*(z)} \nabla_z \psi(z, \boldsymbol{y})^T \boldsymbol{d},$$

as we wanted. □

Our main theorem for the one-dimensional case is below.

**Theorem 4.16** *(Chebyshev's equi-oscillation theorem) Suppose $f : [a, b] \to \mathbb{R}$ is continuous and $p$ is a polynomial of degree $\leq n$. Then $p$ minimizes $\|f - p\|_\infty := \max_{x \in [a,b]} |f(x) - p(x)|$ if and only if there are $n + 2$ points $a \leq t_0 < t_1 < t_2 < \cdots < t_{n+1} \leq b$ where*

$$f(t_i) - p(t_i) = \sigma(-1)^i \|f - p\|_\infty, \qquad i = 0, 1, 2, \ldots, n + 1 \tag{4.6.1}$$

*where $\sigma = \pm 1$.*

***Proof*** We can represent a polynomial $p(x)$ of degree $\leq n$ by means of coefficients (for example): $p(x;\boldsymbol{c}) = \sum_{k=0}^{n} c_k x^k$. The max error is then

$$\begin{aligned}\|f - p(\cdot;\boldsymbol{c})\|_\infty &= \max_{x\in[a,b]} |f(x) - p(x;\boldsymbol{c})| \\ &= \max_{s\in[a,b],\,\sigma=\pm1} \sigma\,(f(s) - p(s;\boldsymbol{c}))\,.\end{aligned}$$

So let us set $Y = \{\,(s,\sigma) \mid s \in [a,b],\ \sigma = \pm 1\,\}$ and $\psi(\boldsymbol{c},\boldsymbol{s}) = \sigma\,(f(s) - p(s;\boldsymbol{c}))$, where $\boldsymbol{s} = (s,\sigma)$. Then $\varphi(\boldsymbol{c}) = \|f - p(\cdot;\boldsymbol{c})\|_\infty$. In what follows, we will assume that $\|f - p(\cdot;\boldsymbol{c})\|_\infty > 0$; otherwise, $f = p(\cdot;\boldsymbol{c})$ and $p(\cdot;\boldsymbol{c})$ is clearly the minimax approximation and the equi-oscillation condition (4.6.1) is satisfied.

So what we need to do now is to compute $\varphi'(\boldsymbol{c};\boldsymbol{d})$ for any given $\boldsymbol{d}$ and show that $\varphi'(\boldsymbol{c};\boldsymbol{d}) \geq 0$ for all $\boldsymbol{d}$ if and only if the equi-oscillation condition holds.

First: note that $Y^*(\boldsymbol{c})$ is the set of $(s,\sigma)$ where $\sigma\,(f(s) - p(s;\boldsymbol{c})) = \|f - p(\cdot;\boldsymbol{c})\|_\infty$. Any such $s \in [a,b]$ for which $|f(s) - p(s;\boldsymbol{c})| = \|f - p(\cdot;\boldsymbol{c})\|_\infty$ we will call an *equi-oscillation point*. Now

$$\begin{aligned}\varphi'(\boldsymbol{c};\boldsymbol{d}) &= \max_{(s,\sigma)\in Y^*(\boldsymbol{c})} \nabla_{\boldsymbol{c}}\psi(\boldsymbol{c},s,\sigma)^T\boldsymbol{d} \\ &= \max_{(s,\sigma)\in Y^*(\boldsymbol{c})} \sigma \sum_{i=0}^{n} \frac{\partial}{\partial c_i}\left(f(s) - \sum_{i=0}^{n} c_i s^i\right) d_i \\ &= \max_{(s,\sigma)\in Y^*(\boldsymbol{c})} \sigma \sum_{i=0}^{n} \left(-s^i\right) d_i = \max_{(s,\sigma)\in Y^*(\boldsymbol{c})} -\sigma\, d(s),\end{aligned}$$

where $d(s) = \sum_{i=0}^{n} d_i s^i$. For an equi-oscillation point $s$, if $(s,\sigma) \in Y^*(\boldsymbol{c})$, then $\sigma = \operatorname{sign}(f(s) - p(s;\boldsymbol{c}))$. Now $d(s)$ can be an arbitrary polynomial of degree $\leq n$. If there are $n+2$ points $t_i$ satisfying the equi-oscillation condition (4.6.1), then in order to have $\varphi'(\boldsymbol{c};\boldsymbol{d}) < 0$ we need

$$-\operatorname{sign}(f(t_i) - p(t_i;\boldsymbol{c}))\; d(t_i) < 0$$

as $t_i$ is an equi-oscillation point for each $i$. Thus $\operatorname{sign} d(t_i) = -\operatorname{sign} d(t_{i+1})$ for $i = 0,\ 1,\ \ldots,\ n+1$, and $d(s)$ would have $n+1$ roots in the interval $[a,b]$. This implies that $d(s) \equiv 0$ and so we cannot have $\varphi'(\boldsymbol{c};\boldsymbol{d}) < 0$.

On the other hand, suppose that $\varphi'(\boldsymbol{c};\boldsymbol{d}) \geq 0$ for all $\boldsymbol{d}$. We then need to find the points $a \leq t_0 < t_1 < t_2 < \cdots < t_{n+1} \leq b$ satisfying (4.6.1). We need to deal with two sets of points in $[a,b]$:

$$\begin{aligned}S_+ &= \left\{\, s \in [a,b] \mid f(s) - p(s;\boldsymbol{c}) = +\|f - p(\cdot;\boldsymbol{c})\|_\infty \,\right\} \qquad \text{and} \\ S_- &= \left\{\, s \in [a,b] \mid f(s) - p(s;\boldsymbol{c}) = -\|f - p(\cdot;\boldsymbol{c})\|_\infty \,\right\}.\end{aligned}$$

Clearly the union $S_+ \cup S_-$ is the set of equi-oscillation points, which is non-empty. Also, both sets $S_\pm$ are closed sets.

Let $u_0 = \min\{\, s \in [a,b] \mid s \in S_+ \cup S_- \,\} = \min(S_+ \cup S_-)$. Then $u_0 \in S_+$ or $u_0 \in S_-$ (exclusive). We let $\widehat{\sigma} \in \{\pm 1\}$ be chosen so that $u_0 \in S_{\widehat{\sigma}}$, $S_{\widehat{\sigma}}$ being $S_+$ or $S_-$ according to whether $\widehat{\sigma} = +1$ or $-1$, respectively. Then let $u_1 = \min S_{-\widehat{\sigma}}$, the first equi-oscillation point where the error has the opposite sign to the error at $u_0$. Now define $v_0 = \max\{\, s \in S_{\widehat{\sigma}} \mid s \le u_1 \,\}$. We can continue defining $u_k = \min\left\{\, s \in S_{(-1)^k\widehat{\sigma}} \mid s \ge u_{k-1} \,\right\}$ and then $v_{k-1} = \max\left\{\, s \in S_{(-1)^{k-1}\widehat{\sigma}} \mid s \le u_k \,\right\}$ until one of the sets becomes empty: say $\left\{\, s \in S_{(-1)^{m+1}\widehat{\sigma}} \mid s \ge u_m \,\right\} = \emptyset$. Then $a \le u_0 \le v_0 < u_1 \le v_1 < u_2 \le \cdots \le v_m \le b$, and $S_{\widehat{\sigma}} \subseteq \bigcup_{j:2j\le m}[u_{2j}, v_{2j}]$ and $S_{-\widehat{\sigma}} \subseteq \bigcup_{j:2j+1\le m}[u_{2j+1}, v_{2j+1}]$. Also, $\operatorname{sign}(f(u_k) - p(u_k; \boldsymbol{c})) = (-1)^k\widehat{\sigma}$. We can choose $m$ points $w_k$ where $v_k < w_k < u_{k+1}$ and set $d(s) = \widehat{\sigma}\prod_{k=0}^{m-1}(w_k - s)$ which is a polynomial of degree $m$. Let $d(s) = \sum_{i=0}^{n} d_i s^i$, provided $m \le n$. Then for $s \in [u_k, v_k]$, $\operatorname{sign} d(s) = \widehat{\sigma}(-1)^k$; for $s \in S_{\pm\widehat{\sigma}}$, $\operatorname{sign}(f(s) - p(s; \boldsymbol{c})) = \pm\widehat{\sigma}$. Thus

$$\begin{aligned}\varphi'(\boldsymbol{c}; \boldsymbol{d}) &= \max_{(s,\sigma)\in Y^*(\boldsymbol{c})} -\sigma\, d(s) \\ &= \max\Big(\max_{s\in S_{\widehat{\sigma}}} -\widehat{\sigma}\, d(s),\ \max_{s\in S_{-\widehat{\sigma}}} +\widehat{\sigma}\, d(s)\Big).\end{aligned}$$

But for $s \in S_{\widehat{\sigma}}$, $\operatorname{sign} -\widehat{\sigma}\, d(s) = -\widehat{\sigma}^2(-1)^{2j} = -1$ for some integer $j$, while for $s \in S_{-\widehat{\sigma}}$, $\operatorname{sign} +\widehat{\sigma}\, d(s) = +\widehat{\sigma}^2(-1)^{2j+1} = -1$. Thus $\varphi'(\boldsymbol{c}; \boldsymbol{d}) < 0$ for this $\boldsymbol{d}$ if $m \le n$.

Thus we conclude that if $\varphi'(\boldsymbol{c}; \boldsymbol{d}) \ge 0$ for all $\boldsymbol{d}$, then we must have $m \ge n+1$. Then we can take $t_k = u_k$ for $k = 0, 1, 2, \ldots, n+1$ as the points satisfying the equi-oscillation condition (4.6.1), as we wanted. □

Related to the Chebyshev equi-oscillation theorem is the theorem of de la Vallée-Poussin which gives a *lower bound* on the minimax approximation error.

**Theorem 4.17** *(Theorem of de la Vallée-Poussin) Suppose $f : [a,b] \to \mathbb{R}$ is continuous and $p$ is a polynomial of degree $\le n$. Suppose there are $n+2$ points $a \le t_0 < t_1 < t_2 < \cdots < t_{n+1} \le b$ where*

$$f(t_i) - p(t_i) = \sigma(-1)^i E_i, \qquad i = 0, 1, 2, \ldots, n+1 \tag{4.6.2}$$

*where $\sigma = \pm 1$. Then for any polynomial $q$ of degree $\le n$,*

$$\|f - q\|_\infty \ge \min_{i=0,1,\ldots,n+1} E_i.$$

***Proof*** We prove this by contradiction. Suppose $\|f - q\|_\infty < \min_{i=0,1,\ldots,n+1} E_i$. Suppose also the $E_i > 0$ for all $i$, as otherwise there is nothing to prove. Then

$$\begin{aligned}q(t_i) - p(t_i) &= (f(t_i) - p(t_i)) - (f(t_i) - q(t_i)) \\ &= \sigma(-1)^i E_i - (f(t_i) - q(t_i)).\end{aligned}$$

Since $\|f - q\|_\infty < \min_{i=0,1,\ldots,n+1} E_i$, it follows that $\text{sign}(q(t_i) - p(t_i)) = \sigma(-1)^i$. Thus $q(t) - p(t)$ oscillates in sign $n+2$ times on $[a, b]$, and so $q - p$ has at least $n+1$ roots. But $p$ and $q$ are polynomials of degree $\le n$, so $p - q$ is a polynomial of degree $\le n$. Thus $q - p = 0$ which contradicts $\|f - q\|_\infty < \min_{i=0,1,\ldots,n+1} E_i$. □

### 4.6.2 *Chebyshev Polynomials and Interpolation*

Chebyshev polynomials are polynomials with an equi-oscillation property. Chebyshev polynomials are usually defined implicitly by

$$T_k(\cos\theta) = \cos(k\theta), \qquad k = 0, 1, 2, \ldots. \tag{4.6.3}$$

At first it can be hard to see why these should be polynomials. We can start with some examples:

$$\begin{aligned}
T_0(\cos\theta) &= \cos(0\theta) = 1,\\
T_1(\cos\theta) &= \cos(1\theta) = \cos\theta,\\
T_2(\cos\theta) &= \cos(2\theta) = 2\cos^2\theta - 1,\\
T_3(\cos\theta) &= \cos(3\theta) = 4\cos^3\theta - 3\cos\theta.
\end{aligned}$$

That is, $T_0(x) = 1$, $T_1(x) = x$, $T_2(x) = 2x^2 - 1$, and $T_3(x) = 4x^3 - 3x$. We can prove the fact that $T_k$ is a polynomial for $k = 0, 1, 2, \ldots$ and provide a useful means of computing Chebyshev polynomials as follows:

$$\begin{aligned}
T_{k+1}(\cos\theta) &= \cos((k+1)\theta) = \cos(k\theta)\cos\theta - \sin(k\theta)\sin\theta,\\
T_{k-1}(\cos\theta) &= \cos((k-1)\theta) = \cos(k\theta)\cos\theta + \sin(k\theta)\sin\theta.
\end{aligned}$$

Adding gives

$$T_{k+1}(\cos\theta) + T_{k-1}(\cos\theta) = 2\cos(k\theta)\cos\theta = 2\,T_k(\cos\theta)\cos\theta.$$

That is,

$$\begin{aligned}
T_{k+1}(x) + T_{k-1}(x) &= 2x\,T_k(x), \qquad \text{or equivalently,}\\
T_{k+1}(x) &= 2x\,T_k(x) - T_{k-1}(x).
\end{aligned} \tag{4.6.4}$$

Since we already know that $T_0$ and $T_1$ are polynomials, we can use induction to see that $T_k$ is a polynomial for $k = 0, 1, 2, \ldots$. Furthermore, the degree of $T_k$ is exactly $k$, and the leading coefficient of $T_k(x)$ is $\max(1, 2^{k-1})$.

Figure 4.6.1 shows the Chebyshev polynomials $T_k(x)$ for $k = 0, 1, 2, 3, 4$ for $-1 \le x \le +1$. The equi-oscillation properties of these Chebyshev polynomials are clearly visible in Figure 4.6.1.

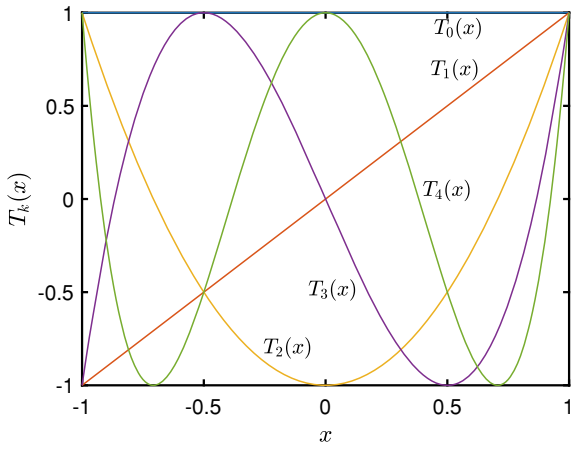



**Fig. 4.6.1** Chebyshev polynomials for $x \in [-1, +1]$

The equi-oscillation property of Chebyshev polynomials gives them a valuable minimax property.

**Theorem 4.18** *The polynomial* $2^{-k}\, T_{k+1}(x) = x^{k+1} - q(x)$ *where* $q(x)$ *minimizes*

$$\max_{-1 \le x \le +1} \left| x^{k+1} - q(x) \right|$$

*over all polynomials* $q$ *of degree* $\le k$.

***Proof*** By Theorem 4.16, it is sufficient to find $k+2$ points $t_i$ where $t_i^{k+1} - q(t_i)$ oscillate in sign and $\left| t_i^{k+1} - q(t_i) \right| = \max_{-1 \le x \le +1} \left| x^{k+1} - q(x) \right|$. First we show that

$$\max_{-1 \le x \le +1} \left| x^{k+1} - q(x) \right| = 2^{-k}.$$

For $-1 \le x \le +1$, we can write $x = \cos\theta$ for some $\theta$; then $T_{k+1}(x) = T_{k+1}(\cos\theta) = \cos((k+1)\theta) \in [-1, +1]$, and therefore $\max_{-1 \le x \le +1} \left| x^{k+1} - q(x) \right| = \max_{-1 \le x \le +1} 2^{-k} \left| T_{k+1}(x) \right| \le 1$. Setting $x = +1 = \cos 0$ shows that this bound is attained as $T_{k+1}(1) = 1$.

The equi-oscillation points we can choose are $t_i = -\cos(i\pi/(k+1))$ for $i = 0, 1, 2, \ldots, k+1$:

$$\begin{aligned} t_i^{k+1} - q(t_i) &= 2^{-k}\, T_{k+1}(t_i) = 2^{-k}(-1)^{k+1} \cos((k+1)i\pi/(k+1)) \\ &= 2^{-k}(-1)^{k+1}(-1)^i. \end{aligned}$$

These are the $k+2$ equi-oscillation points that we sought, and so $q$ achieves the minimum. □

If we consider the interpolation error for interpolating a function $f$ by a polynomial $p$ of degree $\le n$, by (4.1.7),

**Algorithm 57** Remez algorithm

```
1  function remez(f, t, ϵ, a, b)
2    done ← false
3    while not done
4      solve a_0 + a_1 t_i + a_2 t_i^2 + ··· + a_n t_i^n + (−1)^i E = f(t_i)
5        for a_0, a_1, ..., a_n, E
6      let p(x) = a_0 + a_1 x + ··· + a_n x^n
7      t_0, t_1, ..., t_{n+1} ← local maxima of |f(x) − p(x)| on [a, b]
8      if max_i ||p(t_i) − f(t_i)| − |E|| < ϵ: done ← true; end if
9    end while
10 end function
```

$$f(x) - p(x) = \frac{f^{(n+1)}(c_x)}{(n+1)!}(x - x_0)\cdots(x - x_n).$$

If we want to interpolate over the interval $[-1, +1]$, it is reasonable to focus on minimizing

$$\max_{-1\le x\le +1} |(x - x_0)\cdots(x - x_n)|$$

by choosing appropriate interpolation points. Noting that $(x - x_0)\cdots(x - x_n) = x^{n+1} - q(x)$ for some polynomial $q$ of degree $\le n$, we solve this minimization problem by setting $x^{n+1} - q(x) = 2^{-n}\, T_{n+1}(x)$. What, then should the interpolation points be? They should be the roots of $T_{n+1}(x)$, which are

$$x_i = \cos\left(\frac{(i + \frac{1}{2})\pi}{n+1}\right), \qquad \text{for } i = 0, 1, 2, \ldots, n. \tag{4.6.5}$$

For interpolation on a more general interval $[a, b]$, we use

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{(i + \frac{1}{2})\pi}{n+1}\right), \qquad \text{for } i = 0, 1, 2, \ldots, n. \tag{4.6.6}$$

### *4.6.3 Remez Algorithm*

While using Chebyshev interpolation points (4.6.6) often gives us a near-minimax approximation, how do we obtain an actual minimax approximation? Evgenii Remez gave an algorithm in [212] (1934) for computing minimax polynomial approximations based on the Chebyshev equi-oscillation theorem (Theorem 4.16). See Algorithm 57. This is actually *Remez' second algorithm* for minimax approximation.

Difficulties in implementing this algorithm come from trying to find $n + 2$ local maximizers of $|f(x) - p(x)|$. Since the polynomial $p(x)$ satisfies $f(t_i) - p(t_i) = (-1)^i E$ for all $i$, we have oscillation of signs of $f(x) - p(x)$, and so we can find $n + 2$ local minimizers, with at least one between zeros of $f(x) - p(x)$ adjacent

to each $t_i$, or between a zero of $f(x) - p(x)$ and an endpoint of the interval $[a, b]$. One of the local maximizers chosen in line 7 should be the global maximizer of $|f(x) - p(x)|$.

### 4.6.4 Minimax Approximation in Higher Dimensions

In two or more dimensions, the Chebyshev equi-oscillation theorem (Theorem 4.16) does not hold, as there is no longer an ordering of points. In this more general setting, we consider the problem of finding a polynomial $p$ of degree $\leq k$ that minimizes

$$\|f - p\|_\infty = \max_{\boldsymbol{x} \in D} |f(\boldsymbol{x}) - p(\boldsymbol{x})| \,. \tag{4.6.7}$$

We assume that $D$ is a closed and bounded set in $\mathbb{R}^d$.

If we follow the steps in our proof of Theorem 4.16, we noted that the function $p \mapsto \varphi(p) := \|f - p\|_\infty$ is a convex function which is defined in terms of a maximum of $\psi(p, \boldsymbol{x}, \sigma) := \sigma(f(\boldsymbol{x}) - p(\boldsymbol{x}))$ over $(\boldsymbol{x}, \sigma) \in D \times \{\pm 1\}$. Then applying Theorem 4.15, we have a global minimizer at $p$ if and only if for any polynomial $q(\boldsymbol{x})$ of degree $\leq k$ we have

$$\max_{(\boldsymbol{x},\sigma) \in Y^*(p)} \psi'(p, \boldsymbol{x}, \sigma; q) \geq 0, \qquad \text{where}$$
$$Y^*(p) = \left\{ (\boldsymbol{x}, \sigma) \in D \times \{\pm 1\} \mid \sigma(f(\boldsymbol{x}) - p(\boldsymbol{x})) = \|f - p\|_\infty \right\}.$$

That is, $p$ is a global minimizer if for any polynomial $q(\boldsymbol{x})$ of degree $\leq k$ we have

$$\max_{(\boldsymbol{x},\sigma) \in Y^*(p)} -\sigma\, q(\boldsymbol{x}) \geq 0.$$

We call the set $\{\boldsymbol{x} \in D \mid \text{there is } \sigma = \pm 1 \text{ where } (\boldsymbol{x}, \sigma) \in Y^*(p)\}$ the *equi-oscillation set* for $p$ over $D$. Given $p$, each element of the equi-oscillation set is an equi-oscillation point, and we can consider each equi-oscillation point $\boldsymbol{x}$ to be labeled by the sign $\sigma$ where $(\boldsymbol{x}, \sigma) \in Y^*(p)$.

**Example 4.19** As an example, consider the problem of minimizing $\|f - p\|_\infty$ where $f(x, y) = xy$ and $p$ ranges over all linear functions on $D = [-1, +1]^2$. The solution is, in fact, $p(x, y) \equiv 0$. The points $(x, y, \sigma) \in D \times \{\pm 1\}$ are given by the four corners of $D$ with $\sigma = +1$ at $(x, y) = (+1, +1)$ and $(x, y) = (-1, -1)$ while $\sigma = -1$ at $(x, y) = (+1, -1)$ and $(x, y) = (-1, +1)$. No linear function $q(x, y)$ can be negative at $(x, y) = (+1, +1)$ and $(x, y) = (-1, -1)$ and positive at $(x, y) = (+1, -1)$ and $(x, y) = (-1, +1)$. Thus $p(x, y) \equiv 0$ is the minimax linear approximation to $f(x, y) = xy$.

Another example is the minimax linear approximation to $f(x, y) = x^2 + y^2$ over $D = [-1, +1]^2$, which is $p(x, y) = 1$. The maximum error occurs at the corners of

$D$ and at $(x, y) = (0, 0)$. Any linear function that is positive at the corners must be positive at the center also, so $p$ must be the minimax linear approximation.

#### 4.6.4.1 Identifying and Finding Minimax Approximations over $D$

Note that if

$$X^*(p) := \left\{ \boldsymbol{x} \mid \text{there is } \sigma \in \{\pm 1\} \text{ where } (\boldsymbol{x}, \sigma) \in Y^*(p) \right\}$$

is a set of interpolation points for degree $k$ interpolation on $D$, then $p$ is *not* a minimax approximation of degree $\leq k$.

Another way of thinking about minimax approximation is that it is a kind of *linear program*: minimizing a linear function subject to linear equality and inequality constraints:

$$\min_{z} \boldsymbol{c}^T \boldsymbol{z} \qquad \text{subject to } A\boldsymbol{z} \geq \boldsymbol{b}, \tag{4.6.8}$$

where the inequality "$A\boldsymbol{z} \geq \boldsymbol{b}$" is understood componentwise: $(A\boldsymbol{z})_i \geq b_i$ for all $i$. Note that $\sigma(f(\boldsymbol{x}) - p(\boldsymbol{x}))$ is a linear function of $p$. Our minimax approximation problem is then

$$\min_{p,E} E \qquad \text{subject to } E - \sigma(f(\boldsymbol{x}) - p(\boldsymbol{x})) \geq 0 \qquad \text{for all } \boldsymbol{x} \in D,\ \sigma \in \{\pm 1\}. \tag{4.6.9}$$

The difficulty with (4.6.9) is that there are infinitely many constraints. We can replace $D$ in (4.6.9) with a finite subset $D_N$ for some parameter $N > 0$ representing the number of points in $D$ chosen. The question then becomes the trade-off between accuracy and efficiency: larger $N$ means $D_N$ gives better approximations, but greater computational cost. If $D_N \times \{\pm 1\}$ contains $Y^*(p^*)$ for the minimax approximation $p^*$, then the solution (4.6.9) will be $p^*$. The task then becomes one of trying to estimate $Y^*(p^*)$. If we choose $D_N$ to be an interpolation set for degree $k$ interpolation over $D$, then solving

$$\min_{p,E} E \qquad \text{subject to } E - \sigma(f(\boldsymbol{x}) - p(\boldsymbol{x})) \geq 0 \qquad \text{for all } \boldsymbol{x} \in D_N,\ \sigma \in \{\pm 1\} \tag{4.6.10}$$

is equivalent to interpolating $f(\boldsymbol{x}) - p(\boldsymbol{x}) = \sigma E$ for $\boldsymbol{x} \in D_N$ and appropriate choice of sign $\sigma$ for each $\boldsymbol{x}$.

**Exercises.**

(1) Show that if $q(x)$ is the minimax approximation to $x^{n+1}$ over $-1 \leq x \leq +1$ then $x^{n+1} - q(x) = 2^{-n} T_{n+1}(x)$.

(2) Implement Remez' second algorithm (Algorithm 57). The hardest line to implement is line 7. One way of doing this is to evaluate $e(t) := f(t) - p(t)$ at $N + 1$

equally spaced points $s_k = a + k(b-a)/N$ in $[a, b]$ with $N \gg n$. Local maxima and local minima of $e(t)$ can be identified by $e(s_{k-1}) < e(s_k) > e(s_{k+1})$ and $e(s_{k-1}) > e(s_k) < e(s_{k+1})$, respectively. Remember we want to alternate between local minima and local maxima. Comparing $e(s_0)$ with $e(s_1)$ can tell you which to look for first. What to do if there are not enough or too many local maxima and local minima? Try adding arbitrary points if there are not enough, and removing some if there are too many. Initialize with the Chebyshev interpolant. Test at least on $f(x) = e^x$ over $[0, 1]$ and $f(x) = 1/(1 + x^2)$ over $[-5, +5]$ with various $n$.

(3) It is possible to use Newton's method *in some cases* to find minimax polynomial approximations via Chebyshev's equi-oscillation theorem. Here $f : [a, b] \to \mathbb{R}$ is the function to be approximated, which we assume to be smooth. We wish to find a polynomial $p(x) = \sum_{k=0}^{n} a_k x^k$ with coefficients $a_0, a_1, \ldots, a_n$ to be determined as well as $E = \|f - p\|_\infty$ and the equi-oscillation points $a \le z_0 < z_1 < \cdots < z_n < z_{n+1} \le b$. Count the number of unknowns (it should be $2n + 4$). The equations to be solved for are $z_0 = a, z_{n+1} = b$, the equi-oscillation conditions $f(z_j) - p(z_j) = \sigma(-1)^j E$ for $j = 0, 1, 2, \ldots, n + 1$, and from the fact that the $z_j$'s are local minimizers, $f'(z_j) - p'(z_j) = 0$ for $j = 1, 2, \ldots, n$. Count the number of equations. Set up the (square) system of nonlinear equations. Compute the Jacobian matrix for this system and apply a guarded Newton's method. Test this on $f(x) = e^x$ over $[0, 1]$ and $n = 3$ and $n = 4$ using the Chebyshev interpolant for $p(x)$ and $z_j = \frac{1}{2}(a + b) - \frac{1}{2}(b - a)\cos(j\pi/(n + 1))$, $j = 0, 1, 2, \ldots, n + 1$, as the starting point.

(4) Show that if $f(x, y) = x\,y$ then the zero function is the linear minimax approximation to $f$ over $[-1, +1] \times [-1, +1]$. [**Hint:** The points where the minimax error occurs are the vertices of the square.]

(5) Suppose that $D \subset \mathbb{R}^d$ is closed and bounded, $f : D \to \mathbb{R}$ is continuous, and we are looking for a minimax approximation $p$ to $f$ in a finite-dimensional vector space $\mathcal{P}$ of continuous functions $D \to \mathbb{R}$. Call a set $S \subset D$ an *interpolation set* for $\mathcal{P}$ if for any values $v : S \to \mathbb{R}$ there is a unique interpolant $q \in \mathcal{P}$ where $q(\boldsymbol{x}) = v(\boldsymbol{x})$ for all $\boldsymbol{x} \in S$. Show that if a candidate approximation $p \in \mathcal{P}$ has the property that the equi-oscillation set for $p$ is contained in an interpolation set for $\mathcal{P}$, then $p$ is *not* a minimax approximation to $f$.

(6) Suppose that $\mathcal{P}$ is the set of quadratic functions of two variables over a triangle $D$, $f : D \to \mathbb{R}$ is continuous, and the equi-oscillation set for $p \in \mathcal{P}$ over $D$ consists of the vertices of the triangle and a point in the interior of $D$. Show that $p$ is not a minimax approximation for $f$ over $D$. [**Hint:** Show that for any assignment of signs to the equi-oscillation points, there is a quadratic function that has the specified signs at the specified equi-oscillation points.]

(7) Devise a linear program to identify if $Y^*(p) = \{(\boldsymbol{x}_1, \sigma_1), (\boldsymbol{x}_2, \sigma_2), \cdots, (\boldsymbol{x}_r, \sigma_r)\}$ has the properties to imply $p \in \mathcal{P}$ is a minimax approximation to $f : D \to \mathbb{R}$ where $\mathcal{P} = \text{span}\{\phi_1, \phi_2, \ldots, \phi_m\}$. [**Hint:** We want to know if there is a $d \in \mathcal{P}$ where $\max_{i=1,2,\ldots,r} \sigma_i\, d(\boldsymbol{x}_i) < 0$. Write $d = \sum_{j=1}^{m} c_j \phi_j$. Inequalities $v \ge \sigma_i \sum_{j=1}^{m} c_j \phi_j(\boldsymbol{x}_i)$ for all $i$ ensures that $v \ge \max_{i=1,2,\ldots,r} \sigma_i\, d(\boldsymbol{x}_i)$. To

ensure that the problem is bounded we can add inequalities $-1 \le c_j \le +1$ for all $j$. Now minimize $v$ over all $c_j$'s and satisfying the constraints. Show that $v = \max_{i=1,2,\ldots,r} \sigma_i \, d(\boldsymbol{x}_i)$ at the minimum.]

(8) Chebyshev filters are often used in Electrical Engineering. These have some good properties in that they have little distortion within the desired band, but rapid dropoff outside the desired band. For example, a Chebyshev low-pass filter has the frequency response function

$$G(\omega) = \frac{1}{\sqrt{1 + \epsilon^2 \, T_n(\omega/\omega_0)^2}}.$$

To implement a Chebyshev filter is to find a rational function $F(z)$ with real coefficients where $|F(i\omega)| = G(\omega)$ for all $\omega$; for a realizable filter (that is, one that can actually be built), we need all the roots of the denominator of $F(z)$ to have negative real part. Assume that $\omega_0 = 1$.

(a) Show that $|F(z)|^2 = F(z)\,\overline{F(z)} = F(z)\,F(-z)$ for $z = i\omega$ since $F$ is a rational function with real coefficients.
(b) If we write $F(z) = P(z)/Q(z)$ with $P$ and $Q$ are polynomials with real coefficients, then the roots of $Q(z)\,Q(-z)$ are zeros of $1 + \epsilon^2 \, T_n(-iz)\,T_n(+iz)$.
(c) The zeros of $1 + \epsilon^2 \, (-1)^n \, T_n(+iz)^2$ are given by $T_n(+iz) = \pm \, (-1)^{(n+1)/2}/\epsilon$. The definition of $T_n$ is usually made via $T_n(\cos\theta) = \cos(n\theta)$, so $T_n(\frac{1}{2}(e^{i\theta} + e^{-i\theta})) = \frac{1}{2}(e^{in\theta} + e^{-in\theta})$. Thus, we want to solve $\frac{1}{2}(e^{in\theta} + e^{-in\theta}) = \pm(-1)^{(n+1)/2}/\epsilon$. Solve this equation for $e^{in\theta}$, and so find solutions $z$ for $1 + \epsilon^2 \, (-1)^n \, T_n(+iz)^2 = 0$.
(d) Find the zeros of $Q(z)$ for $n = 4$. Let $Q(z) = \prod_{j=1}^{r}(z - z_r)$ where $z_r$ are the roots of $Q$.
(e) Since $P(i\omega)\,P(-i\omega)/(Q(i\omega)\,Q(-i\omega)) = G(\omega)^2$ for real $\omega$, compute $P(z)$ for $n = 4$.
(f) Compute $F(z) = P(z)/Q(z)$. Plot $|F(i\omega)|$ against $\omega$.

(9) Consider the average absolute error $\int_a^b |f(x) - p(x)| \, dx$. By differentiating this with respect the coefficients of $p(x)$, give the equations for minimizing $\int_a^b |f(x) - p(x)| \, dx$ over all polynomials $p$ of degree $\le n$. Apply this to the problem of minimizing $\int_0^1 |e^x - a| \, dx$ with respect to $a$.

(10) *Padé approximation* is about rational approximation of smooth functions using Taylor series. These have the form $f(x) \approx p(x)/q(x)$ where $p$ and $q$ are polynomials of the appropriate degrees with $q(0) = 1$. If we choose deg $p = m$ and deg $q = n$ then we can match Taylor series of $f$ up to $x^{m+n}$. Find the Padé approximation of $f(x) = e^x \approx p(x)/q(x)$ with deg $p$ = deg $q = 3$. Plot this Padé approximation against $f(x)$ for $|x| \le 1$. [**Hint:** Write $f(x)\,q(x) = p(x)$ and expand as power series in $x$, using the fact that $q(0) = 1$. This gives a linear system of equations for the coefficients of both $p(x)$ and $q(x)$.]

(11) Show that the derivative $T_n'(\cos\theta) = n \sin(n\theta)/\sin\theta$. Also show that $U_n(\cos\theta) = \sin((n+1)\theta)/\sin\theta$ is a polynomial of degree $n$ in $\cos\theta$ by show-

ing the recurrence $U_{n+1}(x) + U_{n-1}(x) = 2x\,U_n(x)$ with $U_0(x) = 1$ and $U_1(x) = 2x$. These polynomials $U_n$ are called *Chebyshev polynomials of the second kind*.

## 4.7 Seeking the Best—Least Squares

If the objective function is quadratic, then setting the derivative to zero means solving a linear equation. This makes the task much easier computationally. And so we look at what that means for approximating functions using the 2-norm:

$$\|f - p\|_2 = \left[\int_D (f(\boldsymbol{x}) - p(\boldsymbol{x}))^2\, d\boldsymbol{x}\right]^{1/2}.$$

Squaring the 2-norm gives the integral of the square of $f - p$, which is a quadratic function of $p$. If we represent $p$ as a linear combination of a finite number of basic functions, then we just need to solve a finite system of linear equations. Since we understand a great deal about solving linear systems of equations, this is a straightforward way of approximating functions.

### *4.7.1 Solving Least Squares*

For finite least squares problems that involves a finite sum of squares, instead of integrals, we have the normal equations (2.2.3) from Section 2.2.1: to minimize $\|A\boldsymbol{x} - \boldsymbol{b}\|_2^2$ over $\boldsymbol{x}$, we solve $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$.

The corresponding equations for least squares approximations are given by writing $p(\boldsymbol{x})$ in terms of basis functions:

$$p(\boldsymbol{x}) = \sum_{i=1}^{N} c_i \phi_i(\boldsymbol{x}),$$

where $\{\,\phi_i \mid i = 1, 2, \ldots, N\,\}$ is a basis for our space of approximating functions $\mathcal{P}$. Let $\boldsymbol{\phi}(\boldsymbol{x}) := [\phi_1(\boldsymbol{x}), \ldots, \phi_N(\boldsymbol{x})]^T$ be the vector of basis functions, so $p(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x})$. Then

$$J(\boldsymbol{c}) := \int_D (f(\boldsymbol{x}) - \boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}))^2\, d\boldsymbol{x} = \int_D (f(\boldsymbol{x}) - p(\boldsymbol{x}))^2\, d\boldsymbol{x},$$

which is to be minimized over $\boldsymbol{c}$. Our objective function $J$ is a convex function of $\boldsymbol{c}$. We can compute the gradient of $J$ by

$$\begin{aligned}
\nabla J(\boldsymbol{c})^T \boldsymbol{d} &= \frac{d}{ds} J(\boldsymbol{c} + s\boldsymbol{d})\Big|_{s=0} \\
&= \frac{d}{ds} \int_D (f(\boldsymbol{x}) - (\boldsymbol{c} + s\boldsymbol{d})^T \boldsymbol{\phi}(\boldsymbol{x}))^2 \, d\boldsymbol{x}\Big|_{s=0} \\
&= 2 \int_D (f(\boldsymbol{x}) - (\boldsymbol{c} + s\boldsymbol{d})^T \boldsymbol{\phi}(\boldsymbol{x})) \frac{d}{ds} (f(\boldsymbol{x}) - (\boldsymbol{c} + s\boldsymbol{d})^T \boldsymbol{\phi}(\boldsymbol{x})) \, d\boldsymbol{x}\Big|_{s=0} \\
&= -2 \int_D (f(\boldsymbol{x}) - \boldsymbol{c}^T \boldsymbol{\phi}(\boldsymbol{x})) \, \boldsymbol{d}^T \boldsymbol{\phi}(\boldsymbol{x}) \, d\boldsymbol{x} \\
&= -2 \left[ \int_D (f(\boldsymbol{x}) - \boldsymbol{c}^T \boldsymbol{\phi}(\boldsymbol{x})) \, \boldsymbol{\phi}(\boldsymbol{x}) \, d\boldsymbol{x} \right]^T \boldsymbol{d}.
\end{aligned}$$

So the condition that $\nabla J(\boldsymbol{c}) = \boldsymbol{0}$ means that

$$\left[ \int_D \boldsymbol{\phi}(\boldsymbol{x}) \; \boldsymbol{\phi}(\boldsymbol{x})^T \, d\boldsymbol{x} \right] \boldsymbol{c} = \int_D f(\boldsymbol{x}) \, \boldsymbol{\phi}(\boldsymbol{x}) \, d\boldsymbol{x}. \tag{4.7.1}$$

These are the normal equations for least squares approximation of functions. We should note that the matrix

$$B = \int_D \boldsymbol{\phi}(\boldsymbol{x}) \, \boldsymbol{\phi}(\boldsymbol{x})^T \, d\boldsymbol{x} \tag{4.7.2}$$

is symmetric and positive semi-definite. Symmetry is clear because the integrand is symmetric. To show that $B$ is positive semi-definite,

$$\boldsymbol{z}^T B \boldsymbol{z} = \int_D \boldsymbol{z}^T \boldsymbol{\phi}(\boldsymbol{x}) \, \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{z} \, d\boldsymbol{x} = \int_D (\boldsymbol{z}^T \boldsymbol{\phi}(\boldsymbol{x}))^2 \, d\boldsymbol{x} \geq 0.$$

For $B$ to be positive definite, we need $\boldsymbol{z} \neq \boldsymbol{0}$ implies that $(\boldsymbol{z}^T \boldsymbol{\phi}(\boldsymbol{x}))^2 > 0$ on a set of positive volume in $D$. This is the case if the interpolation functions are continuous, and $\{\phi_1, \phi_2, \ldots, \phi_N\}$ are linearly independent over $D$.

If we pre-multiply (4.7.1) by $\boldsymbol{c}^T$, we obtain

$$\begin{aligned}
\int_D p(\boldsymbol{x})^2 \, d\boldsymbol{x} &= \int_D (\boldsymbol{c}^T \boldsymbol{\phi}(\boldsymbol{x}))^2 d\boldsymbol{x} = \int_D p(\boldsymbol{x}) \, f(\boldsymbol{x}) \, d\boldsymbol{x} \\
&\leq \left[ \int_D p(\boldsymbol{x})^2 \, d\boldsymbol{x} \right]^{1/2} \left[ \int_D f(\boldsymbol{x})^2 \, d\boldsymbol{x} \right]^{1/2}
\end{aligned}$$

by the Cauchy–Schwarz inequality (A.1.5); dividing by $\left[\int_D p(\boldsymbol{x})^2 \, d\boldsymbol{x}\right]^{1/2}$ and squaring gives

$$\int_D p(\boldsymbol{x})^2 \, d\boldsymbol{x} \leq \int_D f(\boldsymbol{x})^2 \, d\boldsymbol{x}.$$

In fact, $p$ is the orthogonal projection of $f$ onto the approximation space $\mathcal{P}$ with respect to the inner product $(f,\ g) = \int_D f(\boldsymbol{x})\, g(\boldsymbol{x})\, d\boldsymbol{x}$.

As with the case of the normal equations for finite sums, we can solve (4.7.1) using LU or Cholesky factorization.

Often the symbolic evaluation of the matrix and right-hand side of (4.7.1) is not possible. In such cases, we fall back on numerical evaluation of the integrals. Care must be taken with such an approach. If we use an integration method

$$\int_D \psi(\boldsymbol{x})\, d\boldsymbol{x} \approx \sum_{j=1}^{M} w_j\, \psi(\boldsymbol{x}_j)$$

for approximating the integrals, then Equation (4.7.1) is replaced by

$$\left[\sum_{j=1}^{M} w_j\, \boldsymbol{\phi}(\boldsymbol{x}_j)\boldsymbol{\phi}(\boldsymbol{x}_j)^T\right] \boldsymbol{c} = \sum_{j=1}^{M} w_j\, \boldsymbol{\phi}(\boldsymbol{x}_j)\, f(\boldsymbol{x}_j).$$

Pre-multiplying by $\boldsymbol{c}^T$ gives

$$\begin{aligned}\sum_{j=1}^{M} w_j(\boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}_j))^2 &= \sum_{j=1}^{M} w_j\, \boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}_j)\, f(\boldsymbol{x}_j)\\ &\le \left[\sum_{j=1}^{M} w_j\, (\boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}_j))^2\right]^{1/2}\left[\sum_{j=1}^{M} w_j\, f(\boldsymbol{x}_j)^2\right]^{1/2}\end{aligned}$$

by the Cauchy–Schwarz inequality (A.1.4) assuming that all weights $w_j > 0$. Dividing by $\left[\sum_{j=1}^{M} w_j\, (\boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}_j))^2\right]^{1/2}$ and squaring gives

$$\sum_{j=1}^{M} w_j(\boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}_j))^2 \le \sum_{j=1}^{M} w_j\, f(\boldsymbol{x}_j)^2.$$

Since $\boldsymbol{c}^T\boldsymbol{\phi}(\boldsymbol{x}) = p(\boldsymbol{x})$, this gives $\sum_{j=1}^{M} w_j\, p(\boldsymbol{x}_j)^2 \le \sum_{j=1}^{M} w_j\, f(\boldsymbol{x}_j)^2$. Treating $\sum_{j=1}^{M} w_j\, f(\boldsymbol{x}_j)^2$ as a discrete approximation to $\int_D f(\boldsymbol{x})^2\, d\boldsymbol{x}$, we can see that this gives a bound on the approximation $p(\boldsymbol{x})$, regardless of how well- or ill-conditioned the matrix $B$ is. However, this result depends on choosing the integration points $\boldsymbol{x}_j$ to be the same for both computing $\sum_{j=1}^{M} w_j\, \boldsymbol{\phi}(\boldsymbol{x}_j)\boldsymbol{\phi}(\boldsymbol{x}_j)^T$ and for computing the right-hand side $\sum_{j=1}^{M} w_j\, \boldsymbol{\phi}(\boldsymbol{x}_j)\, f(\boldsymbol{x}_j)$.

A variation on this approach is to use *weighted least squares*, where a weighting function $w\colon D \to \mathbb{R}$ is used: $w(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in D$, and $\int_D w(\boldsymbol{x})\, d\boldsymbol{x}$ is finite. Then we seek $p$ in our approximation space that minimizes

$$\int_D w(\boldsymbol{x})\,(f(\boldsymbol{x}) - p(\boldsymbol{x}))^2\,d\boldsymbol{x}. \tag{4.7.3}$$

The normal equations for weighted least squares approximation problem are then

$$\left[\int_D w(\boldsymbol{x})\,\boldsymbol{\phi}(\boldsymbol{x})\,\boldsymbol{\phi}(\boldsymbol{x})^T\,d\boldsymbol{x}\right]\boldsymbol{c} = \int_D f(\boldsymbol{x})\,w(\boldsymbol{x})\,\boldsymbol{\phi}(\boldsymbol{x})\,d\boldsymbol{x}. \tag{4.7.4}$$

The matrix for this system of equations is, again, symmetric and positive semi-definite; positive definite if $\{\phi_1, \phi_2, \ldots, \phi_N\}$ are linearly independent over $D$. The same methods for solving the weighted normal equations can be used as for the standard normal equations. Numerical integration methods can also be used:

$$\int_D w(\boldsymbol{x})\,\psi(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{j=1}^{M} w_j\,\psi(\boldsymbol{x}_j),$$

to give fully discrete methods for finding weighted least squares approximations.

### *4.7.2 Orthogonal Polynomials*

For the weighted least squares problem of minimizing

$$\int_D w(\boldsymbol{x})\,(f(\boldsymbol{x}) - p(\boldsymbol{x}))^2\,d\boldsymbol{x}$$

over $p \in \mathcal{P}$ it can be helpful to define the weighted inner product

$$(f,\ g)_w = \int_D w(\boldsymbol{x})\,f(\boldsymbol{x})\,g(\boldsymbol{x})\,d\boldsymbol{x}$$

for a positive weight function $w(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in D$ except on a set of zero volume. The weighted least squares problem can then be expressed as

$$\min_{p\in\mathcal{P}}(f - p,\ f - p)_w.$$

We consider the one-variable case:

$$(f,\ g)_w = \int_a^b w(x)\,f(x)\,g(x)\,dx.$$

If $\{\phi_1, \phi_2, \ldots, \phi_N\}$ is a basis for $\mathcal{P}$, then we can express the weighted normal equations (4.7.4) as

$$\sum_{j=1}^{N} (\phi_i,\ \phi_j)_w\, c_j = (f,\ \phi_i)_w, \qquad i = 1, 2, \ldots, N, \tag{4.7.5}$$

for the coefficients $c_j$ where $p(\boldsymbol{x}) = \sum_{j=1}^{N} c_j\, \phi_j(\boldsymbol{x})$. These equations are much easier to solve if the matrix $b_{ij} = (\phi_i,\ \phi_j)_w$ is a diagonal matrix, that is, if $(\phi_i,\ \phi_j)_w = 0$ for $i \neq j$. In that case, $c_i = (f,\ \phi_i)_w / (\phi_i,\ \phi_i)_w$.

Orthogonal basis functions are also likely to give better conditioned bases than other bases, especially if they are scaled so that $(\phi_i,\ \phi_i)_w = 1$ for all $i$. If the approximation space $\mathcal{P}$ is a space of polynomials, then our orthogonal basis functions are orthogonal polynomials. In fact, for polynomials of one variable, there is additional algebraic structure that we can use. In fact, we say $p_0, p_1, p_2, \ldots$ is a family of *orthogonal polynomials* with respect to the inner product $(\cdot,\ \cdot)_w$ if

- $(p_i,\ p_j)_w = 0$ if $i \neq j$, and
- $\deg p_j = j$ for $j = 0, 1, 2, \ldots$.

Note that scaling each $p_j$ by a non-zero scale factor $\alpha_j$ to give $\alpha_j p_j$ does not change these properties. So we can set $p_0(x) = 1$, or any other non-zero constant. Suppose we have created $p_j$ for $j = 0, 1, 2, \ldots, k$ and we want to get $p_{k+1}$. We can start with $u_{k+1}(x) = x^{k+1}$ and apply the Gram–Schmidt process (Algorithm 14):

$$q_{k+1} = u_{k+1} - \sum_{j=0}^{k} \frac{(u_{k+1},\ p_j)_w}{(p_j,\ p_j)_w} p_j.$$

Since the degree of $p_j$ is $j < k+1$ for $j = 1, 2, \ldots, k$, $q_{k+1}$ is a polynomial with leading term $x^{k+1}$ and so cannot be zero. We can then set $p_{k+1} = \alpha_{k+1} q_{k+1}$ for some non-zero $\alpha_{k+1}$ chosen according to some other objective.

We can define the following operation on functions: $Xf(x) = x\, f(x)$. Then

$$(Xf,\ g)_w = \int_a^b w(x)\, x\, f(x)\, g(x)\, dx = (f,\ Xg)_w. \tag{4.7.6}$$

We will now show that $(Xp_k,\ p_j)_w = 0$ for $j = 0, 1, 2, \ldots, k-2$. First, note that since $\deg p_j = j$, the polynomials $\{p_0, p_1, \ldots, p_{k-1}\}$ form a basis for all polynomials of degree $\leq k-1$. Since $(p_k,\ p_j)_w = 0$ for all $j < k$, it follows that $(p_k,\ q)_w = 0$ for all polynomials $q$ with $\deg q \leq k-1$. Now $(Xp_k,\ p_j) = (p_k,\ Xp_j)$ by (4.7.6), but if $j \leq k-2$ then $\deg Xp_j = j+1 \leq k-1$ so $(p_k,\ Xp_j)_w = 0$. So if we apply the Gram–Schmidt process to $Xp_k$ we get

$$\begin{aligned} q_{k+1} &= Xp_k - \sum_{j=0}^{k} \frac{(Xp_k,\ p_j)_w}{(p_j,\ p_j)_w} p_j \\ &= Xp_k - \frac{(Xp_k,\ p_k)_w}{(p_k,\ p_k)_w} p_k - \frac{(Xp_k,\ p_{k-1})_w}{(p_{k-1},\ p_{k-1})_w} p_{k-1}. \end{aligned}$$

**Table 4.7.1** Legendre polynomials

| $k$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| $P_k(x)$ | 1 | $x$ | $\frac{1}{2}(3x^2-1)$ | $\frac{1}{2}(5x^3-3x)$ | $\frac{1}{8}(35x^4-30x^2+3)$ | $\frac{1}{8}(63x^5-70x^3+15x)$ |

Setting $p_{k+1} = \alpha_k q_{k+1}$ now gives

$$p_{k+1} = \alpha_k q_{k+1} = \alpha_k X p_k - \beta_k p_k - \gamma_k p_{k-1} \tag{4.7.7}$$

for suitable constants $\beta_k$ and $\gamma_k$. This can be re-written as

$$p_{k+1}(x) = (\alpha_k x - \beta_k) p_k(x) - \gamma_k p_{k-1}(x), \tag{4.7.8}$$

which is the general form of a three-term recurrence relation for orthogonal polynomials. An example of a three-term recurrence relation is the one for Chebyshev polynomials (4.6.4). These recurrence relations give efficient ways of computing $p_j(x)$ for $j = 0, 1, 2, \ldots$ once the values of $\alpha_j$, $\beta_j$ and $\gamma_j$ are known for $j = 0, 1, 2, \ldots$.

#### 4.7.2.1 Legendre Polynomials

An important example of orthogonal polynomials is the orthogonal polynomials for $(f,\ g)_w = \int_{-1}^{+1} f(x)\, g(x)\, dx$. Up to a scale factor, these orthogonal polynomials are the *Legendre polynomials*, which we can write using a *Rodriguez formula*:

$$P_n(x) = \frac{1}{2^n\, n!} \frac{d^n}{dx^n} \left[ \left(x^2 - 1\right)^n \right]. \tag{4.7.9}$$

The first few Legendre polynomials are shown in Table 4.7.1.

Some other examples of orthogonal polynomials are Chebyshev polynomials with the weight function $w(x) = 1/\sqrt{1-x^2}$ over the interval $(-1,\ +1)$, *Laguerre polynomials* with weight function $w(x) = e^{-x}$ over the interval $[0, \infty)$, and *Hermite polynomials* with weight function $w(x) = e^{-x^2}$ over the interval $(-\infty, +\infty)$. Rodrigues formulas for Laguerre and Hermite polynomials are

$$L_n(x) = \frac{e^x}{n!} \frac{d^n}{dx^n}(e^{-x} x^n), \qquad \text{and}$$
$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n}(e^{-x^2}).$$

### *4.7.3 Trigonometric Polynomials and Fourier Series*

Approximation of periodic functions $f$ by trigonometric polynomials

$$f(x) \approx a_0 + \sum_{k=1}^{N} (a_k \cos(kx) + b_k \sin(kx))$$

is a classical topic in analysis, starting with Fourier's *Analytical Theory of Heat* [93] (1822) with later work by Lagrange, Dirichlet, Fejér, Calderón, Zygmund, and many others. The standard approach is to minimize the error squared:

$$\min_{p_N \in \mathcal{P}_N} \int_0^{2\pi} (f(x) - p_N(x))^2 \, dx,$$

where $\mathcal{P}_N = \text{span}\{ x \mapsto \cos(kx),\ x \mapsto \sin(kx) \mid k = 0, 1, 2, \ldots, N \}$. Since

$$\int_0^{2\pi} \cos(kx) \cos(\ell x) \, dx = \int_0^{2\pi} \sin(kx) \sin(\ell x) \, dx = 0 \qquad \text{for all } k \neq \ell,$$
$$\text{and} \quad \int_0^{2\pi} \cos(kx) \sin(\ell x) \, dx = 0 \qquad \text{for all } k,\ \ell,$$

we have an orthogonal basis, from which we can easily compute the coefficients

$$a_0 = (2\pi)^{-1} \int_0^{2\pi} f(x) \, dx,$$
$$a_k = \pi^{-1} \int_0^{2\pi} f(x) \cos(kx) \, dx, \qquad \text{for } k > 0, \text{ and}$$
$$b_k = \pi^{-1} \int_0^{2\pi} f(x) \sin(kx) \, dx, \qquad \text{for } k > 0.$$

However, simply using this least squares approximation results in significant overshoot for jump discontinuities and does not guarantee convergence of $p_N$ to $f$ uniformly for continuous $f$. Fejér proved [91] that if

$$\widetilde{p}_N(x) = a_0 + \sum_{k=1}^{N} \left(1 - \frac{k}{N+1}\right) (a_k \cos(kx) + b_k \sin(kx))$$

then $\widetilde{p}_N \to f$ uniformly as $N \to \infty$ for any continuous $f$. This approach was extended by D. Jackson [133] to give Jackson's theorem (see Section 4.5.2).

Trigonometric polynomials can also be thought of in terms of complex exponentials. Since

$$e^{i\theta} = \cos\theta + i \sin\theta \qquad (i = \sqrt{-1})$$

we can also write

$$\cos\theta = \frac{1}{2}(e^{i\theta} + e^{-i\theta}) = \text{Re}\, e^{i\theta} \text{ and } \sin\theta = \frac{1}{2i}(e^{i\theta} - e^{-i\theta}) = \text{Im}\, e^{i\theta},$$

where $\operatorname{Re} z$ and $\operatorname{Im} z$ are the real and imaginary parts of a complex number $z$. Thus trigonometric polynomials of order $N$ can be written in the form $p_N(x) = c_0 + \sum_{k=-N}^{+N} c_k\, e^{ikx}$. Note that $e^{-ikx} = \overline{e^{ikx}}$ taking complex conjugates.

#### 4.7.3.1 Trigonometric Interpolation and the Discrete Fourier Transform

Interpolation with trigonometric polynomials on equally spaced points is a natural task: if $x_k = 2\pi k/N$ for $k = 0, 1, 2, \ldots, N-1$ and $y_k$ are given values, then we seek $c_j$ where

$$y_k = \sum_{j=0}^{N-1} c_j\, e^{2\pi i\, jk/N} \qquad \text{for } k = 0, 1, 2, \ldots, N-1. \tag{4.7.10}$$

Note that since $j$ and $k$ are integers, $e^{2\pi i\, j(N-k)/N} = e^{-2\pi i\, jk} = \overline{e^{2\pi i\, jk}}$.

The transformation $\boldsymbol{c} \mapsto \boldsymbol{y}$ defined by $y_k = \sum_{j=0}^{N-1} c_j\, e^{2\pi i\, jk/N}$ is called the discrete Fourier transform (DFT). It is clearly a linear transformation $\mathbb{C}^N \to \mathbb{C}^N$. And it is invertible:

$$\begin{aligned} \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N} y_k &= \frac{1}{N} \sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N} \sum_{j=0}^{N-1} c_j\, e^{2\pi i\, jk/N} \\ &= \frac{1}{N} \sum_{j=0}^{N-1} c_j \sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N}\, e^{2\pi i\, jk/N}. \end{aligned} \tag{4.7.11}$$

But

$$\sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N}\, e^{2\pi i\, jk/N} = \sum_{k=0}^{N-1} e^{2\pi i\, (j-\ell)k/N} = \sum_{k=0}^{N-1} \left(e^{2\pi i\, (j-\ell)/N}\right)^k.$$

If $e^{2\pi i\, (j-\ell)/N} \neq 1$ we can use the standard formula for summing a finite geometric series to get

$$\sum_{k=0}^{N-1} \left(e^{2\pi i\, (j-\ell)/N}\right)^k = \frac{\left(e^{2\pi i\, (j-\ell)/N}\right)^N - 1}{e^{2\pi i\, (j-\ell)/N} - 1} = \frac{e^{2\pi i\, (j-\ell)} - 1}{e^{2\pi i\, (j-\ell)/N} - 1} = 0$$

since $2\pi(j-\ell)$ is an integer multiple of $2\pi$. Thus, for $0 \le j, \ell \le N-1$, $j \neq \ell$ implies $e^{2\pi i\, (j-\ell)/N} \neq 1$ and so $\sum_{k=0}^{N-1} e^{2\pi i\, (j-\ell)k/N} = 0$. So the only non-zero term in the outer sum of (4.7.11) is the term where $j = \ell$. So

$$\frac{1}{N}\sum_{j=0}^{N-1} c_j \sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N}\, e^{2\pi i\, jk/N} = \frac{1}{N} c_\ell \sum_{k=0}^{N-1} 1 = \frac{1}{N} c_\ell N = c_\ell.$$

That is,

$$c_\ell = \frac{1}{N}\sum_{k=0}^{N-1} e^{-2\pi i\, k\ell/N}\, y_k, \tag{4.7.12}$$

which is the inverse discrete Fourier transform.

#### 4.7.3.2 The Fast Fourier Transform

The obvious way of computing a discrete Fourier transform (DFT) (4.7.10) or inverse discrete Fourier transform (4.7.12) takes $\mathcal{O}(N^2)$ floating point operations. The hidden constant is more than two since complex addition requires two real additions, while complex multiplication takes four real multiplications and two real additions. However, it was discovered by Cooley and Tukey [56] (1965) that there is a general and fast way of carrying out a DFT on $N$ data points in $\mathcal{O}(N \log N)$ operations. This work was based on earlier work by Danielson and Lanczos [68] (1942), whose approach was pre-figured by unpublished work by C.F. Gauss. A similar method had been published by Yates [264] for some different transforms.

The basic idea of the fast Fourier transform (FFT) can be seen best where $N$ is a power of two, although other prime factors of $N$ were used by Cooley and Tukey. To see how this works, we first consider the case where $N$ is even and write $N = 2M$. Consider

$$y_k = \sum_{j=0}^{N-1} c_j\, e^{2\pi i\, jk/N} = \sum_{j=0}^{2M-1} c_j\, e^{2\pi i\, jk/(2M)}.$$

We split the sum into two parts: even $j = 2\ell$ and odd $j = 2\ell + 1$. Then

$$\begin{aligned} y_k &= \sum_{j=0}^{2M-1} c_j\, e^{2\pi i\, jk/(2M)} \\ &= \sum_{\ell=0}^{M-1} \left(c_{2\ell}\, e^{2\pi i\, 2\ell k/(2M)} + c_{2\ell+1}\, e^{2\pi i\, (2\ell+1)k/(2M)}\right) \\ &= \sum_{\ell=0}^{M-1} c_{2\ell}\, e^{2\pi i\, \ell k/M} + e^{2\pi i\, k/(2M)} \sum_{\ell=0}^{M-1} c_{2\ell+1}\, e^{2\pi i\, \ell k/M}. \end{aligned}$$

The first sum is the DFT of $(c_0, c_2, \ldots, c_{2(M-1)})$ while the second sum is the DFT of $(c_1, c_3, \ldots, c_{2M-1})$. We can also split the computation of $y_k$ into the cases where $0 \leq$

**Algorithm 58** Fast Fourier transform for $N = 2^m$, decimation-in-time version

```
1  function fft(c)
2    N ← length(c); M ← N/2
3    a ← [c_{2ℓ} | ℓ = 0, 1, ..., M − 1]; b ← [c_{2ℓ+1} | ℓ = 0, 1, ..., M − 1]
4    if M > 1
5      u ← fft(a); v ← fft(b)
6    else
7      u ← a; v ← b
8    end
9    for k = 0, 1, ..., M − 1
10     y_k     ← u_k + e^{2πi k/(2M)} v_k
11     y_{k+M} ← u_k − e^{2πi k/(2M)} v_k
12   end
13   return y
14 end function
```

$k \le M-1$ and $M \le k \le 2M-1$. Note that for $0 \le k \le M-1$, $e^{2\pi i\,\ell(k+M)/M} = e^{2\pi i\,\ell k/M}$. If we write the DFT of $(c_0, c_2, \ldots, c_{2(M-1)})$ as $(u_0, u_1, \ldots, u_{M-1})$ and the DFT of $(c_1, c_3, \ldots, c_{2M-1})$ as $(v_0, v_1, \ldots, v_{M-1})$ we have for $k = 0, 1, 2\ldots, M-1$:

$$y_k = u_k + e^{2\pi i\,k/(2M)} v_k, \tag{4.7.13}$$

$$y_{k+M} = u_k - e^{2\pi i\,k/(2M)} v_k. \tag{4.7.14}$$

The operations (4.7.13), (4.7.14) were called a *butterfly* by Tukey. This gives a recursive algorithm for computing FFTs as shown in Algorithm 58.

Note that Algorithm 58 is labeled as a "decimation in time" version. That is because the $c_k$'s are split into two vectors depending on whether the index $k$ is even or odd. There are versions where the split into even and odd indexes does not occur for the $c_k$'s but for the $y_k$'s. These versions are called "decimation in frequency" versions. As noted above, we can also create variants of the FFT algorithm to handle other prime factors of $N$, allowing for general FFT algorithms.

Inverse Fourier transforms can be computed by a variant of the FFT where the factors $e^{2\pi i\,k/(2M)}$ are replaced by the conjugates $e^{-2\pi i\,k/(2M)}$, and by dividing the result by $N$ at the end of the computation.

#### 4.7.3.3 Lebesgue Numbers and Error Estimates

The minimax approximation error obtained by trigonometric polynomials of order $\le N$ is given by the Jackson theorems to be $\mathcal{O}(N^{-m})$ if $f$ is continuously differentiable $m$ times (see Section 4.5.2).

On the other hand, by estimating the Lebesgue numbers (see Section 4.1.2) for interpolation with trigonometric polynomials of order $\le N$ we can obtain bounds on the interpolation error. The Lagrange interpolation functions for equally spaced trigonometric polynomial interpolation on $[0, 2\pi]$ with $x_j = 2\pi j/N$,

$j = 0, 1, 2, \ldots, N-1$, can be shown to be

$$L_j(x) = \frac{\sin(N(x - x_j)/2)}{N \sin((x - x_j)/2)}, \qquad j = 0, 1, 2, \ldots, N.$$

The Lebesgue function $\sum_{j=0}^{N-1} \left|L_j(x)\right|$ can be bounded asymptotically by $(2/\pi) \ln N$ as $N \to \infty$. Because of the slow growth of the Lebesgue numbers for equally spaced trigonometric polynomial interpolation, the interpolation error tends to be very good. Only for very rough functions do we expect the interpolation error to increase with $N$.

### 4.7.4 Chebyshev Expansions

Chebyshev polynomials are given by $T_n(\cos\theta) = \cos(n\theta)$ for $n = 0, 1, 2, \ldots$. This makes Chebyshev polynomials orthogonal with respect to the weight function $w(x) = 1/\sqrt{1 - x^2}$ over $(-1, +1)$:

$$\begin{aligned}\int_{-1}^{+1} \frac{T_m(x)\, T_n(x)}{\sqrt{1 - x^2}} dx &= \int_0^{\pi} \frac{T_m(\cos\theta)\, T_n(\cos\theta)}{\sqrt{1 - \cos^2\theta}} \sin\theta\, d\theta \\ &= \int_0^{\pi} \cos(m\theta)\, \cos(n\theta)\, d\theta \\ &= \begin{cases} \pi, & \text{if } m = n = 0, \\ \pi/2, & \text{if } m = n \neq 0, \\ 0, & \text{otherwise.} \end{cases}\end{aligned}$$

So for a given function $f : [-1, +1] \to \mathbb{R}$ we have the Fourier series expansion

$$f(\cos\theta) = \sum_{k=0}^{\infty} a_k \cos(k\theta), \qquad \text{and we get}$$

$$f(x) = \sum_{k=0}^{\infty} a_k\, T_k(x).$$

For $f$ differentiable an arbitrary number of times, the coefficients $a_k$ go to zero asymptotically faster than any rational function. Since $|T_k(x)| \leq 1$ for all $k$ and $-1 \leq x \leq +1$, the *Chebyshev expansion* $f(x) = \sum_{k=0}^{\infty} a_k\, T_k(x)$ converges uniformly and usually rapidly.

Instead of using integrals to compute the coefficients $a_k$, we can use trigonometric interpolation on equally spaced points. This is equivalent to using the discrete Fourier transform to compute estimates of Fourier series coefficients. Applied to the function $f(x) = 1/(1 + (5x)^2)$ on $[-1, +1]$ (see the Runge phenomenon in Section 4.1.1.13),

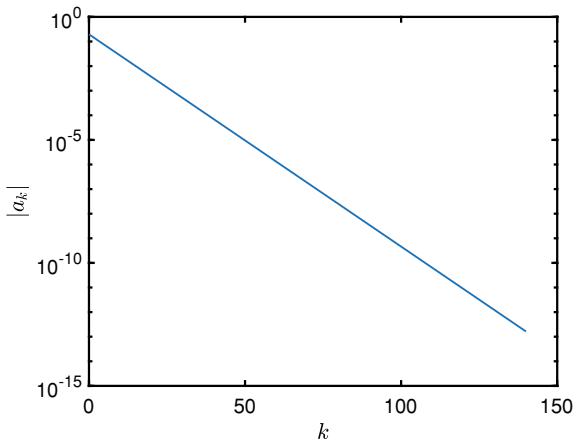



**Fig. 4.7.1** Chebyshev expansion coefficients for $f(x) = 1/(1 + (5x)^2)$ (even coefficients only, odd coefficients are zero)

we obtain Chebyshev coefficients. These coefficients decay exponentially in $k$, as we can see in Figure 4.7.1.

**Exercises.**

(1) Show that the three-term recurrence (4.7.8) for a family of orthogonal polynomials can be re-written as

$$x\, p_k(x) = c_k\, p_{k-1}(x) + a_k\, p_k(x) + b_k\, p_{k+1}(x). \tag{4.7.15}$$

Further, show that if the orthogonal polynomials are normalized (that is, $(p_j,\ p_j)_w = 1$ for all $j$), then $c_k = b_{k-1}$.

(2) Show that if the three-term recurrence (4.7.15) holds with $c_k = b_{k-1}$, then the eigenvalues of

$$T_n = \begin{bmatrix} a_0 & b_0 & & & \\ b_0 & a_1 & b_1 & & \\ & b_1 & a_2 & \ddots & \\ & & \ddots & \ddots & b_{n-1} \\ & & & b_{n-1} & a_n \end{bmatrix}$$

are the zeros of $p_{n+1}$.

(3) Use the Rodriguez formula (4.7.9) for the Legendre polynomials to show that they form a family of orthogonal polynomials with respect to the inner product $(f, g) = \int_{-1}^{+1} f(x)\, g(x)\, dx$. [**Hint:** Use integration by parts, *many* times!]

(4) Show that the coefficients $a_k$ of a Chebyshev expansion $f(x) = \sum_{k=0}^{\infty} a_k\, T_k(x)$ are given by $a_k = (1/\pi) \int_0^{2\pi} f(\cos\theta)\, \cos(k\theta)\, d\theta$ for $k = 1, 2, \ldots$ and $a_0 = (1/(2\pi)) \int_0^{2\pi} f(\cos\theta)\, d\theta$.

(5) If $f(x) = \sum_{k=0}^{\infty} a_k\, T_k(x) \approx \sum_{k=0}^{N-1} a_k T_k(x)$ show that we can estimate the coefficients $a_k$ by $\widehat{a}_k$ through the equations $f(\cos(j\pi/N)) = \sum_{k=0}^{N-1} \widehat{a}_k \cos(kl\pi/N)$. Show how we can compute the $\widehat{a}_k$ through a discrete Fourier transform of the data $f(\cos(j\pi/N))$, $j = 0, 1, 2, \ldots, 2N-1$.
(6) Implement a function to evaluate $\sum_{k=0}^{N-1} a_k\, T_k(x)$ that takes $\mathcal{O}(N)$ floating point operations. [**Hint:** Use the Chebyshev three-term recurrence relation.]
(7) Plot the size of the coefficients $|a_k|$ of the Chebyshev expansion $f(x) = \sum_{k=0}^{\infty} a_k\, T_k(x)$ for $-1 \le x \le +1$ where $f(x) = e^x$. Use a logarithmic scale for $|a_k|$. Show that empirically we have $|a_k| \sim C\, r^k$ for $k$ large. Estimate the value of $r$ from the empirical data.
(8) Repeat the previous Exercise with $f(x) = 1/(1 + (5x)^2)$.
(9) Show that if $m$ divides $N$ then a DFT of order $N$ can be created by combining $m$ DFT's or order $N/m$ just as the Cooley–Tukey creates a DFT of order $N = 2^n$ by combining two DFTs of order $N/2 = 2^{n-1}$.
(10) Compute orthogonal polynomials of degree $\le 4$ for the inner product

$$(f, g) = -\int_0^1 \ln(x)\, f(x)\, g(x)\, dx.$$

(11) Let $\mathrm{DFT}(\boldsymbol{f})_k = \sum_{j=0}^{N-1} e^{-2\pi i jk/N} f_j$ be the discrete Fourier transform applied to $\boldsymbol{f}$. Show that $\mathrm{DFT}(\boldsymbol{f}) \circ \mathrm{DFT}(\boldsymbol{g}) = \mathrm{DFT}(\boldsymbol{f} * \boldsymbol{g})$ where "$\circ$" is the Hadamard or componentwise product ($(\boldsymbol{u} \circ \boldsymbol{v})_k = u_k v_k$) and "$*$" is the cyclic discrete convolution ($(\boldsymbol{f} * \boldsymbol{g})_k = \sum_{j=0}^{N-1} f_j g_{k-j \bmod N}$).

## Project

Using a linear program solver, create software to compute the minimax polynomial approximation $p(x, y)$ of degree $\le m$ for a given function $f(x, y)$ over the standard triangle $\widehat{K} = \{ (x, y) \mid 0 \le x, y \,\&\, x + y \le 1 \}$. In order to approximate the local maxima of $|f(x, y) - p(x, y)|$ over $(x, y) \in \widehat{K}$, evaluate $|f(x, y) - p(x, y)|$ at many points in $\widehat{K}$. Test this method by applying it to $f(x, y) = e^{x+y} \cos(x - \frac{1}{2}y)\,(1 + y^2)$ and $m = 1, 2, 3, 4, 5$.

Avoid adding constraints for each of the many points used for evaluating $|f(x, y) - p(x, y)|$. This can greatly add to the cost of solving the linear program. Instead, add constraints

$$\pm(f(x, y) - p(x, y)) \le s$$

for points $(x, y)$ only when they are apparently relevant. Write $p(x, y) = \sum_{j=0}^{m-1} c_j \phi_j(x, y)$ with basis functions $\phi_j$, $j = 0, 1, 2, \ldots, m-1$, making the $c_j$'s the main unknowns in the linear program. The algorithm should maintain a set $\mathcal{S} = \{ (x_k, y_k) \mid k = 1, 2, \ldots, n \} \subset \widehat{K}$. At each iteration, the method solves the linear program

$$\min_{s,\boldsymbol{c}} s \qquad \text{subject to}$$

$$+ (f(x_k, y_k) - \sum_{j=0}^{m-1} c_j \phi_j(x_k, y_k)) \le s, \qquad k = 1, 2, \ldots, n,$$

$$- (f(x_k, y_k) - \sum_{j=0}^{m-1} c_j \phi_j(x_k, y_k)) \le s, \qquad k = 1, 2, \ldots, n.$$

Initialize $\mathcal{S}$ to be an interpolation set and add points to $\mathcal{S}$ according to the values $|f(x, y) - p(x, y)|$ over $(x, y) \in \widehat{K}$.

# Chapter 5
# Integration and Differentiation


## 5.1 Integration via Interpolation

For functions of one, two, or three variables, numerical integration is often done via interpolation. Error estimates for interpolation can be used to obtain error estimates for integration. In high dimensions, these approaches lose value as the amount of data needed to obtain a reasonable interpolant becomes exorbitant. But in low dimensions, and especially in dimension one, these approaches work very well.

### *5.1.1 Rectangle, Trapezoidal and Simpson's Rules*

Geometric intuition is useful in defining integrals as limits of areas of rectangles approximating the graph of the function to be integrated, as illustrated in Figure 5.1.1.

The *rectangle rule* for estimating the integral, using the left-hand endpoint of each rectangle, is

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} f(x_i)\,(x_{i+1} - x_i). \tag{5.1.1}$$

The *trapezoidal rule* is

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \frac{1}{2}\left[f(x_i) + f(x_{i+1})\right]\,(x_{i+1} - x_i). \tag{5.1.2}$$

If, instead of evaluating the function at the left endpoint of each rectangle, we have the *mid-point rule*:



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_5

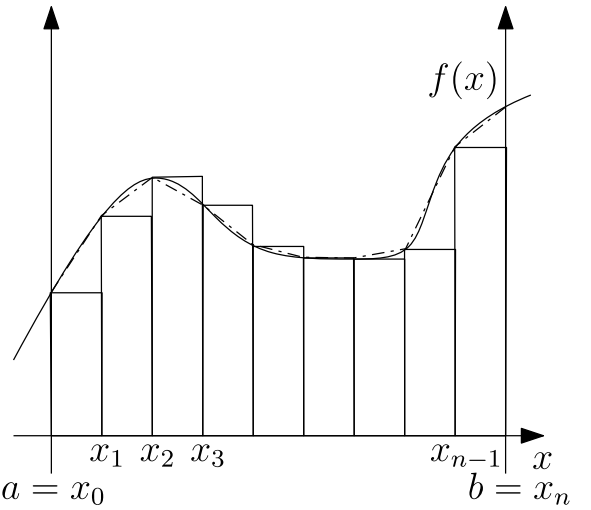



**Fig. 5.1.1** Approximate integration via rectangles and trapezoids; rectangles in solid lines, trapezoids in dot-dashed lines

**Algorithm 59** Rectangle rule for integration

```
1  function rectangle(f, a, b, n)
2    s ← 0
3    for i = 0, 1, 2, ..., n − 1
4      s ← s + f(x_i)
5    end for
6    return s · (b − a)/n
7  end function
```

**Algorithm 60** Trapezoidal rule for integration

```
1  function trapezoidal(f, a, b, n)
2    s ← (f(a) + f(b))/2;  h ← (b − a)/n
3    for i = 1, 2, ..., n − 1
4      s ← s + f(a + i h)
5    end for
6    return s · (b − a)/n
7  end function
```

**Algorithm 61** Mid-point rule for integration

```
1  function midpoint(f, a, b, n)
2    s ← 0;  h ← (b − a)/n
3    for i = 0, 1, 2, ..., n − 1
4      s ← s + f(a + (i + 1/2)h)
5    end for
6    return s · (b − a)/n
7  end function
```

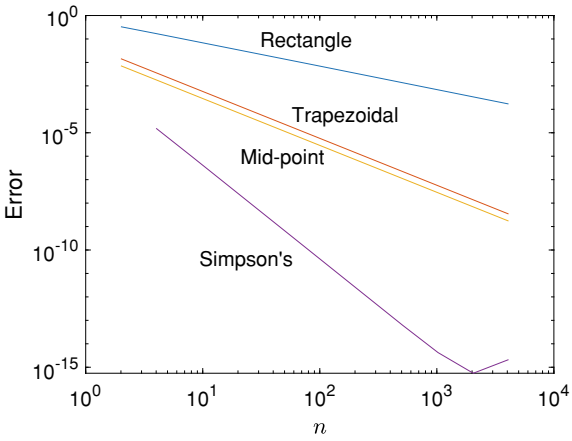



**Fig. 5.1.2** Comparison of basic methods for integrals

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n-1} \frac{1}{2} f\left(\frac{x_i + x_{i+1}}{2}\right)(x_{i+1} - x_i). \tag{5.1.3}$$

Pseudo-code for these methods is shown in Algorithms 59, 60, and 61.

The error for each of these methods applied to $\int_1^2 x \ln x\,dx$ is shown in Figure 5.1.2. From these empirical results, we can estimate the slopes of the error curves on the log–log plot to estimate the exponent $\alpha$ in error $\sim$ const $h^\alpha$: for the rectangle rule, $\alpha \approx 1$, while for the trapezoidal and mid-point rules, $\alpha \approx 2$, and for Simpson's rule (to be discussed later) we obtain $\alpha \approx 4$ if we avoid the bend near the end of the graph due to roundoff error.

### 5.1.1.1 Error Analysis

Each of the rectangle, trapezoidal, and mid-point rules can be analyzed by using polynomial interpolation theory. The rectangle and mid-point rules use piecewise constant interpolation, while the trapezoidal rule uses piecewise linear interpolation.

*Rectangle rule.* The simplest of these is the rectangle rule. Using the interpolation error formula (4.1.7),

$$f(x) - f(x_i) = f'(c_{x,i})(x - x_i) \qquad \text{with } x_i \le c_{x,i} \le x,$$

we can estimate the integration error $e_i$ on one piece $[x_i, x_{i+1}]$:

$$
\begin{aligned}
e_i &= \int_{x_i}^{x_{i+1}} f(x)\,dx - f(x_i)(x_{i+1} - x_i) \\
&= \int_{x_i}^{x_{i+1}} [f(x) - f(x_i)]\,dx = \int_{x_i}^{x_{i+1}} f'(c_{x,i})(x - x_i)\,dx \\
&= f'(c_i) \int_{x_i}^{x_{i+1}} (x - x_i)\,dx = f'(c_i)\frac{1}{2}(x_{i+1} - x_i)^2
\end{aligned}
$$

for some $c_i \in [x_i, x_{i+1}]$ by the Generalized Mean Value Theorem, since $x - x_i$ does not change sign over $(x_i, x_{i+1})$. Thus the total error is

$$
\begin{aligned}
e &= \sum_{i=0}^{n-1} e_i = \sum_{i=0}^{n-1} f'(c_i)\frac{1}{2}(x_{i+1} - x_i)^2, \qquad \text{so} \\
|e| &\le \sum_{i=0}^{n-1} \left|f'(c_i)\right| \frac{1}{2}(x_{i+1} - x_i)^2 \le \frac{1}{2} \max_i \left(\left|f'(c_i)\right| (x_{i+1} - x_i)\right) \sum_{i=0}^{n-1} (x_{i+1} - x_i) \\
&\le \frac{1}{2} \max_{a \le c \le b} \left|f'(c)\right| \max_i |x_{i+1} - x_i|\ (x_n - x_0).
\end{aligned}
$$

If $|x_{i+1} - x_i| \le h$ for all $i$, we get

$$
|e| \le \frac{1}{2} \max_{a \le c \le b} \left|f'(c)\right|\ h\,(b - a) = \mathcal{O}(h) \qquad \text{as } h \to 0. \tag{5.1.4}
$$

Most often we use equally-spaced evaluation points $x_i$: $x_i = a + i\,h$, with $h = (b - a)/n$. Then asymptotically,

$$
e \sim \frac{1}{2}\left(\int_a^b f'(c)\,dc\right) h = \frac{1}{2}(f(b) - f(a))\,h \qquad \text{as } h \to 0. \tag{5.1.5}
$$

In either case, we write $e = \mathcal{O}(h)$, and say that the rectangle method is first order, confirming our empirical results from Figure 5.1.2.

*Trapezoidal rule.* For the trapezoidal rule, we are using piecewise linear interpolation, and so for $x_i < x < x_{i+1}$ we have the linear interpolant $p_{1,i}(x)$ of $p_{1,i}(x_i) = f(x_i)$ and $p_{1,i}(x_{i+1}) = f(x_{i+1})$. From (4.1.7),

$$
f(x) - p_{1,i}(x) = \frac{f''(c_{x,i})}{2!}(x - x_i)(x - x_{i+1}) \qquad \text{for some } x_i < c_{x,i} < x_{i+1}.
$$

The error for the piece $[x_i, x_{i+1}]$ is

$$e_i = \int_{x_i}^{x_{i+1}} f(x)\,dx - \int_{x_i}^{x_{i+1}} p_{1,i}(x)\,dx = \int_{x_i}^{x_{i+1}} \big(f(x) - p_{1,i}(x)\big)\,dx$$
$$= \int_{x_i}^{x_{i+1}} \frac{f''(c_{x,i})}{2!}(x - x_i)(x - x_{i+1})\,dx.$$

Again $(x - x_i)(x - x_{i+1})$ does not change sign on the interval so by the Generalized Mean Value Theorem,

$$e_i = \frac{f''(c_i)}{2!}\int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\,dx \qquad \text{for some } c_i \in [x_i, x_{i+1}].$$

To evaluate $\int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\,dx$, we can use a change of variable: $x = x_i + s\,h_i$ with $h_i = x_{i+1} - x_i$ and $0 \le s \le 1$. With this change of variable,

$$\int_{x_i}^{x_{i+1}} (x - x_i)(x - x_{i+1})\,dx = \int_0^1 h_i\,s(h_i\,s - h_i)\,h_i\,ds$$
$$= h_i^3 \int_0^1 s(s-1)\,ds = -\frac{1}{6}h_i^3.$$

Thus

$$e_i = -\frac{1}{12} f''(c_i)\,h_i^3.$$

Summing the errors for each piece gives the total error:

$$e = \sum_{i=0}^{n-1} e_i = \sum_{i=0}^{n-1} -\frac{1}{12} f''(c_i)\,h_i^3 = -\frac{1}{12}\sum_{i=0}^{n-1} f''(c_i)\,h_i^3.$$

This enables us to obtain a bound on the total error:

$$|e| \le \frac{1}{12}\sum_{i=0}^{n-1} \big|f''(c_i)\big|\ h_i^3 \le \frac{1}{12}\max_{a\le c\le b}\big|f''(c)\big| \max_i h_i^2 \sum_{i=0}^{n-1} h_i$$
$$= \frac{1}{12}\max_{a\le c\le b}\big|f''(c)\big|\ h^2\,(b-a). \tag{5.1.6}$$

For equally spaced evaluation points ($h_i = h$ for all $i$), we have the asymptotic estimate:

$$e \sim -\frac{1}{12}\int_a^b f''(c)\,dc\,h^2 = \frac{1}{12}(f'(a) - f'(b))\,h^2 \qquad \text{as } h \to 0. \tag{5.1.7}$$

That is, $e = \mathcal{O}(h^2)$ as $h \to 0$.

*Mid-point rule.* A quick short-hand for thinking about this is that degree $m$ interpolation with equal spacing $h$ gives an error of $\mathcal{O}(h^{m+1})$; this error in the interpolant, integrated across an interval of length $b-a$, gives an error in the integral of $\mathcal{O}(h^{m+1})$. This certainly works for the rectangle rule ($m=0$) and the trapezoidal rule ($m=1$). But the mid-point rule uses constant interpolation and yet appears to give an error of $\mathcal{O}(h^2)$. Why?

Using the standard formulas with $p_{0,i}(x) = f((x_i + x_{i+1})/2)$, the constant interpolant on $[x_i, x_{i+1}]$, we get

$$\begin{aligned} e_i &= \int_{x_i}^{x_{i+1}} f(x)\,dx - f(\frac{x_i + x_{i+1}}{2})\,h_i \\ &= \int_{x_i}^{x_{i+1}} \left[ f(x) - f(\frac{x_i + x_{i+1}}{2}) \right] dx \\ &= \int_{x_i}^{x_{i+1}} f'(c_{x,i})(x - \frac{x_i + x_{i+1}}{2})dx. \end{aligned}$$

But we cannot apply the Generalized Mean Value Theorem as $x - (x_i + x_{i+1})/2$ does change sign on $[x_i, x_{i+1}]$. Even more importantly, $\int_{x_i}^{x_{i+1}} (x - (x_i + x_{i+1})/2)\,dx = 0$, so

$$\begin{aligned} \int_{x_i}^{x_{i+1}} f'(c_{x,i})(x - \frac{x_i + x_{i+1}}{2})dx &\approx \int_{x_i}^{x_{i+1}} f'(\frac{x_i + x_{i+1}}{2})(x - \frac{x_i + x_{i+1}}{2})dx \\ &= f'(\frac{x_i + x_{i+1}}{2}) \int_{x_i}^{x_{i+1}} (x - \frac{x_i + x_{i+1}}{2})dx = 0. \end{aligned}$$

Instead of getting errors $e \sim \text{const}\, h$ as expected using piecewise constant interpolants, we get $e \sim \text{const}\, h^2$ as we can see empirically from Figure 5.1.2. The phenomenon of getting asymptotically better convergence rates than we expect from the derivation is called *super-convergence*. The mid-point rule on a single piece $[x_i, x_{i+1}]$ is not only exact for constant functions, as we would expect by using a constant interpolant, but is also exact for linear functions because $f'(x)$ constant implies $e_i = 0$.

Since the mid-point rule is exact for linear functions, we can consider linear interpolants $p_{1,i}(x)$ that interpolate $f$ at $(x_i + x_{i+1})/2$ and one other point in the interval. We can also use the Hermite interpolant (4.1.17) $h_{1,i}$ that interpolates $f$ at $(x_i + x_{i+1})/2$ and $f'$ at the same point. With the Hermite interpolant being exactly integrated, the error for the piece $[x_i, x_{i+1}]$ is

$$\begin{aligned} e_i &= \int_{x_i}^{x_{i+1}} (f(x) - h_{1,i}(x))\,dx \\ &= \int_{x_i}^{x_{i+1}} \frac{f''(c_{x,i})}{2!}(x - \frac{x_i + x_{i+1}}{2})^2 dx. \end{aligned}$$

*Now* we can apply the Generalized Mean Value Theorem to get

$$e_i = \frac{1}{2} f''(c_i) \int_{x_i}^{x_{i+1}} (x - \frac{x_i + x_{i+1}}{2})^2 dx \quad \text{for some } c_i \in [x_i, x_{i+1}].$$

The integral $\int_{x_i}^{x_{i+1}} (x - (x_i + x_{i+1})/2)^2 dx$ can be easily evaluated using a change of variables $x = (x_i + x_{i+1})/2 + s\, h_i$ with $-\frac{1}{2} \le s \le +\frac{1}{2}$:

$$\int_{x_i}^{x_{i+1}} (x - (x_i + x_{i+1})/2)^2 dx = \int_{-1/2}^{+1/2} (s\, h_i)^2\, h_i\, ds = \frac{1}{12} h_i^3.$$

Then

$$e = \sum_{i=1}^{n-1} e_i = \sum_{i=0}^{n-1} \frac{1}{2} f''(c_i) \frac{1}{12} h_i^3 = \frac{1}{24} \sum_{i=0}^{n-1} f''(c_i) h_i^3, \qquad \text{so}$$

$$|e| \le \frac{1}{24} \max_i \left| f''(c_i) \right| \max_i h_i^2 \sum_{i=0}^{n-1} h_i = \frac{1}{24} \max_i \left| f''(c_i) \right| \max_i h_i^2 (b - a). \tag{5.1.8}$$

Asymptotically, for equally spaced evaluation points ($h_i = h$ for all $i$),

$$e \sim \frac{1}{24} \left( \int_a^b f''(c)\, dc \right) h^2 = \frac{1}{24} (f'(b) - f'(a))\, h^2 \qquad \text{as } h \to 0. \tag{5.1.9}$$

### 5.1.1.2 Simpson's Rule

Simpson's rule starts by integrating the piecewise quadratic interpolation of the function $f(x)$. Because quadratic interpolation requires three data points, we use interpolation on pieces $[x_{2i}, x_{2i+2}]$. The interpolation points are $x_0, x_1, x_2, \ldots, x_n$. We need $n$ to be even. We assume equal spacing $h$ between interpolation points. On the piece $[x_{2i}, x_{2i+2}]$, the quadratic interpolant is

$$p_{2,i}(x) = f(x_{2i})\, L_{0,i}(x) + f(x_{2i+1})\, L_{1,i}(x) + f(x_{2i+2})\, L_{2,i}(x)$$

using the appropriate Lagrange interpolation functions (4.1.3). The Lagrange interpolation functions satisfy $L_{j,i}(x_{2i+\ell}) = 1$ if $j = \ell$ and zero if $j \ne \ell$ for $\ell = 0, 1, 2$. We can write $L_{j,i}(x) = L_j((x - x_{2i})/h)$ where $L_j(\ell) = 1$ if $j = \ell$ and zero if $j \ne \ell$ for $j, \ell = 0, 1, 2$. These functions are easier to compute: $L_0(s) = (s - 1)(s - 2)/2$, $L_1(s) = s(2 - s)$, and $L_2(s) = s(s - 1)/2$. Then

**Algorithm 62** Simpson's rule

```
1  function simpson(f, a, b, n) // n assumed even
2    h ← (b − a)/n
3    s ← f(a) + f(b)
4    s ← s + 4f(x_1)
5    for i = 1, 2, ..., n − 1
6      s ← s + 2 f(x_{2i}) + 4 f(x_{2i+1})
7    end for
8    return (h/3) · s
9  end function
```

$$\begin{aligned}\int_{x_{2i}}^{x_{2i+2}} f(x)\,dx \approx \int_{x_{2i}}^{x_{2i+2}} p_{2,i}(x)\,dx &= \int_{x_{2i}}^{x_{2i+2}} \sum_{j=0}^{2} f(x_{2i+j})\,L_{j,i}(x)\,dx \\ &= \sum_{j=0}^{2} f(x_{2i+j}) \int_{x_{2i}}^{x_{2i+2}} L_{j,i}(x)\,dx \\ &= \sum_{j=0}^{2} f(x_{2i+j})\,h \int_{0}^{2} L_j(s)\,ds \\ &= h\left[\frac{1}{3}f(x_{2i}) + \frac{4}{3}f(x_{2i+1}) + \frac{1}{3}f(x_{2i+2})\right].\end{aligned}$$

Thus

$$\begin{aligned}\int_a^b f(x)\,dx \approx \sum_{i=0}^{(n/2)-1} p_{2,i}(x)\,dx &= \sum_{i=0}^{(n/2)-1} h\left[\frac{1}{3}f(x_{2i}) + \frac{4}{3}f(x_{2i+1}) + \frac{1}{3}f(x_{2i+2})\right] \\ &= (h/3)\left[f(x_0) + 4\sum_{i=0}^{(n/2)-1} f(x_{2i+1}) + 2\sum_{i=1}^{(n/2)-1} f(x_{2i}) + f(x_n)\right].\end{aligned}$$

Pseudo-code for Simpson's rule is given in Algorithm 62.

The natural approach to error analysis is to estimate the error on each piece:

$$\begin{aligned}e_i &= \int_{x_{2i}}^{x_{2i+2}} f(x)\,dx - \int_{x_{2i}}^{x_{2i+2}} p_{2,i}(x)\,dx \\ &= \int_{x_{2i}}^{x_{2i+2}} (f(x) - p_{2,i}(x))\,dx \\ &= \int_{x_{2i}}^{x_{2i+2}} \frac{f'''(c_{x,i})}{3!}(x - x_{2i})(x - x_{2i+1})(x - x_{2i+2})\,dx. \qquad (5.1.10)\end{aligned}$$

Note that like the case of the mid-point rule, we cannot apply the Generalized Mean Value Theorem as the polynomial part $(x - x_{2i})(x - x_{2i+1})(x - x_{2i+2})$ changes sign on the interval $[x_{2i}, x_{2i+2}]$. Also like the mid-point rule,

$$\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+1})(x - x_{2i+2})\,dx = 0 \tag{5.1.11}$$

because $x_{2i+1}$ is the exact midpoint of $[x_{2i}, x_{2i+2}]$. The empirical evidence from Figure 5.1.2 shows that the error is $\mathcal{O}(h^4)$ instead of $\mathcal{O}(h^3)$. We can find the reason as (5.1.10) shows that if $f$ is a cubic polynomial, $f'''(c_{x,i})$ is constant and so (5.1.11) gives $e_i = 0$. Let $\widetilde{p}_{3,i}(x)$ be the cubic interpolant of $f(x_{2i})$, $f(x_{2i+1})$, $f(x_{2i+2})$ and $f'(x_{2i+1})$. As Simpson's rule is exact for cubic polynomials,

$$\int_{x_{2i}}^{x_{2i+2}} \widetilde{p}_{3,i}(x)\,dx = \frac{h}{3}\left[f(x_{2i}) + 4\,f(x_{2i+1}) + f(x_{2i+2})\right] = \int_{x_{2i}}^{x_{2i+2}} p_{2,i}(x)\,dx.$$

Then the error for $[x_{2i}, x_{2i+2}]$ is

$$\begin{aligned}
e_i &= \int_{x_{2i}}^{x_{2i+2}} (f(x) - \widetilde{p}_{3,i}(x))\,dx \\
&= \int_{x_{2i}}^{x_{2i+2}} \frac{f^{(4)}(\widetilde{c}_{c,i})}{4!}(x - x_{2i})(x - x_{2i+1})^2(x - x_{2i+2})\,dx \\
&= \frac{f^{(4)}(\widetilde{c}_i)}{4!}\int_{x_{2i}}^{x_{2i+2}} (x - x_{2i})(x - x_{2i+1})^2(x - x_{2i+2})\,dx \\
&\qquad \text{for some } \widetilde{c}_i \in [x_{2i}, x_{2i+2}] \text{ by Generalized Mean Value Theorem} \\
&= \frac{f^{(4)}(\widetilde{c}_i)}{4!}h^5\int_0^2 s(s-1)^2(s-2)\,ds = \frac{1}{4!}\,\frac{-4}{15}f^{(4)}(\widetilde{c}_i)\,h^5 = -\frac{1}{90}f^{(4)}(\widetilde{c}_i)\,h^5. \\
e &= \sum_{i=0}^{(n/2)-1} -\frac{1}{90}f^{(4)}(\widetilde{c}_i)\,h^5 = -\frac{1}{90}h^5\sum_{i=0}^{(n/2)-1} f^{(4)}(\widetilde{c}_i).
\end{aligned}$$

This gives us a bound

$$|e| \le \frac{1}{90}h^5\frac{n}{2}\max_i \left|f^{(4)}(\widetilde{c}_i)\right| = \frac{1}{180}h^4(b-a)\max_{a\le c\le b}\left|f^{(4)}(c)\right|. \tag{5.1.12}$$

Asymptotically,

$$e \sim \frac{-1}{180}h^4\int_a^b f^{(4)}(c)\,dc = \frac{1}{180}h^4(f'''(a) - f'''(b)). \tag{5.1.13}$$

### *5.1.2 Newton–Cotes Methods*

Newton–Cotes methods are integration methods based on interpolation with equally spaced points. These generalize the trapezoidal and Simpson's rules. For interpolation polynomials of order $k$ we need $k + 1$ points, so each piece $[x_{ki}, x_{k(i+1)}]$ must have

$k+1$ interpolation points. The interpolation points are therefore $x_j = a + j\,h$ where $h = (b-a)/n$ and $n$ is a multiple of $k$. The polynomial interpolant on $[x_{ki}, x_{k(i+1)}]$ is

$$p_{k,i}(x) = \sum_{j=0}^{k} f(x_{ki+j})\, L_{j,i}(x)$$

where $L_{j,i}$ is the $j$th Lagrange interpolation polynomial on piece $i$: $L_{j,i}(x_{ki+\ell}) = 1$ if $j = \ell$ and zero if $j \neq \ell$. Using the assumption of equally spaced interpolation points, $L_{j,i}(x) = \widetilde{L}_j((x - x_{ki})/h)$ where $\widetilde{L}_j(\ell) = 1$ if $j = \ell$ and zero if $j \neq \ell$ for $j, \ell = 0, 1, 2, \ldots, k$. Again, $\widetilde{L}_j$ is a polynomial of degree $k$. The integral

$$\begin{aligned}\int_{x_{ki}}^{x_{k(i+1)}} f(x)\,dx \approx \int_{x_{ki}}^{x_{k(i+1)}} p_{k,j}(x)\,dx &= \sum_{j=0}^{k} f(x_{ki+j}) \int_{x_{ki}}^{x_{k(i+1)}} L_{j,i}(x)\,dx \\ &= \sum_{j=0}^{k} f(x_{ki+j})\, h \int_0^k \widetilde{L}_j(s)\,ds = h \sum_{j=0}^{k} \widetilde{w}_j\, f(x_{ki+j})\end{aligned}$$

where $\widetilde{w}_j = \int_0^k \widetilde{L}_j(s)\,ds$. Then

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{(n/k)-1} \int_{x_{ki}}^{x_{k(i+1)}} p_{k,j}(x)\,dx = h \sum_{i=0}^{(n/k)-1} \sum_{j=0}^{k} \widetilde{w}_j\, f(x_{ki+j}).$$

As with Simpson's method, if $k$ is *even*, then we get extra bump in the order of convergence as the method is also exact for polynomials of degree $k+1$. The reason is the symmetry of the method: $\widetilde{w}_j = \widetilde{w}_{k-j}$, so

$$\sum_{j=0}^{k} w_j (j - (k/2))^{k+1} = 0 = \int_0^k (s - (k/2))^{k+1} ds;$$

since the method is already exact for all polynomials of degree $\leq k$, the method is also exact for all polynomials of degree $\leq k+1$. Consequently, the error is $\mathcal{O}(h^{k+2})$ rather than $\mathcal{O}(h^{k+1})$ as is the case for $k$ *odd*.

Simpson's method is already exact for polynomials of degree $\leq 3$ so its error is $\mathcal{O}(h^4)$. To get errors that are asymptotically smaller, we need to go to $k = 4$ instead of just $k = 3$. Weights for some higher order Newton–Cotes are shown in Table 5.1.1.

Unlike the trapezoidal and Simpson's methods, the weights for higher order Newton–Cotes methods can be negative. This becomes an important issue for very higher order Newton–Cotes methods. As noted regarding the Runge phenomenon (see Section 4.1.1.13), it is better to reduce spacing before increasing the degree of the interpolant, especially with equally-spaced interpolation points. A theoretical

**Table 5.1.1** Some higher order Newton–Cotes integration weights; note that $\widetilde{w}_{k-j} = \widetilde{w}_j$.

| $k$ | $j$ | $\widetilde{w}_j$ | $k$ | $j$ | $\widetilde{w}_j$ | $k$ | $j$ | $\widetilde{w}_j$ |
|---|---|---|---|---|---|---|---|---|
| 4 | 0 | 14 / 45 | 8 | 0 | 3 956 / 14 175 | 10 | 0 | 80 355 / 299 376 |
| | 1 | 64 / 45 | | 1 | 23 552 / 14 175 | | 1 | 132 875 / 74 844 |
| | 2 | 8 / 15 | | 2 | −3 712 / 14 175 | | 2 | −80 875 / 99 792 |
| 6 | 0 | 41 / 140 | | 3 | 41 984 / 14 175 | | 3 | 28 375 / 6 237 |
| | 1 | 54 / 35 | | 4 | −3 632 / 2 835 | | 4 | −24 125 / 5 544 |
| | 2 | 27 / 140 | | | | | 5 | 89 035 / 12 474 |
| | 3 | 68 / 35 | | | | | | |

result that drives this point home is the following, which connects the integration error to the minimax approximation error.

**Theorem 5.1** *If $f : D \to \mathbb{R}$ is continuous with $D \subset \mathbb{R}^d$, and the integration method*

$$\int_D w(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x} \approx \sum_{i=1}^{N} w_i\, f(\boldsymbol{x}_i)$$

*is exact for all polynomials of degree $\le k$, then the error in the integral is bounded above by*

$$\left( \int_D |w(\boldsymbol{x})|\, d\boldsymbol{x} + \sum_{i=1}^{N} |w_i| \right) \min_{q \in \mathcal{P}_{k,d}} \|f - q\|_\infty .$$

***Proof*** The integration error is

$$\begin{aligned}
&\left| \int_D w(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x} - \sum_{i=1}^{N} w_i\, f(\boldsymbol{x}_i) \right| \qquad \text{and for any } q \in \mathcal{P}_{k,d}, \\
&\quad \le \left| \int_D w(\boldsymbol{x})\, \left(f(\boldsymbol{x}) - q(\boldsymbol{x})\right)\, d\boldsymbol{x} - \sum_{i=1}^{N} w_i\, \left(f(\boldsymbol{x}_i) - q(\boldsymbol{x}_i)\right) \right| \\
&\quad \le \int_D |w(\boldsymbol{x})|\, |f(\boldsymbol{x}) - q(\boldsymbol{x})|\, d\boldsymbol{x} + \sum_{i=1}^{N} |w_i|\, |f(\boldsymbol{x}_i) - q(\boldsymbol{x}_i)| \\
&\quad \le \int_D |w(\boldsymbol{x})|\, d\boldsymbol{x}\, \|f - q\|_\infty + \sum_{i=1}^{N} |w_i|\, \|f - q\|_\infty \\
&\quad = \left( \int_D |w(\boldsymbol{x})|\, d\boldsymbol{x} + \sum_{i=1}^{N} |w_i| \right) \|f - q\|_\infty .
\end{aligned}$$

Taking the minimum over all $q \in \mathcal{P}_{k,d}$ gives the desired result. □

Note that if our integration method is exact for constant functions (which is necessary for any order of convergence), we have

$$\int_D w(\boldsymbol{x})\, d\boldsymbol{x} = \sum_{i=1}^{N} w_i .$$

So if $w(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in D$ and $w_i > 0$, we have the integration error bounded by

$$2 \int_D w(\boldsymbol{x})\, d\boldsymbol{x}\ \|f - q\|_\infty .$$

Thus, we can bound the integration error in terms of the minimax approximation error.

However, Newton–Cotes methods of higher order have negative weights, as can be seen in Table 5.1.1. In fact, if $w_j^{(k)}$ are the weights of the $k$th order Newton–Cotes method, then [206]

$$\sum_{j=0}^{k} \left| w_j^{(k)} \right| \to \infty \qquad \text{as } k \to \infty.$$

In fact, $\sum_{j=0}^{k} \left| w_j^{(k)} \right|$ grows exponentially in $k$ as $k \to \infty$ [206, p. 279]. This makes them unsuitable for high order integration methods. However, the trapezoidal and Simpson's methods from the Newton–Cotes family do have positive weights, and in fact are very robust methods.

### 5.1.3 *Product Integration Methods*

In dealing with singularities, such as $\int_0^b x^{-\alpha} f(x)\, dx$ or $\int_0^b \ln x\ f(x)\, dx$, we need new methods. A simple approach we can use is to apply polynomial interpolation for $f(x)$, and determining the exact value of the integrals of the interpolating polynomials. For evaluation points $x_0, x_1, \ldots, x_n$, we write $f(x) \approx p_n(x) := \sum_{i=0}^{n} f(x_i)\, L_i(x)$ and we use

$$\int_a^b w(x)\ f(x)\, dx \approx \int_a^b w(x)\ p_n(x)\, dx = \sum_{i=0}^{n} f(x_i) \int_a^b w(x)\, L_i(x)\, dx.$$

The error in these approximations is bounded by

$$\int_a^b |w(x)|\, dx\ \|f - p_n\|_\infty .$$

We can use whatever interpolation scheme is appropriate, including equally-spaced interpolation points, and using Chebyshev interpolation points.

Product integration methods should not be restricted to cases of singularities. Product integration methods are also appropriate for smooth integrands, such as

$$\int_a^b \exp(-\beta(x-\widehat{x})^2)\, f(x)\, dx \qquad \text{with } \beta \gg |b-a|^{-2}\,.$$

Large value of $\beta$ means that to obtain reasonable accuracy using ordinary integration methods requires a spacing of interpolation points $h \ll \beta^{-1/2}$. However, we can accurately compute these integrals with $h \gg \beta^{-1/2}$ if we use

$$\int_a^b \exp(-\beta(x-\widehat{x})^2)\, f(x)\, dx \approx \int_a^b \exp(-\beta(x-\widehat{x})^2)\, p_n(x)\, dx,$$

and using exact, or near-exact, methods for computing the integral on the right. As was noted in Section 6.1.4, it is more important to reduce spacing than it is to increase degree of polynomial interpolants. In that sense, we find that when dealing with a singularity at $x = a$ in an integral

$$\int_a^b w(x)\, f(x)\, dx = \sum_{j=0}^{M-1} \int_{z_j}^{z_{j+1}} w(x)\, f(x)\, dx$$

with $a = z_0 < z_1 < \cdots < z_M = b$. Product integration methods can definitely be applied to the integral $\int_{z_0}^{z_1} w(x)\, f(x)\, dx$. However, the interval $[z_1,\ z_2]$ is often not far enough from the singularity for standard integration methods to work well. Instead, we should continue to use product integration methods. For example, for estimating

$$\int_0^b x^{-\alpha}\, f(x)\, dx = \sum_{j=0}^{M-1} \int_{z_j}^{z_{j+1}} x^{-\alpha}\, f(x)\, dx$$

we should compute

$$\int_u^v x^{-\alpha}\, p(x)\, dx$$

for $0 \le u < v$ and polynomials $p$ of degree $\le k$. To implement these methods, we need weights that depend on the interval:

$$w_i(u, v) = \int_u^v x^{-\alpha}\, L_i(x)\, dx, \qquad i = 0, 1, 2 \ldots, n$$

where $L_i$ is the $i$th Lagrange interpolation polynomial for interpolation points $x_i = u + \xi_i(v-u)$ in $[u, v]$. Here $\xi_i$ represent standardized interpolation points:

$\xi_i = i/n$ for equally spaced points, $\xi_i = \frac{1}{2}(1 - \cos((i + \frac{1}{2})\pi/(n + 1))$ for Chebyshev interpolation points.

Gaussian quadrature methods (see Section 5.2) can be applied if we can compute the orthogonal polynomials for the weight function $w(x) > 0$. In this case, very high-order methods can be applied as the weights $w_i > 0$, and so Theorem 5.1 gives good bounds on the error.

Product integration methods can also be used for multidimensional integration. See Section 5.3.

### 5.1.4 Extrapolation

"

*If we know the error, we can subtract it out to get a better approximation*."
While we very rarely know exactly what the error is for our numerical integration methods, often we can get asymptotic estimates, such as (5.1.5, 5.1.7, 5.1.9, 5.1.13). These enable us extrapolate from computed results using a known method to obtain superior results.

We start by supposing that the error $e_n = v_n - v^*$ in some computation that depends on a parameter $n$ has a known asymptotic behavior:

$$e_n \sim C\, n^{-\alpha} \qquad \text{as } n \to \infty. \tag{5.1.14}$$

Then *Richardson extrapolation* takes the values $v_n$ and $v_{2n}$ with the asymptotic error behavior

$$\begin{aligned} v_n - v^* &\approx C\, n^{-\alpha}, \\ v_{2n} - v^* &\approx C\, (2n)^{-\alpha}. \end{aligned}$$

Treating these approximations as exact we can solve for a new approximation for $v^*$: $v_{2n} - v^* \approx 2^{-\alpha}(v_n - v^*)$, so

$$v^* \approx v_{2n}^{(1)} := \frac{v_{2n} - 2^{-\alpha} v_n}{1 - 2^{-\alpha}}. \tag{5.1.15}$$

Now $v_{2n}^{(1)} \to v^*$ as $n \to \infty$, but at a higher asymptotic order.

As an example, consider using the trapezoidal rule for estimating $\int_1^2 x \, \ln x \, dx$. Let $T_n$ be the estimated value using the trapezoidal method with $n + 1$ function evaluations. From (5.1.7) we have $T_n - T^* \sim C\, n^{-2}$ where $T^*$ is the exact value of the integral, so $\alpha = 2$. Then Richardson extrapolation gives the estimates

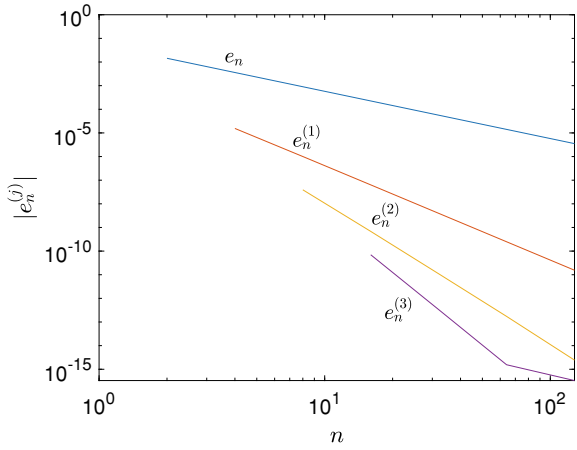



**Fig. 5.1.3** Errors for Romberg method for $\int_1^2 x \ln x \, dx$

$$T_{2n}^{(1)} = \frac{T_{2n} - \frac{1}{4}T_n}{1 - \frac{1}{4}}.$$

It turns out that $T_{2n}^{(1)}$ is the result of Simpson's method with $2n + 1$ evaluations. The error $e_n^{(1)} \sim C^{(1)} n^{-4}$ as $n \to \infty$, so we can repeat the process to get

$$T_{2n}^{(2)} = \frac{T_{2n}^{(1)} - \frac{1}{16}T_n^{(1)}}{1 - \frac{1}{16}}.$$

Continuing in this way we have

$$T_{2n}^{(j+1)} = \frac{T_{2n}^{(j)} - 2^{-2j-2}T_n^{(j)}}{1 - 2^{-2j-2}} \qquad \text{for } j = 0, 1, 2, \ldots. \tag{5.1.16}$$

This repeated application of Richardson extrapolation is the *Romberg method* [215]. The error $e_n^{(j)} \sim C^{(j)} n^{-2j-2}$ as $n \to \infty$. This can give very fast convergence, as can be seen in Figure 5.1.3.

This approach can be used for other techniques and integrals with other asymptotic behavior. Consider, for example, computing $\int_0^1 \sqrt{x}\, e^x \, dx$. If we use the trapezoidal method we still get convergence: $e_n = T_n - T^* \to 0$ as $n \to \infty$ by Theorem 5.1. The error is shown in Figure 5.1.4. The slope of $|e_n|$ the straight part of the graph on the log–log plot is estimated to be $-1.477$; using $\alpha = -3/2$ for applying Richardson extrapolation, we get $T_{2n}^{(1)} = (T_{2n} - 2^{-3/2}T_n)/(1 - 2^{-3/2})$. The errors $e_n^{(1)} = T_n^{(1)} - T^*$ go to zero faster. An empirical estimate of the slope on the log–log plots obtained as $-1.997$; using $\alpha = -2$ for applying Richardson extrapolation, we get $T_{2n}^{(2)} = (T_{2n}^{(1)} - 2^{-2}T_n^{(1)})/(1 - 2^{-2})$. The slope of the errors $e_n^{(2)} = T_n^{(2)} - T^*$ was estimated to be $-2.493$, which is remarkably close to $-5/2$. Repeating Richardson

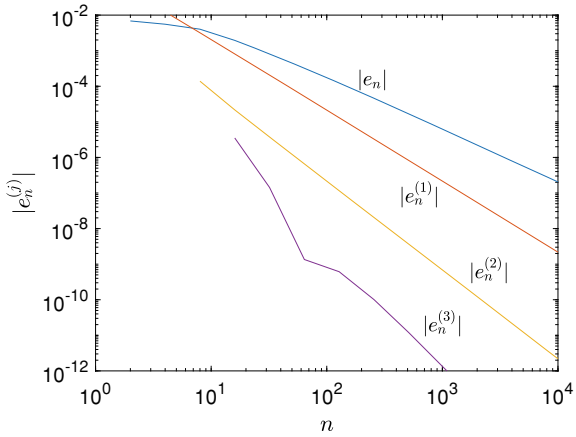



**Fig. 5.1.4** Romberg method adapted to $\int_0^1 \sqrt{x}\, e^x\, dx$

extrapolation with $\alpha = -5/2$ we get $T_{2n}^{(3)} = (T_{2n}^{(2)} - 2^{-5/2} T_n^{(2)})/(1 - 2^{-5/2})$, a third derived sequence where the error has a slope estimated to be about $-2.96$ which is very close to $-3$. Figure 5.1.4 shows that this gives a much more rapid reduction in the error as $n$ increases.

The reason why the Romberg method works is that there is a fully developed asymptotic expansion

$$e_n \sim \frac{C_1}{n^2} + \frac{C_2}{n^4} + \frac{C_3}{n^6} + \cdots$$

Proving this for smooth $f$ leads to *Euler–MacLaurin summation*, a way of estimating sums via integrals (or vice-versa) that uses a fully developed asymptotic expansion.

Fully developed asymptotic expansions

$$f(n) \sim C_1 g_1(n) + C_2 g_2(n) + C_3 g_3(n) + \cdots$$

mean that $g_{j+1}(n)/g_j(n) \to 0$ as $n \to \infty$,

$$\begin{aligned} f(n)/g_1(n) &\to C_1 && \text{as } n \to \infty, \text{ and} \\ \big[f(n) - C_1 g_1(n) - \cdots - C_j g_j(n)\big]/g_{j+1}(n) &\to C_{j+1} && \text{as } n \to \infty \text{ for } j = 1, 2, \ldots. \end{aligned}$$

The error due to the trapezoidal method can be represented via an integral:

$$\int_0^1 f(x)\, dx = \frac{1}{2}\,[f(0) + f(1)] - \frac{1}{2!} \int_0^1 f''(x)\, B_2(x)\, dx,$$

where $B_2(x) = x(1 - x)$ is the quadratic Bernoulli polynomial. Repeated applications of integration by parts along with the definitions of Bernoulli numbers and polynomials give

$$\int_0^1 f(x)\,dx = \frac{1}{2}\,[f(0)+f(1)] + \sum_{k=1}^{j-1} \frac{B_{2k}}{(2k)!}[f^{(2k-1)}(1) - f^{(2k-1)}(0)]$$

$$- \frac{1}{(2j)!}\int_0^1 f^{(2j)}(x)\,B_{2j}(x)\,dx. \tag{5.1.17}$$

The Bernoulli numbers $B_k$ are given by the formula

$$\frac{t}{e^t-1} = \sum_{j=0}^{\infty} B_j \frac{t^j}{j!} \qquad \text{for all } t, \tag{5.1.18}$$

while the Bernoulli polynomials $B_k(x)$ are given by

$$\frac{t(e^{xt}-1)}{e^t-1} = \sum_{j=0}^{\infty} B_j(x)\frac{t^j}{j!} \qquad \text{for all } t. \tag{5.1.19}$$

We let $h = (b-a)/n$ be a fixed spacing: $x_{i+1} - x_i = h$ for all $i$. Scaling and shifting (5.1.17) to obtain $\int_{x_i}^{x_{i+1}} f(x)\,dx$ by, and then summing over $i$ gives

$$\int_a^b f(x)\,dx = h\left[\frac{1}{2}\,(f(a)+f(b)) + \sum_{i=1}^{n-1} f(x_i)\right]$$
$$+ \sum_{k=1}^{j-1} \frac{h^{2k}B_{2k}}{(2k)!}[f^{(2j-1)}(b) - f^{(2j-1)}(a)]$$
$$- \frac{h^{2j}}{(2j)!}\int_a^b f^{(2j)}(x)\,\overline{B_{2j}}((x-a)/h)\,dx,$$

which gives us the fully developed asymptotic expansion for the error in the trapezoidal method:

$$e_n \sim \sum_{k=1}^{\infty} \frac{(b-a)^{2k}B_{2k}}{(2k)!}[f^{(2j-1)}(b) - f^{(2j-1)}(a)]\frac{1}{n^{2k}}. \tag{5.1.20}$$

The infinite sum is understood as an asymptotic expansion, not as a convergent series.

**Exercises.**

(1) Apply the rectangle, trapezoidal, and Simpson's rules to estimate $\int_0^{10} x\cos(x^2)\,dx$ with $n+1$ function evaluations ($n$ function evaluations in the case of the rectangle rule) for $n = 2^k$, $k = 1, 2, \ldots, 15$. Plot the error magnitude against $n$ on a log–log plot. Estimate the exponent $\alpha$ for the error asymptotes $|\text{error}_n| \approx C\,n^{-\alpha}$ for the three methods. Note that the exact value is $\frac{1}{2}\sin(100)$.

(2) Repeat the previous Exercise for estimating $\int_{-5}^{+5} dx/(1+x^2)$. Exact value is $2\tan^{-1}(5)$.

(3) Repeat the previous Exercise for estimating $\int_0^2 \left|x-\sqrt{2}\right|^{1/2} dx$. Exact value is $\frac{1}{3}2^{7/4}[1+(\sqrt{2}-1)^{3/2}]$.

(4) Repeat the previous Exercise for estimating $\int_0^{2\pi} d\theta/(1+\sin^2\theta)$. Exact value is $\sqrt{2}\,\pi$.

(5) Show rapid convergence of the rectangle rule $\int_0^{2\pi} f(x)\,dx$ for smooth periodic $f$: $f(t+2\pi)=f(t)$ for all $t$. [**Hint:** First show that the rectangle rule with $n$ function evaluations on $[0,2\pi]$ is exact for all trigonometric polynomials $\sum_{k=0}^{n}[a_k\cos(kx)+b_k\sin(kx)]$. Now show that $f$ smooth implies that $f$ can be accurately approximated by truncated Fourier series of order $n$ (which is a trigonometric polynomial). This can be done using integration by parts.]

(6) The Clenshaw–Curtis integration method [51] is to compute the finite Chebyshev expansion $f(x)\approx\sum_{k=0}^{n}a_k\,T_k(x)$ from interpolation of $f(x_j)$, $j=0,1,2,\ldots,n$, with $x_j=-\cos(j\pi/n)$. This can be done efficiently using the DFT. Show that $\int_{-1}^{+1} f(x)\,dx\approx\sum_{k=0}^{n}a_k\int_0^{\pi}\cos(k\theta)\,\sin\theta\,d\theta$. Compute the exact value of $\int_0^{\pi}\cos(k\theta)\,\sin\theta\,d\theta$.

(7) Fejér [90] used interpolation at the points $x_j=\cos((j+\frac{1}{2})\pi/n)$, $j=0,1,2,\ldots,n-1$, to create the integration scheme $\int_{-1}^{+1} f(x)\,dx\approx\sum_{k=0}^{n}f(x_k)\int_{-1}^{+1}L_k(x)\,dx$ where $L_k$ is the Lagrange interpolation polynomial. Show, as Fejér did in 1933, that $\int_{-1}^{+1}L_k(x)\,dx>0$ for $k=0,1,2,\ldots,n$. [**Hint:** Start by showing that $L_k(\cos\theta)=1+2\sum_{j=1}^{n-1}\cos(j\theta)\,\cos(j(k+\frac{1}{2})\pi/n)$.]

(8) Implement a tensor-product Simpson's method in two dimensions: if $\int_a^b f(x)\,dx\approx(b-a)\sum_{i=0}^{n}w_i f(a+(b-a)\xi_i)$ then $\int_a^b\int_c^d f(x,y)\approx(b-a)(d-c)\sum_{i=0}^{n}\sum_{j=0}^{n}w_i w_j f(a+(b-a)\xi_i,\,c+(d-c)\xi_j)$. Implement this, if possible, using a one-dimensional Simpson's rule code.

(9) The asymptotic trapezoidal expansion (5.1.20) can be used to give accurate estimates of infinite sums. Consider the problem of estimating $\sum_{k=0}^{\infty}1/(1+k^2)$. Since $\int_N^{\infty}dx/(1+x^2)=\pi/2-\tan^{-1}N$ approximates $\frac{1}{2}(1+N^2)^{-1}+\sum_{k=N+1}^{\infty}1/(1+k^2)$ with an asymptotic error formula (5.1.20) we can obtain asymptotically accurate estimates for $\frac{1}{2}(1+N^2)^{-1}+\sum_{k=N+1}^{\infty}1/(1+k^2)$. Added to the numerically computed value $\sum_{k=0}^{N-1}1/(1+k^2)+\frac{1}{2}(1+N^2)^{-1}$, this can give accurate estimates for $\sum_{k=0}^{\infty}1/(1+k^2)$. Use $N=10^2,\ 10^3,\ 10^5,\ 10^6$ and asymptotic error estimates with one, two and three terms to obtain estimates of $\sum_{k=0}^{\infty}1/(1+k^2)$. Check that these estimates appear to be consistent with each other.

(10) Develop a product integration method for computing

$$\int_a^b (2\pi\sigma)^{-1}\exp(-(x-\mu)^2/(2\sigma^2))\,f(x)\,dx$$

that remains accurate even for small $\sigma > 0$ and arbitrary $\mu \in \mathbb{R}$ by using a piecewise quadratic interpolant of $f$. Note that you will need to have explicit formulas for $\int_a^b \exp(-(x-\mu)^2/(2\sigma^2))\, x^k\, dx$ for $k = 0, 1, 2$. These formulas can involve the error function $\operatorname{erf}(z) = (2/\sqrt{\pi}) \int_0^z e^{-z^2}\, dz$, which is available on most modern computing systems.

## 5.2 Gaussian Quadrature

When we use the "integrate the interpolant" approach to numerical integration, we expect the order of accuracy of the computed integral to be the order of accuracy of the interpolant. But, as we have seen with the mid-point and Simpson's rules, sometimes we get a better order of accuracy than expected. This is super-convergence. Both the mid-point and Simpson's rules give an order of accuracy one more than expected. Newton–Cotes methods using even degree interpolation show this superconvergent behavior: the method is accurate for degrees one higher than the interpolant.

In the case of these methods, because the integration rule is symmetric under the reflection of the interval $[a, b]$ with weights unchanged, we obtain superconvergence with the order of accuracy one more than expected. How far can we take this approach? If we choose $n$ interpolation points $x_i$ and weights $w_i$, when can we guarantee that

$$\int_a^b p(x)\, dx = \sum_{i=1}^{n} w_i\, p(x_i) \tag{5.2.1}$$

for all polynomials $p$ of degree $\leq k$? In this way of thinking, we have $n$ evaluation points $x_i$ and $n$ weights $w_i$ for the integration method, giving a total of $2n$ parameters to define the method. To satisfy the equality in (5.2.1) for all polynomials of degree $\leq k$, we need to satisfy $k + 1$ independent equations. Balancing unknowns and equations gives $2n = k + 1$, so we have reason to expect that $n$ function evaluationss we could make (5.2.1) hold for all polynomials of degree $\leq 2n - 1$. How to achieve that is the subject of this section. The theory is based on orthogonal polynomials (see Section 4.7.2).

### *5.2.1 Orthogonal Polynomials Reprise*

Given an inner product on functions

$$(f,\ g) = \int_a^b w(x)\, f(x)\, g(x)\, dx, \tag{5.2.2}$$

with integrable $w(x) > 0$ for all $x \in (a, b)$, a family of polynomials $\phi_0, \phi_1, \phi_2, \ldots$ is a family of orthogonal polynomials if $(\phi_i, \phi_j) = 0$ for all $i \neq j$, and $\deg \phi_j = j$ for $j = 0, 1, 2, \ldots$. Scaling each member of this family by $\psi_j = \alpha_j \phi_j$ with $\alpha_j \neq 0$ gives a new family of orthogonal polynomials with respect to the inner product (5.2.2). In fact, given the inner product (5.2.2), every family of orthogonal polynomials can only differ by these scalings. Since $\deg \phi_j = j$, the span of $\{\phi_0, \ldots, \phi_\ell\}$ is exactly the set of polynomials of degree $\leq \ell$. Thus $\phi_{\ell+1}$ is orthogonal to $\mathcal{P}_\ell$, the set of polynomials of degree $\leq \ell$, which has dimension $\ell + 1$. On the other hand, $\phi_{\ell+1}$ is in $\mathcal{P}_{\ell+1}$, which has dimension $\ell + 2$. Thus $\phi_{\ell+1}$ has to lie on a one-dimensional subspace of $\mathcal{P}_{\ell+1}$. The only choice determining $\phi_{\ell+1}$ is its scaling.

Properties and examples of orthogonal polynomials can be found in Section 4.7.2. One property not in Section 4.7.2 that is relevant to us here is the number of zeros of $\phi_j$ in $(a, b)$:

**Theorem 5.2** *In a family of orthogonal polynomials $\{\phi_0, \phi_1, \ldots\}$ with respect to the inner product (5.2.2), $\phi_j$ has exactly $j$ simple roots in $(a, b)$.*

***Proof*** First we show that $\phi_j$ has at least $j$ roots in $(a, b)$ by contradiction. Suppose $\phi_j$ has $r < j$ roots in $(a, b)$. Then the number of points $x$ at which $\phi_j(x)$ changes sign is $s \leq r < j$. Call these points $z_1, z_2, \ldots, z_s \in (a, b)$. Let $\psi(x) = \prod_{i=1}^{s} (x - z_s)$ which has degree $s \leq r < j$. Because $\deg \psi = s$, we can write $\psi$ as a linear combination of $\phi_0, \phi_1, \ldots, \phi_s$, each of which is orthogonal to $\phi_j$. Therefore, $\psi$ is orthogonal to $\phi_j$ and so

$$0 = (\phi_j, \psi) = \int_a^b w(x)\, \phi_j(x)\, \psi(x)\, dx.$$

The integrand $w(x)\, \phi_j(x)\, \psi(x)$ does not change sign between positive and negative and can only be zero at a finite number of points. Thus, $(\phi_j, \psi)_w = \int_a^b w(x)\, \phi_j(x)\, \psi(x)\, dx \neq 0$, which is a contradiction.

Thus $\phi_j$ has at least $j$ roots in $(a, b)$. To show that $\phi_j$ has at most $j$ roots in $(a, b)$, we recall that $\phi_j$ has degree $j$, and so the total number of roots of $\phi_j$ is $j$, counting multiplicity. Thus the multiplicity of each root must be one, and there cannot be more than $j$ of them. That is, $\phi_j$ has exactly $j$ simple roots in $(a, b)$. □

Since scaling a polynomial does not change its roots, the inner product uniquely specifies the roots of $\phi_j$ in any family of orthogonal polynomials.

### 5.2.2 Orthogonal Polynomials and Integration

We seek an integration method

$$\int_a^b w(x)\, f(x)\, dx \approx \sum_{i=1}^{n} w_i\, f(x_i)$$

that is exact if $f$ is a polynomial of degree $\leq 2n-1$. These methods are *Gaussian quadrature* methods. If we have already chosen the points $x_i$, then the weights $w_i$ are given by

$$w_i = \int_a^b w(x)\, L_i(x)\, dx \tag{5.2.3}$$

where $L_i$ is the $i$th Lagrange interpolation polynomial (4.1.3) for the interpolation points $x_1, x_2, \ldots, x_n$. With $n$ interpolation points, each $L_i$ has degree $n-1$. This ensures that the method is exact for all polynomials of degree $\leq n-1$. If $f$ is a polynomial of degree $\leq 2n-1$ we can use *synthetic division* to write

$$f(x) = q(x)\, \phi_n(x) + r(x),$$

where $q$ and $r$ are polynomials of degree $\leq n-1$. Then

$$\begin{aligned}\int_a^b w(x)\, f(x)\, dx &= \int_a^b w(x)\, \left[q(x)\, \phi_n(x) + r(x)\right]\, dx \\ &= \int_a^b w(x)\, q(x)\, \phi_n(x)\, dx + \int_a^b w(x)\, r(x)\, dx. \\ &= (q,\ \phi_n) + \int_a^b w(x)\, r(x)\, dx.\end{aligned}$$

Now $(q,\ \phi_n) = 0$ since $q$ is a polynomial of degree $\leq n-1$, and therefore can be written as a linear combination of $\phi_0, \phi_1, \ldots, \phi_{n-1}$, each of which is orthogonal to $\phi_n$. On the other hand, by the construction of the $w_i$'s in (5.2.3), the method is already exact for polynomials of degree $\leq n-1$, so it must be exact for $r(x)$. Therefore

$$\int_a^b w(x)\, f(x)\, dx = \int_a^b w(x)\, r(x)\, dx = \sum_{i=1}^n w_i\, r(x_i).$$

What we want is

$$\int_a^b w(x)\, f(x)\, dx = \sum_{i=1}^n w_i\, f(x_i) = \sum_{i=1}^n w_i\, \left[q(x_i)\, \phi_n(x_i) + r(x_i)\right].$$

In order to guarantee this, we need $\sum_{i=1}^n w_i\, q(x_i)\, \phi_n(x_i) = 0$ for any polynomial $q$ of degree $\leq n-1$. Setting $q(x) = L_j(x)$ for $j = 1, 2, \ldots, n$ we see that $\phi_n(x_i) = 0$. That is, the evaluation points should be the roots of $\phi_n$.

In summary, to create an integration method

$$\int_a^b w(x)\, f(x)\, dx \approx \sum_{i=1}^n w_i\, f(x_i)$$

that is exact whenever $f$ is a polynomial of degree $\leq 2n-1$, we first compute the roots of $\phi_n$ which is the degree $n$ polynomial from the family of orthogonal polynomials for the inner product (5.2.2). The evaluation points $x_i$ are the roots of $\phi_n$. We then set the weights $w_i$ according to (5.2.3).

### 5.2.3 *Why the Weights are Positive*

In contrast to the high order Newton–Cotes integration methods of Section 5.1.2, the weights for Gaussian quadrature methods are all positive. To see why, consider $f(x) = L_i(x)^2$ where $L_i$ is the $i$th Lagrange interpolation polynomial for $x_1, x_2, \ldots, x_n$. Since we have just $n$ interpolation points, $\deg L_i = n-1$, and therefore $\deg L_i^2 = 2(n-1) = 2n-2 \leq 2n-1$, so the method is exact for $L_i^2$:

$$0 < \int_a^b w(x)\, L_i(x)^2\, dx = \sum_{j=1}^{n} w_j\, L_i(x_j)^2.$$

But $L_i(x_j) = 1$ if $i = j$ and zero if $i \neq j$, so $\sum_{j=1}^{n} w_j\, L_i(x_j)^2 = w_i$; thus $w_i > 0$.

Having positive weights is an important bonus as then we can apply Theorem 5.1 to show that the integration error is bounded by

$$2 \int_a^b w(x)\, dx\ \|f - q\|_\infty$$

for any polynomial $q$ of degree $\leq 2n-1$.

To estimate the errors in a more conventional way, we can use the Hermite interpolant $\widetilde{p}_n(x)$ where $\widetilde{p}_n(x_i) = f(x_i)$ and $\widetilde{p}_n'(x_i) = f'(x_i)$ for $i = 1, 2, \ldots, n$, which is a polynomial of degree $2n-1$: since $\widetilde{p}_n$ is exactly integrated by this method,

$$\begin{aligned}
&\int_a^b w(x)\ f(x)\, dx - \sum_{i=1}^{n} w_i\ f(x_i) \\
&\quad = \int_a^b w(x)\, (f(x) - \widetilde{p}_n(x))\, dx - \sum_{i=1}^{n} w_i\, (f(x_i) - \widetilde{p}_n(x_i)) \\
&\quad = \int_a^b w(x)\, (f(x) - \widetilde{p}_n(x))\, dx \\
&\quad = \int_a^b w(x) \frac{f^{(2n)}(c_x)}{(2n)!} \prod_{i=1}^{n} (x - x_i)^2\, dx \\
&\quad = \frac{f^{(2n)}(c)}{(2n)!} \int_a^b w(x) \prod_{i=1}^{n} (x - x_i)^2\, dx
\end{aligned}$$

for some $c$ between $a$ and $b$ by the Generalized Mean Value Theorem as $w(x)\prod_{i=1}^{n}(x-x_i)^2$ is never negative on $(a,b)$.

In the case where $w(x)=1$ for all $x$, the orthogonal polynomials for $(f,g)=\int_{-1}^{+1} f(x)\,g(x)\,dx$ are the Legendre polynomials given by the Rodriguez formula (4.7.9)

$$P_n(x)=\frac{1}{2^n\,n!}\frac{d^n}{dx^n}\left[\left(x^2-1\right)^n\right].$$

Because $w(x)$ is constant, this Gaussian quadrature method can be used as a composite method with error $\mathcal{O}(h^{2n})$ where $h$ is the common width of each piece in

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{j=0}^{M-1}\int_{z_j}^{z_{j+1}} f(x)\,dx\\ &= \sum_{j=0}^{M-1}\sum_{i=1}^{n} h\,w_i\,f(z_j+h(1+x_i)/2)\end{aligned}$$

where $x_i$ is the $i$th zero of $P_n$ in $(-1,+1)$. The points $x_i$ and weights $w_i$ can be efficiently computed via the coefficients of the three-term recurrence relation (4.7.8) by the algorithm of [106].

**Exercises.**

(1) From the Rodriguez formula for Legendre polynomials, compute $(P_n,\ P_n)$ and $(P_n,\ x\,P_{n-1})=\int_{-1}^{+1} P_n(x)\,x\,P_{n-1}(x)\,dx$.

(2) Determine the coefficients $a_n$, $b_n$, and $c_n$ where $P_{n+1}(x)=(a_n x+b_n)\,P_n(x)-c_n\,P_{n-1}(x)$. Re-arrange these equations to find coefficients $\alpha_n$ and $\beta_n$ so that $x\,P_n(x)=\gamma_{n-1}P_{n-1}(x)+\alpha_n P_n(x)+\beta_n P_{n+1}(x)$.

(3) Suppose that the polynomials $p_0,\ p_1,\ p_2,\ \ldots$ are a family of orthogonal polynomials with respect to the inner product $(f,g)_w=\int_a^b w(x)\,f(x)\,g(x)\,dx$. Once the roots $x_0,x_1,\ldots,x_n$ of $p_{n+1}$ have been found, the weights for Gaussian quadrature can be computed as $w_j=\int_a^b w(x)\,L_j(x)\,dx$ where $L_j$ is the $j$th Lagrange interpolation polynomial. Show that $L_j(x)=p_{n+1}(x)/[p'_{n+1}(x_j)(x-x_j)]$.

(4) Compute the family of orthogonal polynomials (up to a scale factor for each polynomial) of all degrees $\le 5$ for $(f,g)_w=\int_0^1 -\ln(x)\,f(x)\,g(x)\,dx$.

(5) Use the results of the previous Exercise to give a 5-point formula that can compute $\int_0^1 -\ln x\,f(x)\,dx$ whenever $f$ is a polynomial of degree $\le 9$.

(6) Compute the family of orthogonal polynomials (up to a scale factor for each polynomial) of all degrees $\le 5$ for $(f,g)_w=\int_0^1 x^{-1/2}\,f(x)\,g(x)\,dx$.

(7) Compute the orthogonal polynomials in Exercise 6 via the Rodriguez formula

$$p_n(x)=c_n\,x^{1/2}\frac{d^n}{dx^n}\left[x^{n-1/2}(1-x)^n\right].$$

(8) Using polar co-ordinates, develop a method to compute integrals over the unit disk $\mathbb{D}$: $\int_{\mathbb{D}} f(x, y)\,dx\,dy = \int_0^{2\pi} \int_0^1 f(r\,\cos\theta,\ r\,\sin\theta)\,r\,dr\,d\theta$. Use scaled and shifted Gauss–Legendre quadrature in $r$ for the interval $[0, 1]$, and the rectangle rule in $\theta$ (because the integrand is periodic in $\theta$). Use $n$ integration points for each polar co-ordinate with $n = 2^k$, $k = 1, 2, \ldots, 5$. Test this method on the function $f(x, y) = e^x(1 + x^2 + 3y^2)^{-1/2}$; for the "exact" value, use $n = 2^8$.

(9) Using spherical polar co-ordinates, develop a method to compute integrals over the unit ball $\mathbb{B}$:

$$\int_{\mathbb{B}} f(x, y, z)\,dx\,dy\,dz = \int_0^1 \int_{-\pi/2}^{+\pi/2} \int_0^{2\pi} f(r\,\cos\theta\,\sin\phi,\ r\,\cos\theta\,\cos\phi,\ r\,\sin\theta)\,r^2\,\cos\theta\,d\phi\,d\theta\,dr.$$

Use Gauss–Legendre quadrature in $r$ and $\theta$, and the rectangle rule in $\phi$. Use $n$ points in each polar co-ordinate to give an estimate for the integral:

$$\int_{\mathbb{B}} f(x, y, z)\,dx\,dy\,dz \approx \sum_{j,k,\ell=0}^{n-1} w_j^{(1)} w_k^{(2)} w_\ell^{(3)}\, f(r_j \cos\theta_k \sin\phi_\ell, r_j \cos\theta_k \cos\phi_\ell, r_j \sin\theta_k)\, r_j^2 \cos\theta_k.$$

## 5.3 Multidimensional Integration

Once one-variable integration is mastered, multivariate integration can be understood as repeated one-variable integration. In this way, multivariate integration should not be much harder than one-variable integration. But many integration regions do not have such nice Cartesian product structure. High-dimensional integration also requires different approaches to avoid exponential growth of the computational cost as the dimension increases.

### *5.3.1 Tensor Product Methods*

Tensor product methods are based on the observation that integrals over rectangles, and rectangular solids, can be expressed as repeated one-variable integrals. Specifically, if $R = \{\,(x, y) \mid a \le x \le b,\ c \le y \le d\,\}$, then

$$\int_R f(x, y)\,dx\,dy = \int_a^b \int_c^d f(x, y)\,dy\,dx.$$

A one-variable integration rule can be used in each co-ordinate: $\int_a^b g(x)\,dx \approx \sum_{i=1}^m v_i g(x_i)$, $\int_c^d h(y)\,dy \approx \sum_{j=1}^n w_j h(y_j)$. Combining these gives

$$\int_a^b \int_c^d f(x, y)\,dy\,dx \approx \sum_{i=1}^m \sum_{j=1}^n v_i w_j\, f(x_i, y_j). \tag{5.3.1}$$

In $d$ dimensions, if $R = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_d, b_d] \subset \mathbb{R}^d$ is the region of integration, then we can use a single rule $\int_a^b h(x)\,dx = (b-a)\sum_{i=1}^n w_i\, h(x_i)$ across each variable of integration to approximate

$$\int_R f(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{i_1,i_2,\ldots,i_d=1}^{n} \left(\prod_{j=1}^{d} w_{i_j}\right) f(x_{i_1}, x_{i_2}, \ldots, x_{i_d}) \tag{5.3.2}$$

if $R = [a, b]^d$. The number of function evaluations needed is $n^d$. While this is usually acceptable in two or three dimensions, it becomes much less so in higher dimensions.

### 5.3.2 *Lagrange Integration Methods*

Lagrange integration methods continue the tradition of integration methods based on interpolation. In this case, we use Lagrange interpolation on triangles, tetrahedra, and other simplices as described in Section 4.3.1. The best known method is arguably the one based on the vertices of a triangle $K \subset \mathbb{R}^2$:

$$\int_K f(\boldsymbol{x})\,d\boldsymbol{x} \approx \frac{1}{3}\left(f(\boldsymbol{v}_1) + f(\boldsymbol{v}_2) + f(\boldsymbol{v}_3)\right)\,\text{area}(K), \tag{5.3.3}$$

which is based on linear interpolation on the triangle. For a tetrahedron, $K \subset \mathbb{R}^3$ we have

$$\int_K f(\boldsymbol{x})\,d\boldsymbol{x} \approx \frac{1}{4}\sum_{i=1}^{4} f(\boldsymbol{v}_i)\,\text{vol}(K). \tag{5.3.4}$$

We can develop formulas for integration on a reference triangle, tetrahedron, or simplex $\widehat{K}$, and transfer the formula to a given element $K$ via an affine transformation, $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}}) = A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K$:

$$\int_K f(\boldsymbol{x})\,d\boldsymbol{x} = |\det A_K| \int_{\widehat{K}} f(\boldsymbol{T}_K(\widehat{\boldsymbol{x}}))\,d\widehat{\boldsymbol{x}}. \tag{5.3.5}$$

If an integration method is exact for polynomials of degree $\leq k$ on the reference element $\widehat{K}$, then using (5.3.5) will give a method that is exact for polynomials of degree $\leq k$ on $K$.

### 5.3.3 Symmetries and Integration

Symmetries can be very helpful in understanding integration methods on shapes like triangles and tetrahedra. An *affine symmetry* of a region $R \subset \mathbb{R}^d$ is an invertible affine transformation $R \to R$. The set of affine symmetries $R \to R$ forms a *group* under composition. The *order* of a group is the number of transformations in the group.

Unlike an interval $[a, b]$ which just has a reflection symmetry $x \mapsto a + b - x$ (the group of affine symmetries has order two), the group of affine symmetries on a triangle has order $3! = 6$, and on a tetrahedron has order $4! = 24$. Squares have groups of affine symmetries of order $2^2 \times 2!$ (involving reflections about each axis and $(x, y) \leftrightarrow (y, x)$) while cubes have symmetries of order $2^3 \times 3!$. Circles, disks, spheres, and balls have infinite affine symmetry groups. Symmetry can be helpful in improving the order of accuracy for a given number of evaluation points for an interval. Symmetry is discussed in Section 5.1.2 as the source of super-convergence for symmetric Newton–Cotes methods of even order.

Most integration methods on regions $R$ in $\mathbb{R}^d$ are symmetric in ways that reflect the symmetry of $R$. Of particular interest are points that are invariant under every affine symmetry of $R$. For triangles, the centroid is invariant under every affine symmetry of the triangle. For disks or spheres, the center is invariant under every affine symmetry. Often other useful points are often invariant under a subgroup of the affine symmetries, such as the mid-point of an edge, which is invariant under the subgroup that keeps the edge fixed.

Just as was noted regarding the mid-point and Simpson's rules in one variable (see Sections 5.1.1.1 and 5.1.1.2), symmetry of the interpolation points and weights is often related to an improved order of convergence. For example, the method (5.3.3) is invariant under all affine symmetries. Even better, though, is the one-point rule evaluated at the centroid:

$$\int_K f(\boldsymbol{x})\, d\boldsymbol{x} \approx f\left(\frac{\boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_3}{3}\right) \text{area}(K). \tag{5.3.6}$$

Both (5.3.3) and (5.3.6) are exact for linear functions on triangles.

Integrating the Lagrange quadratic interpolant (4.3.4) gives a method that is exact for quadratic functions. Interestingly, in this case, the integrals associated with the Lagrange interpolating polynomials for the vertices evaluate to zero; thus only the values at the mid-points need be evaluated, giving the rule

$$\int_K f(\boldsymbol{x})\, d\boldsymbol{x} \approx \frac{1}{3}\left[f\left(\frac{\boldsymbol{v}_1 + \boldsymbol{v}_2}{2}\right) + f\left(\frac{\boldsymbol{v}_2 + \boldsymbol{v}_3}{2}\right) + f\left(\frac{\boldsymbol{v}_3 + \boldsymbol{v}_1}{2}\right)\right] \text{area}(K), \tag{5.3.7}$$

which is a three-point formula that is exact for all quadratics. Again, this rule is invariant under all affine symmetries of a triangle.

**Table 5.3.1** Types of groups of points by symmetries. Note that unless indicated, all co-ordinate values are distinct ($\alpha \neq \beta \neq \gamma$ etc.). Also the sum of the barycentric coordinates must be one.

| Symmetry type | Barycentric coordinates | # Points |
|---|---|---|
| I | $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ | 1 |
| II | $(\alpha, \alpha, \beta)$ | 3 |
| III | $(\alpha, \beta, \gamma)$ | 6 |

(A) For tetrahedra

| Symmetry type | Barycentric coordinates | # Points |
|---|---|---|
| I | $(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$ | 1 |
| II | $(\alpha, \alpha, \alpha, \beta)$ | 4 |
| III | $(\alpha, \alpha, \beta, \beta)$ | 6 |
| IV | $(\alpha, \alpha, \beta, \gamma)$ | 12 |
| V | $(\alpha, \beta, \gamma, \delta)$ | 24 |

(B) For triangles

### *5.3.4 Triangles and Tetrahedra*

High-order integration formulas for triangles, tetrahedra, and other shapes have been found by a number of authors (see the foundational book [244] and the survey papers [57, 58, 167]). More recent high-order formulas have been obtained by direct application of numerical methods to the conditions rather than assuming symmetry properties (for example, see [252] for spherically symmetric regions). However, for low to moderate degree, symmetry principles have been very useful for finding efficient rules over triangles and tetrahedra (see, for example [168] from 1975 for triangles).

A question that may arise is why to have high-order rules of order, say ten or more, for most applications calling for integration over triangles or tetrahedra? Rarely do we need such high order accuracy. Finite element methods for second order elliptic partial differential equations need to compute integrals of the form $\int_K a(\boldsymbol{x})\, \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, d\boldsymbol{x}$ over a triangle or tetrahedron $K$. If we used, for example, the Argyris element for continuous first derivatives, the basis functions $\phi_i$ have degree five, which means that $\nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})$ has degree $2 \times (5-1) = 8$. Thus the degree of polynomials that our integration method integrates exactly should be *at least eight*. An integration method of order ten, exactly integrating polynomials of degree $\leq 10$, would then be able to exactly integrate $\int_K a(\boldsymbol{x})\, \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, d\boldsymbol{x}$ where $a(\boldsymbol{x})$ is a polynomial of degree $\leq 2$. Thus the error for general $a(\boldsymbol{x})$ would then be $\mathcal{O}(h_K^3 |K|)$ rather than $\mathcal{O}(h_K^{11} |K|)$ where $h_K = \operatorname{diam} K$. Thus the order of the integration method needs to be co-ordinated with the order of the elements chosen.

Below we show some formulas for low to moderate order on triangles and tetrahedra. The points are represented by their barycentric coordinates. All the methods

**Table 5.3.2** Integration methods for triangles

| | | Points | | Symmetry | |
|---|---|---|---|---|---|
| Order – # points | Weight | $\alpha$ | $\beta$ | type | Notes |
| 1 – 1 | 1 | 1/3 | – | I | Centroid method |
| 2 – 3 | 1/3 | 1/6 | 2/3 | II | |
| 3 – 4 | −9/16 | 1/3 | – | I | Stroud (1971) |
| | 25/48 | 1/5 | 3/5 | II | [244] |
| 5 – 7 | 9/40 | 1/3 | – | I | Radon (1948) |
| | $\frac{155-\sqrt{15}}{1200}$ | $\frac{6-\sqrt{15}}{21}$ | $\frac{9+2\sqrt{15}}{21}$ | II | [210] |
| | $\frac{155+\sqrt{15}}{1200}$ | $\frac{6+\sqrt{15}}{21}$ | $\frac{9-2\sqrt{15}}{21}$ | II | |
| 7 – 13 | −0.149570044467682 | 1/3 | – | I | Cowper (1973) |
| | 0.175615257433208 | 0.260345966079040 | 0.479308067841920 | II | [61] |
| | 0.053347235608838 | 0.065130102902216 | 0.869739794195568 | II | |
| | 0.077113760890257 | 0.048690315425316 | 0.312865496004874 | III | |
| 9 – 19 | 0.097135796282799 | 1/3 | – | I | Dunavant (1985) |
| | 0.031334700227139 | 0.489682519198738 | 0.020634961602525 | II | [83] |
| | 0.077827541004774 | 0.437089591492937 | 0.125820817014127 | II | |
| | 0.079647738927210 | 0.188203535619033 | 0.623592928761935 | II | |
| | 0.025577675658698 | 0.044729513394453 | 0.910540973211095 | II | |
| | 0.043283539377289 | 0.221962989160766 | 0.036838412054736 | III | |

**Table 5.3.3** Integration methods for tetrahedra

| | | Points | | | Symmetry | |
|---|---|---|---|---|---|---|
| Order – # points | Weight | $\alpha$ | $\beta$ | $\gamma$ | Type | Notes |
| 1–1 | 1 | 1/4 | – | – | I | Centroid method |
| 2–4 | 1/4 | 1/6 | 2/3 | – | II | |
| 3–5 | −4/5 | 1/4 | – | – | I | |
| | 9/20 | 1/6 | 1/2 | – | II | |
| 5–17 | 0.188418 556736 5411 | 1/4 | – | – | I | Jinyun (1984) |
| | 0.067038 583726 04275 | 0.089454 364014 12733 | 0.731636 907957 6179 | – | II | [266] |
| | 0.045285 592363 27399 | 0.421439 431066 2522 | 0.024540 037929 0300 | 0.132581 099938 4657 | IV | |
| 7–31 | 0.018264 223466 108793 9 | 1/4 | – | – | I | Keast (1986) |
| | 0.009700 176366 842967 0 | 1/2 | 0 | – | III | [141] |
| | 0.010599 941524 414160 9 | 0.078213 192330 318654 9 | 0.765360 423009 044044 | – | II | |
| | −0.062517 740114 329949 4 | 0.121843 216663 904411 | 0.634470 350008 286765 | – | II | |
| | 0.004891 425263 073536 5 | 0.332539 164446 420554 | 0.002382 506660 738345 5 | – | II | |
| | 0.027557 319223 985091 7 | 0.1 | 0.2 | 0.6 | IV | |

in the tables below are symmetric methods, so it is not necessary to list all the points. Instead, we list one point in a set, and the other points in the set are generated by applying the relevant symmetries. In terms of barycentric coordinates, applying the relevant symmetries simply means permuting the coordinates.

The different types of symmetric sets of points for a triangle are listed in Table 5.3.1.

The methods have the form

$$\int_K f(\boldsymbol{x})\,d\boldsymbol{x} \approx |K| \sum_{i=1}^{m} w_i\, f(\boldsymbol{x}_i);$$

again, the weights are not repeated as the weight $w_i$ is the same for each symmetric set of points $\boldsymbol{x}_i$. The methods for triangles are shown in Table 5.3.2, and the methods for tetrahedra are shown in Table 5.3.3.

**Exercises.**

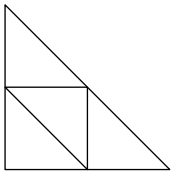
(1) Show that for any region $R \subset \mathbb{R}^n$, if $\overline{\boldsymbol{x}}_R = \int_R \boldsymbol{x}\,d\boldsymbol{x} / \int_R d\boldsymbol{x}$ is the centroid of $R$, then the one-point rule $\int_R f(\boldsymbol{x})\,d\boldsymbol{x} \approx \mathrm{vol}_n(R)\, f(\overline{\boldsymbol{x}}_R)$ is exact for all linear functions $f$. Note that $\mathrm{vol}_n(R) = \int_R d\boldsymbol{x}$ is the $n$-dimensional volume of $R$.

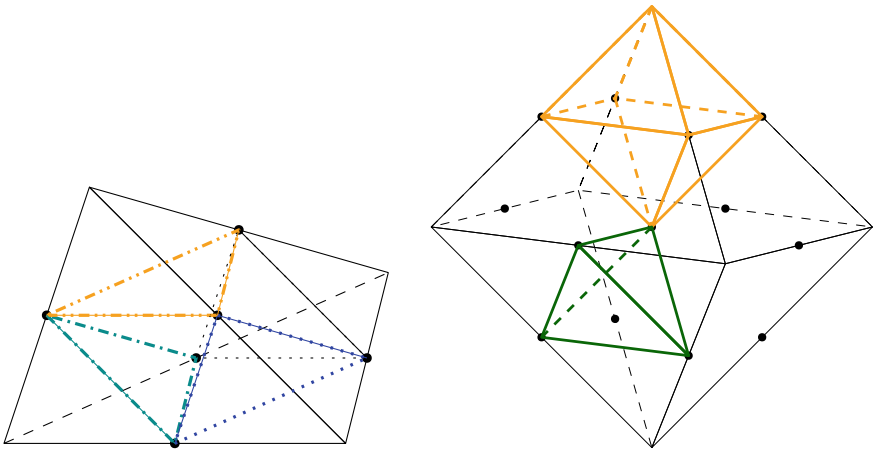
(2) Show that the symmetric rule for a triangle $K$ with vertices $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ and centroid $\boldsymbol{v}_0 = \frac{1}{3}(\boldsymbol{v}_1 + \boldsymbol{v}_2 + \boldsymbol{v}_3)$, the rule $\int_K f(\boldsymbol{x})\,d\boldsymbol{x} \approx \mathrm{area}(K)\left[\frac{3}{4} f(\boldsymbol{v}_0) + \frac{1}{12}\,(f(\boldsymbol{v}_1) + f(\boldsymbol{v}_2) + f(\boldsymbol{v}_3))\right]$ is exact for all quadratic $f$.

(3) Let $\widehat{K}$ be the triangle with vertices $(0, 0)$, $(1, 0)$, and $(0, 1)$. Suppose a symmetric integration rule $\int_{\widehat{K}} f(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{j=1}^{n} w_j\, f(\boldsymbol{v}_j)$ is exact for all linear functions and also for $f(x, y) = x^2$; show that the rule is exact for *all* quadratic functions. [**Hint:** Consider the action of the affine symmetries $x \leftrightarrow y$ and $(x, y) \mapsto (y, 1 - x - y) \mapsto (1 - x - y, x)$ of $f(x, y) = x^2$. Show that apart from linear terms, this can generate all quadratic monomials $x^2$, $y^2$, and $xy$.]

(4) Following the previous Exercise, consider $\widehat{K}$ to be the unit tetrahedron with vertices $\mathbf{0}$ and $\boldsymbol{e}_j$, $j = 1, 2, 3$, in $\mathbb{R}^3$. Suppose a symmetric integration rule $\int_{\widehat{K}} f(\boldsymbol{x})\,d\boldsymbol{x} \approx \sum_{j=1}^{n} w_j\, f(\boldsymbol{v}_j)$ is exact for all linear functions on $\widehat{K}$. Suppose also that it is exact for $f(x, y, z) = x^2$ and for $f(x, y, z) = xy$. Show that it is therefore exact for *all* quadratics on $\mathbb{R}^3$. [**Hint:** Use the affine symmetries $x \leftrightarrow y$, $x \leftrightarrow z$, and $y \leftrightarrow z$; then use the symmetry $(x, y, z) \mapsto (y, z, 1 - x - y - z)$.]

(5) Using the decomposition of a triangle into congruent sub-triangles below, we estimate the integral of $\int_{\widehat{K}} f(x, y)\,dx\,dy$ for $f(x, y) = e^{x-y}\cos(x + 2y)$ where $\widehat{K}$ is the standard unit triangle with vertices $(0, 0)$, $(1, 0)$, and $(0, 1)$. We do this by subdividing $\widehat{K}$ $m$ times in this way recursively for $m = 1, 2, \ldots, 6$, and applying the centroid rule for each triangle in the final subdivision. Note that after $m$ subdivisions, there are $4^m$ triangles.





(A) Tetrahedron (B) Octahedron

**Fig. 5.3.1** Tetrahedral–octahedral decompositions

Plot the error in the computed integrals against $m$, the number of subdivisions. How does the error relate to the number of function evaluations?

(6) Unfortunately, it is not possible to subdivide a tetrahedron into 8 congruent sub-tetrahedra. However, it is possible to have a mutually recursive decomposition of a tetrahedron into 4 tetrahedron and an octahedron, and of an octahedron into 6 octahedra and 8 tetrahedra. This decomposition is described in [229]. In this Exercise, we will verify that if we begin with either regular solid, then the resulting subdivision is into regular solids with the edge lengths halved. Applying affine transformations ensures that the resulting subdivisions of any given tetrahedron will be into congruent tetrahedra and congruent octahedra. Figure 5.3.1 illustrates the decompositions.

   (a) The decomposition of the tetrahedron creates four sub-tetrahedra by joining each vertex of the original tetrahedron to the midpoints of the incident edges. Assuming that the original tetrahedron is regular (all edge lengths are the same, so each face is an equilateral triangle), show that all edge lengths of the sub-tetrahedra are half the edge lengths of the original tetrahedron. Show that the solid remaining after removing these four tetrahedra is a regular octahedron.

   (b) To decompose a regular octahedron, join each vertex of the original octahedron to the midpoints of the incident edges and the centroid. Show that this creates six regular sub-octahedra. Show that the remainder of the original octahedron forms eight regular tetrahedra formed by joining the edge midpoints of each face to the centroid of the original octahedron.

(7) Test the methods listed in Table 5.3.2 for integration over triangles, by applying them to the integral $\int_{\widehat{K}} e^{x-y} \cos(x+2y)\, dx\, dy$ where $\widehat{K}$ is the standard unit triangle with vertices at (0, 0), (1, 0), and (0, 1).
(8) Test the methods listed in Table 5.3.3 for integration over tetrahedra, by applying them to the integral $\int_{\widehat{K}} e^{x-y} \cos(x+2y-z)\, dx\, dy\, dz$ where $\widehat{K}$ is the standard unit tetrahedron with vertices at (0, 0, 0), (1, 0, 0), (0, 1, 0), and (0, 0, 1).
(9) Verify that Radon's method in Table 5.3.2 is exact for all polynomials $p(x, y)$ of degree $\leq 5$. Use symmetry where possible to reduce the number of equations to check.

## 5.4 High-Dimensional Integration

If the dimension $d$ is larger than three or four, standard integration methods lose their effectiveness. Integrating over a hypercube $[a, b]^d$ with $n$ evaluation points on each co-ordinate gives methods needing $n^d$ evaluation points. This is strongly exponential in the dimension. In many applications, particularly in connection with statistical and machine learning applications, the dimension $d$ is much more than five. The dimension $d$ can be hundreds to thousands, or even more. In these cases, no tensor product method can be successful.

For these high-dimensional integration problems, we need a different approach. The approach taken will often appear to be more "statistical" in flavor. The analysis of convergence of high-dimensional integration methods should not focus on so much on the differentiability of a function, but more on variance and variance reduction strategies.

### *5.4.1 Monte Carlo Integration*

Basic concepts, definitions, and theorems of probability theory and random variables are discussed in Chapter 7, especially Section 7.1. Of particular importance, here are the concepts of random variables (usually denoted by capital letters), probabilities of events $\Pr[X \in E]$, expectations of a random variable ($\mathbb{E}[X]$), variance of a random variable ($\mathbb{V}\mathrm{ar}[X]$), and independence of a pair of random variables.

Suppose the random variable $\boldsymbol{X}$ takes values in $\mathbb{R}^d$ and has a probability density function $p(\boldsymbol{x})$, so that the probability that $\boldsymbol{X} \in E$ is $\Pr[\boldsymbol{X} \in E] = \int_E p(\boldsymbol{x})\, d\boldsymbol{x}$. Then

$$\int_{\mathbb{R}^d} p(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x} = \mathbb{E}[f(\boldsymbol{X})]\,. \tag{5.4.1}$$

Alternatively, probability distributions can represented by measures: $\Pr[\boldsymbol{X} \in E] = \pi(E)$ for all measurable $E$. This relationship between the random variable $\boldsymbol{X}$ and the measure $\pi$ is denoted $\boldsymbol{X} \sim \pi$. If $\pi$ has a density function $p(\boldsymbol{x})$, then we can also

write $\boldsymbol{X} \sim p$ as well. The expectation can be written as an integral with respect to the probability measure $\pi$:

$$\int_{\mathbb{R}^d} f(\boldsymbol{x})\,\pi(d\boldsymbol{x}) = \mathbb{E}\left[f(\boldsymbol{X})\right]. \tag{5.4.2}$$

If we take independent samples $\boldsymbol{X}_i$, $i = 1, 2, \ldots, N$, from the same probability distribution ($\boldsymbol{X}_i \sim \pi$), then the Law of Large Numbers (7.1.16) implies that with probability one

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{X}_i) = \mathbb{E}\left[f(\boldsymbol{X})\right]. \tag{5.4.3}$$

The question we need to ask is: "How quickly?"

Provided $\mathbb{E}\left[f(\boldsymbol{X})^2\right]$ is finite, we can give a simple answer. For a random variable $Y$, the variance of $Y$ is

$$\mathbb{V}\mathrm{ar}\left[Y\right] = \mathbb{E}\left[(Y - \mathbb{E}\left[Y\right])^2\right] = \mathbb{E}\left[Y^2\right] - \mathbb{E}\left[Y\right]^2, \tag{5.4.4}$$

which is bounded above by $\mathbb{E}\left[Y^2\right]$ and below by zero. If $Y$ and $Z$ are independent random variables, then

$$\begin{aligned} \Pr\left[Y \in E \text{ and } Z \in F\right] &= \Pr\left[Y \in E\right] \cdot \Pr\left[Z \in F\right], \\ \mathbb{E}\left[Y \cdot Z\right] &= \mathbb{E}\left[Y\right] \cdot \mathbb{E}\left[Z\right], \qquad \text{and} \\ \mathbb{V}\mathrm{ar}\left[Y + Z\right] &= \mathbb{V}\mathrm{ar}\left[Y\right] + \mathbb{V}\mathrm{ar}\left[Z\right], \end{aligned}$$

by Lemma 7.2 and Theorem 7.1.

Thus the average

$$A_n = \frac{1}{n} \sum_{i=1}^{n} f(\boldsymbol{X}_i)$$

with the $\boldsymbol{X}_i$ independent with identical distribution $\boldsymbol{X}_i \sim \pi$ we see that

$$\mathbb{E}\left[A_n\right] = \mathbb{E}\left[f(\boldsymbol{X}_i)\right] = \int_{\mathbb{R}^d} f(\boldsymbol{x})\,\pi(d\boldsymbol{x}) = \mu.$$

Note that if $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent random variables, $f(\boldsymbol{X})$ and $g(\boldsymbol{Y})$ are also independent:

$$\begin{aligned} \Pr\left[f(\boldsymbol{X}) \in E \;\&\; f(\boldsymbol{Y}) \in F\right] &= \Pr\left[\boldsymbol{X} \in f^{-1}(E) \;\&\; \boldsymbol{Y} \in g^{-1}(F)\right] \\ &= \Pr\left[\boldsymbol{X} \in f^{-1}(E)\right] \cdot \Pr\left[\boldsymbol{Y} \in g^{-1}(F)\right] \\ &= \Pr\left[f(\boldsymbol{X}) \in E\right] \cdot \Pr\left[g(\boldsymbol{Y}) \in F\right]. \end{aligned}$$

To see the rate of convergence, we note that

$$\begin{aligned}\mathbb{V}\mathrm{ar}\left[A_n\right] &= \mathbb{V}\mathrm{ar}\left[\frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{X}_i)\right] \\ &= \frac{1}{n^2}\sum_{i=1}^{n}\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}_i)\right] \\ &= \frac{1}{n}\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}_1)\right].\end{aligned}$$

Let $\sigma = \mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}_1)\right]^{1/2}$. Then $\mathbb{V}\mathrm{ar}\left[A_n\right] = \sigma^2/n$. Chebyshev's inequality (7.1.14) then implies that for any $k > 0$,

$$\Pr\left[|A_n - \mu| \geq k\,\sigma/\sqrt{n}\right] \leq 1/k^2.$$

Thus with high probability, $|A_n - \mu| = \mathcal{O}(n^{-1/2})$. The hidden constant in this $\mathcal{O}$ expression is $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}_1)\right]^{1/2}$. Obtaining more accurate estimates involves finding ways to reframe the problem but with a smaller value of $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}_1)\right]$.

We can estimate $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X})\right]$ in terms of integrals via

$$\begin{aligned}\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X})\right] &= \mathbb{E}\left[f(\boldsymbol{X})^2\right] - \mathbb{E}\left[f(\boldsymbol{X})\right]^2 \\ &= \int_{\mathbb{R}^d}\int_{\mathbb{R}^d}\frac{1}{2}\left(f(\boldsymbol{x}) - f(\boldsymbol{y})\right)^2\,\pi(d\boldsymbol{x})\,\pi(d\boldsymbol{y}) \\ &= \frac{1}{2}\mathbb{E}\left[\left(f(\boldsymbol{X}) - f(\boldsymbol{Y})\right)^2\right] \qquad (5.4.5)\end{aligned}$$

where $\boldsymbol{X}$ and $\boldsymbol{Y}$ are independent random variables distributed identically: $\boldsymbol{X},\ \boldsymbol{Y} \sim \pi$. A quick and clear conclusion from this is that $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X})\right] \leq \|f\|_\infty^2$. Some improvements on this basic bound are easy to find: for any constant $c$, $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X})\right] = \mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X}) - c\right] \leq \|f - c\|_\infty^2$. Choosing $c = \frac{1}{2}\left(\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \max_{\boldsymbol{x}} f(\boldsymbol{x})\right)$ we get $\mathbb{V}\mathrm{ar}\left[f(\boldsymbol{X})\right] \leq \frac{1}{4}\left(\max_{\boldsymbol{x}} f(\boldsymbol{x}) - \min_{\boldsymbol{x}} f(\boldsymbol{x})\right)^2$.

#### 5.4.1.1 Variance Reduction: Importance Sampling and Rare Events

Suppose that $\boldsymbol{X} \sim p$ for a probability density function $p(\boldsymbol{x})$. Then

$$\mathbb{E}_{\boldsymbol{X}\sim p}\left[f(\boldsymbol{X})\right] = \int_{\mathbb{R}^d} p(\boldsymbol{x})\,f(\boldsymbol{x})\,d\boldsymbol{x}.$$

We can sample $\boldsymbol{X}$ differently, according to another density function:

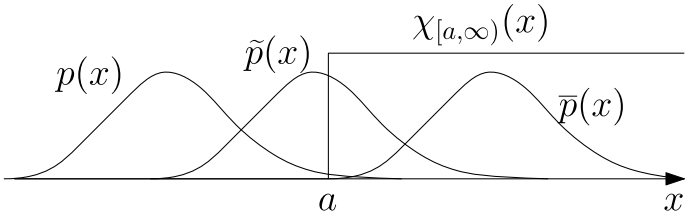



**Fig. 5.4.1** Rare event example of Monte Carlo integration

$$\begin{aligned}\int_{\mathbb{R}^d} p(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x} &= \int_{\mathbb{R}^d} \widetilde{p}(\boldsymbol{x})\, \frac{p(\boldsymbol{x})}{\widetilde{p}(\boldsymbol{x})}\, f(\boldsymbol{x})\, d\boldsymbol{x} \\ &= \mathbb{E}_{\boldsymbol{X}\sim\widetilde{p}}\left[\frac{p(\boldsymbol{X})}{\widetilde{p}(\boldsymbol{X})}\, f(\boldsymbol{X})\right].\end{aligned}$$

By choosing $\widetilde{p}$ appropriately, we can reduce the variance. Ideally, we would have $(p(\boldsymbol{x})/\widetilde{p}(\boldsymbol{x}))\, f(\boldsymbol{x})$ constant to minimize the variance of $(p(\boldsymbol{x})/\widetilde{p}(\boldsymbol{x}))\, f(\boldsymbol{x})$. To apply this technique we need to know what the ratio $p(\boldsymbol{x})/\widetilde{p}(\boldsymbol{x})$ is.

Consider estimating a rare event: $X \geq a$ where $X$ is a random variable with probability density function $p(x)$ whose mode is far below $a$, as illustrated in Figure 5.4.1. While the example is written as a one-dimensional integral, the random variable $X$ could be the result of a large computation involving many basic random variables $X_1, X_2, \ldots, X_d$; treating the problem in its original form could be a very high dimensional integration problem. Here, we consider the problem as a one-dimensional integration problem to better illustrate certain aspects of the problem.

The probability density function of the random variable $X$ is $p(x)$, so $\widehat{p} := \Pr[X \geq a] = \mathbb{E}\left[\chi_{[a,\infty)}(X)\right] = \int_a^\infty p(x)\, dx$. Note that $\chi_E(x) = 1$ if $x \in E$ and zero if $x \notin E$. The variance of $\chi_{[a,\infty)}(X)$ for a single sample is $\mathbb{V}\mathrm{ar}\left[\chi_{[a,\infty)}(X)\right] = \widehat{p}(1-\widehat{p})$ so the standard deviation is $\sqrt{\widehat{p}(1-\widehat{p})} \gg \widehat{p}$ if $\widehat{p}$ is small. Thus applying the Monte Carlo integration method will require many samples. If we instead sample $X \sim \widetilde{p}$ we have $\widehat{p} = \mathbb{E}_{X\sim\widetilde{p}}\left[(p(X)/\widetilde{p}(X))\, \chi_{[a,\infty)}(X)\right]$. The variance of this method is given by

$$\mathbb{V}\mathrm{ar}_{X\sim\widetilde{p}}\left[\frac{p(X)}{\widetilde{p}(X)}\chi_{[a,\infty)}(X)\right] = \mathbb{E}_{X\sim\widetilde{p}}\left[\frac{p(X)^2}{\widetilde{p}(X)^2}\chi_{[a,\infty)}(X)^2\right] - \widehat{p}^2.$$

We do not want $\mathbb{E}_{X\sim\widetilde{p}}\left[\left(p(X)^2/\widetilde{p}(X)^2\right)\chi_{[a,\infty)}(X)^2\right]$ to be a large multiple of $\widehat{p}^2$. Noting that

$$\mathbb{E}_{X\sim\widetilde{p}}\left[\frac{p(X)^2}{\widetilde{p}(X)^2}\chi_{[a,\infty)}(X)^2\right] = \mathbb{E}_{X\sim\widetilde{p}}\left[\frac{p(X)^2}{\widetilde{p}(X)^2} \mid X \geq a\right] \Pr_{X\sim\widetilde{p}}[X \geq a]$$

we see that we want $p(x)/\widetilde{p}(x)$ to be small for $x \geq a$. So we should aim to have $\widetilde{p}$ more-or-less as shown in Figure 5.4.1: $\Pr_{X\sim\widetilde{p}}[X \geq a] \approx \frac{1}{2}$, but $p(x)/\widetilde{p}(x)$ is small (perhaps $\mathcal{O}(\widehat{p})$). Using $\overline{p}(x)$ of Figure 5.4.1 is probably counter-productive:

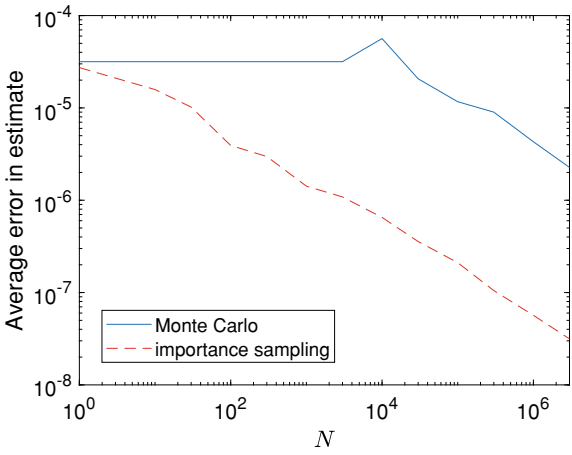



**Fig. 5.4.2** Rare event probability estimation error, comparing standard Monte Carlo with importance sampling

$p(x)/\overline{p}(x)$ for $x$ near $a$ does not appear to be small, and would result in large variance. Even worse, the *probability* of getting $X \approx a$ for $X \sim \overline{p}$ would be small, meaning that a simulation might give apparently accurate results until the rare event (in the simulation) of getting $X \approx a$ occurs.

As an example, consider the "rare events" problem of estimating $\Pr[X \geq a]$ where $X \sim \text{Normal}(0, 1)$. The standard Monte Carlo approach is the take independent samples from the Normal(0, 1) distribution, and we count the average number of times $X \geq a$ occurs. However, the variance of $\chi_{[a,\infty)}(X)$ is large compared to $\Pr[X \geq a]^2$ meaning that the number of samples is likely to be large before reasonable estimates can be obtained. The importance sampling approach is to estimate $\mathbb{E}_{X\sim\widetilde{p}}\left[(p(X)/\widetilde{p}(X))\,\chi_{[a,\infty)}(X)\right]$. Since we know the probability density functions for these probability distributions, we can directly compute $p(x)/\widetilde{p}(x) = \exp(-ax + a^2/2)$. Figure 5.4.2 shows result for this where $a = 4$ so that $\Pr[X \geq a] \approx 3.167 \times 10^{-4}$. The error estimates are averaged over 10 trials. It shows that the error in the estimate using standard Monte Carlo method has 100% error until $N > 10^4$. In fact, the probability estimate using standard Monte Carlo for these trials is zero for $N \leq 3 \times 10^3$. Importance sampling, when done right, can give reasonably accurate results with a much smaller number of samples.

### 5.4.1.2 Variance Reduction: Antithetic Variates

Another approach to reducing the variance of the estimate is to use *antithetic variates*. This applies where $p(\boldsymbol{x}) = p(-\boldsymbol{x})$ for all $\boldsymbol{x}$; we can then take samples $\boldsymbol{X}_i$ from the probability distribution and estimate

$$\mathbb{E}[f(\boldsymbol{X})] \approx \frac{1}{2N}\sum_{i=1}^{N}[f(\boldsymbol{X}_i) + f(-\boldsymbol{X}_i)] \tag{5.4.6}$$

with $\boldsymbol{X}_i \sim p$ and generated independently. Errors in Monte Carlo type computational results for integrating $\int_0^1 dx/(1+x^2) = \pi/4$ are shown in Figure 5.4.3 where $X \sim$ Uniform$(-\frac{1}{2}, +\frac{1}{2})$ and $f(x) = 1/(1+x^2)$ is evaluated at $\frac{1}{2} + X$ and $\frac{1}{2} - X$.

First note that if $f$ is an affine function, then $\frac{1}{2}(f(\boldsymbol{x}) + f(-\boldsymbol{x})) = f(\boldsymbol{0}) = \int_{\mathbb{R}^d} p(\boldsymbol{z})\, f(\boldsymbol{z})\, d\boldsymbol{z}$ for any $\boldsymbol{x}$. Beyond this, we should look at the variance of the estimates

$$A_N = \frac{1}{2N} \sum_{i=1}^{N} [f(\boldsymbol{X}_i) + f(-\boldsymbol{X}_i)]\,.$$

Since the $\boldsymbol{X}_i$'s are independent, it suffices to estimate $\mathbb{V}\mathrm{ar}\left[\frac{1}{2}(f(\boldsymbol{X}) + f(-\boldsymbol{X}))\right]$ where $\boldsymbol{X} \sim p$. If $Q_X = \frac{1}{2}(\delta(\cdot - \boldsymbol{X}) + \delta(\cdot + \boldsymbol{X}))$ then using the usual inner product

$$\frac{1}{2}(f(\boldsymbol{X}) + f(-\boldsymbol{X})) = (Q_X,\ f) = \int_{\mathbb{R}^d} \overline{Q_X(\boldsymbol{x})}\, f(\boldsymbol{x})\, d\boldsymbol{x},$$

and we want to estimate

$$\mathbb{V}\mathrm{ar}\,[(Q_X,\ f)] = \mathbb{E}\left[(Q_X,\ f)^2\right] - \mathbb{E}\,[(Q_X,\ f)]^2\,.$$

Using Fourier transforms, $(g,\ f) = (2\pi)^{-d}(\mathcal{F}g,\ \mathcal{F}f)$. We first note that

$$\begin{aligned}
\mathbb{E}\,[(Q_X,\ f)] &= (\mathbb{E}\,[Q_X],\ f)\,, \quad \text{but} \\
\mathbb{E}\,[Q_X]\,(\boldsymbol{z}) &= \int_{\mathbb{R}^d} p(\boldsymbol{x}) \frac{1}{2} [\delta(\boldsymbol{z} - \boldsymbol{x}) + \delta(\boldsymbol{z} + \boldsymbol{x})]\, d\boldsymbol{x} \\
&= \frac{1}{2}[p(\boldsymbol{z}) + p(-\boldsymbol{z})] = p(\boldsymbol{z}).
\end{aligned}$$

Thus

$$\mathbb{E}\,[(Q_X,\ f)] = (2\pi)^{-d}\,(\mathcal{F}p,\ \mathcal{F}f)\,.$$

Let $\phi(\boldsymbol{\xi}) = \mathcal{F}p(\boldsymbol{\xi}) = \mathbb{E}\left[e^{-i\boldsymbol{\xi}^T \boldsymbol{X}}\right]$ for $\boldsymbol{X} \sim p$. Note that

$$|\phi(\boldsymbol{\xi})| = \left| \int_{\mathbb{R}^d} e^{-i\boldsymbol{\xi}^T \boldsymbol{x}}\, p(\boldsymbol{x})\, d\boldsymbol{x} \right| \le \int_{\mathbb{R}^d} \left| e^{-i\boldsymbol{\xi}^T \boldsymbol{x}} \right| p(\boldsymbol{x})\, d\boldsymbol{x} = 1 \qquad \text{for all } \boldsymbol{\xi}. \tag{5.4.7}$$

Then, using the fact that $Q_X$ is real,

$$\begin{aligned}
\mathbb{E}\,[(Q_X,\ f)]^2 &= (2\pi)^{-2d} \overline{\int_{\mathbb{R}^d} \overline{\phi(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\xi})\, d\boldsymbol{\xi}} \int_{\mathbb{R}^d} \overline{\phi(\boldsymbol{\eta})}\, \mathcal{F}f(\boldsymbol{\eta})\, d\boldsymbol{\eta} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\, \overline{\phi(\boldsymbol{\xi})}\, \phi(\boldsymbol{\eta})\, d\boldsymbol{\xi}\, d\boldsymbol{\eta}.
\end{aligned}$$

Since $p(\boldsymbol{x})$ is real and $p(\boldsymbol{x}) = p(-\boldsymbol{x})$, $\phi(\boldsymbol{\xi}) = \mathcal{F}p(\boldsymbol{\xi}) = \mathcal{F}p(-\boldsymbol{\xi}) = \overline{\mathcal{F}p(\boldsymbol{\xi})} = \overline{\phi(\boldsymbol{\xi})}$. So

$$\mathbb{E}\left[(Q_{\boldsymbol{X}},\ f)\right]^2 = (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\, \phi(\boldsymbol{\xi})\, \phi(\boldsymbol{\eta})\, d\boldsymbol{\xi}\, d\boldsymbol{\eta}.$$

On the other hand,

$$\begin{aligned}
\mathbb{E}\left[(Q_{\boldsymbol{X}},\ f)^2\right] &= \int_{\mathbb{R}^d} p(\boldsymbol{x})\ (Q_{\boldsymbol{x}},\ f)^2\, d\boldsymbol{x} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} p(\boldsymbol{x})\ (\mathcal{F}Q_{\boldsymbol{x}},\ \mathcal{F}f)^2\, d\boldsymbol{x} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} p(\boldsymbol{x}) \int_{\mathbb{R}^d} \mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\xi})\, \overline{\mathcal{F}f(\boldsymbol{\xi})}\, d\boldsymbol{\xi} \int_{\mathbb{R}^d} \overline{\mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\eta})}\, \mathcal{F}f(\boldsymbol{\eta})\, d\boldsymbol{\eta}\, d\boldsymbol{x} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\, d\boldsymbol{\xi}\, d\boldsymbol{\eta} \int_{\mathbb{R}^d} p(\boldsymbol{x})\, \mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\xi})\, \overline{\mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\eta})}\, d\boldsymbol{x}.
\end{aligned}$$

The inner integral

$$\begin{aligned}
&\int_{\mathbb{R}^d} p(\boldsymbol{x})\, \mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\xi})\, \overline{\mathcal{F}Q_{\boldsymbol{x}}(\boldsymbol{\eta})}\, d\boldsymbol{x} \\
&\quad = \int_{\mathbb{R}^d} p(\boldsymbol{x})\, \frac{1}{2}(e^{-i\boldsymbol{\xi}^T\boldsymbol{x}} + e^{+i\boldsymbol{\xi}^T\boldsymbol{x}})\, \frac{1}{2}(e^{+i\boldsymbol{\eta}^T\boldsymbol{x}} + e^{-i\boldsymbol{\eta}^T\boldsymbol{x}})\, d\boldsymbol{x} \\
&\quad = \frac{1}{4} \int_{\mathbb{R}^d} p(\boldsymbol{x})\, \left(e^{-i(\boldsymbol{\xi}+\boldsymbol{\eta})^T\boldsymbol{x}} + e^{-i(\boldsymbol{\xi}-\boldsymbol{\eta})^T\boldsymbol{x}} + e^{-i(-\boldsymbol{\xi}+\boldsymbol{\eta})^T\boldsymbol{x}} + e^{-i(-\boldsymbol{\xi}-\boldsymbol{\eta})^T\boldsymbol{x}}\right) d\boldsymbol{x} \\
&\quad = \frac{1}{4}\left(\phi(\boldsymbol{\xi}+\boldsymbol{\eta}) + \phi(\boldsymbol{\xi}-\boldsymbol{\eta}) + \phi(-\boldsymbol{\xi}+\boldsymbol{\eta}) + \phi(-\boldsymbol{\xi}-\boldsymbol{\eta})\right) \\
&\quad = \frac{1}{2}\left(\phi(\boldsymbol{\xi}+\boldsymbol{\eta}) + \phi(\boldsymbol{\xi}-\boldsymbol{\eta})\right), \qquad \text{using } \phi(\boldsymbol{\zeta}) = \phi(-\boldsymbol{\zeta}).
\end{aligned}$$

Combining the integrals gives

$$\begin{aligned}
\mathbb{V}\mathrm{ar}\left[(Q_{\boldsymbol{X}},\ f)\right] &= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\ \times \\
&\qquad \times \left[\frac{1}{2}\left(\phi(\boldsymbol{\xi}+\boldsymbol{\eta}) + \phi(\boldsymbol{\xi}-\boldsymbol{\eta})\right) - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta})\right] d\boldsymbol{\xi}\, d\boldsymbol{\eta} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\, \psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta})\, d\boldsymbol{\xi}\, d\boldsymbol{\eta} \qquad \text{where} \\
\psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta}) &= \frac{1}{2}\left(\phi(\boldsymbol{\xi}+\boldsymbol{\eta}) + \phi(\boldsymbol{\xi}-\boldsymbol{\eta})\right) - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta}).
\end{aligned}$$

Since $\phi(\mathbf{0}) = 1$ we can see that $\psi_{AV}(\mathbf{0}, \mathbf{0}) = 0$. Also, since $p(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x}$ and $\int_{\mathbb{R}^d} p(\boldsymbol{x})\, d\boldsymbol{x} = 1$, $\phi(\boldsymbol{\xi}) = \mathcal{F}p(\boldsymbol{\xi})$ is real analytic in $\boldsymbol{\xi}$, and therefore, so is $\psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta})$

in $(\boldsymbol{\xi}, \boldsymbol{\eta})$. In addition, $\phi(\boldsymbol{\xi}) \to 0$ as $\|\boldsymbol{\xi}\|_2 \to \infty$ and $\phi(\boldsymbol{\xi}) = \phi(-\boldsymbol{\xi})$ for all $\boldsymbol{\xi}$, so the Taylor series expansion of $\phi$ about $\boldsymbol{\xi} = \mathbf{0}$ can only contain even powers of $\boldsymbol{\xi}$.

We can bound

$$\mathbb{V}\text{ar}\,[(Q_X,\ f)] \le (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} |\mathcal{F}f(\boldsymbol{\xi})|\ |\mathcal{F}f(\boldsymbol{\eta})|\ |\psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta})|\ d\boldsymbol{\xi}\, d\boldsymbol{\eta}.$$

Unfortunately, $\sup_{\boldsymbol{\xi},\,\boldsymbol{\eta}} \psi(\boldsymbol{\xi}, \boldsymbol{\eta}) = \frac{1}{2}$ (take $\boldsymbol{\xi} = \boldsymbol{\eta}$ and $\|\boldsymbol{\xi}\|_2 \to \infty$).

By comparison, the standard Monte Carlo method would use $Q_X^{(0)} = \delta(\cdot - \boldsymbol{X})$ and we get

$$\begin{aligned}
\mathbb{V}\text{ar}\left[\left(Q_X^{(0)},\ f\right)\right] &= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\ [\phi(\boldsymbol{\xi} - \boldsymbol{\eta}) - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta})]\ d\boldsymbol{\xi}\, d\boldsymbol{\eta} \\
&= (2\pi)^{-2d} \int_{\mathbb{R}^d} \int_{\mathbb{R}^d} \overline{\mathcal{F}f(\boldsymbol{\xi})}\, \mathcal{F}f(\boldsymbol{\eta})\, \psi_{MC}(\boldsymbol{\xi}, \boldsymbol{\eta})\, d\boldsymbol{\xi}\, d\boldsymbol{\eta}, \qquad \text{where} \\
\psi_{MC}(\boldsymbol{\xi}, \boldsymbol{\eta}) &= \phi(\boldsymbol{\xi} - \boldsymbol{\eta}) - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta}).
\end{aligned}$$

For $\boldsymbol{\zeta} \approx \mathbf{0}, \phi(\boldsymbol{\zeta}) = \phi(-\boldsymbol{\zeta}) = 1 - \boldsymbol{\zeta}^T B \boldsymbol{\zeta} + \mathcal{O}(\|\boldsymbol{\zeta}\|_2^4)$ for a symmetric, positive definite matrix $B$ using (5.4.7) and $\phi(\boldsymbol{\zeta}) = \phi(-\boldsymbol{\zeta}) = \overline{\phi(\boldsymbol{\zeta})}$. Then

$$\begin{aligned}
\psi_{MC}(\boldsymbol{\xi}, \boldsymbol{\eta}) &= \phi(\boldsymbol{\xi} - \boldsymbol{\eta}) - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta}) \\
&= 1 - (\boldsymbol{\xi} - \boldsymbol{\eta})^T B(\boldsymbol{\xi} - \boldsymbol{\eta}) - (1 - \boldsymbol{\xi}^T B\boldsymbol{\xi})(1 - \boldsymbol{\eta}^T B\boldsymbol{\eta}) + \mathcal{O}(\|\boldsymbol{\xi}\|_2^4 + \|\boldsymbol{\eta}\|_2^4) \\
&= 2\boldsymbol{\xi}^T B\boldsymbol{\eta} = \mathcal{O}(\|\boldsymbol{\xi}\|_2^2 + \|\boldsymbol{\eta}\|_2^2), \qquad \text{while} \\
\psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta}) &= \frac{1}{2}\left[\phi(\boldsymbol{\xi} + \boldsymbol{\eta}) + \phi(\boldsymbol{\xi} - \boldsymbol{\eta})\right] - \phi(\boldsymbol{\xi})\phi(\boldsymbol{\eta}) \\
&= \frac{1}{2}\left[2 - (\boldsymbol{\xi} - \boldsymbol{\eta})^T B(\boldsymbol{\xi} - \boldsymbol{\eta}) - (\boldsymbol{\xi} + \boldsymbol{\eta})^T B(\boldsymbol{\xi} + \boldsymbol{\eta})\right] \\
&\qquad - (1 - \boldsymbol{\xi}^T B\boldsymbol{\xi})(1 - \boldsymbol{\eta}^T B\boldsymbol{\eta}) + \mathcal{O}(\|\boldsymbol{\xi}\|_2^4 + \|\boldsymbol{\eta}\|_2^4) \\
&= \mathcal{O}(\|\boldsymbol{\xi}\|_2^4 + \|\boldsymbol{\eta}\|_2^4).
\end{aligned}$$

This means that for $\boldsymbol{\xi},\ \boldsymbol{\eta} \approx \mathbf{0}, |\psi_{AV}(\boldsymbol{\xi}, \boldsymbol{\eta})|$ is generally much smaller than $|\psi_{MC}(\boldsymbol{\xi}, \boldsymbol{\eta})|$. Thus with smooth functions $f$, $\mathcal{F}f(\boldsymbol{\xi}) \to 0$ rapidly as $\|\boldsymbol{\xi}\|_2 \to \infty$, and antithetic variates should give much lower variances.

**Example 5.3** Figure 5.4.3 shows the average of the absolute value of the difference between the integral estimate and the exact value of $\int_0^1 dx/(1 + x^2) = \pi/4$. The average was taken over 100 trials, using Matlab's Mersenne Twister random number generator with seed 32451098. The value of $N$ is the number of function evaluations used. The control variate method, described in the next section, is used with $\varphi(x) = 1 - x/2$. The slopes of the lines are estimated as between $-0.45$ and $-0.47$; theoretically, they should be $-1/2$ indicating an error of $\mathcal{O}(N^{-1/2})$. The small discrepancy is likely due to the pseudo-random variation in the precise values computed.

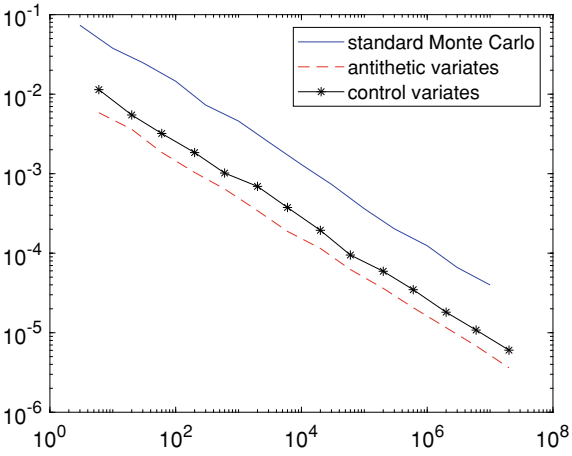



**Fig. 5.4.3** Errors in standard Monte Carlo integration vs antithetic variates vs control variates method for $\int_0^1 dx/(1+x^2)$

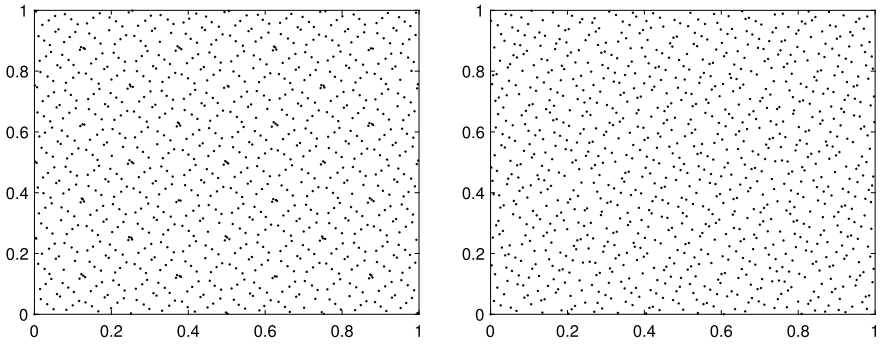
#### 5.4.1.3 Variance Reduction: Control Variates

Another way of reducing the variance is to subtract a function $\varphi$ from $f$ where $\mathbb{E}[\varphi(\boldsymbol{X})] = \int_{\mathbb{R}^d} p(\boldsymbol{x})\,\varphi(\boldsymbol{x})\,d\boldsymbol{x}$ is known. Then $\mathbb{E}[f(\boldsymbol{X})] = \mathbb{E}[f(\boldsymbol{X}) - \varphi(\boldsymbol{X})] + \mathbb{E}[\varphi(\boldsymbol{X})]$ and it is $\mathbb{E}[f(\boldsymbol{X}) - \varphi(\boldsymbol{X})]$ that is estimated using the Monte Carlo method. If $\varphi$ is chosen to also be a good approximation to $f$ over $\{\boldsymbol{x} \mid p(\boldsymbol{x}) > 0\}$, then the variance can be greatly reduced, as can be seen from (5.4.5): $\mathbb{V}\mathrm{ar}[f(\boldsymbol{X}) - \varphi(\boldsymbol{X})] \leq \frac{1}{4}\|f - \varphi\|_\infty^2$. Provided $f$ is smooth, we can use an appropriate interpolation or approximation scheme, such as radial basis functions (see Section 4.4), which can use scattered interpolation points.

### *5.4.2 Quasi-Monte Carlo Methods*

Quasi-Monte Carlo methods [47, Secs. 5 & 6] aim to achieve integration errors of size $\mathcal{O}((\ln n)^m/n)$ rather than $\mathcal{O}(1/\sqrt{n})$ as the number of function evaluations $n \to \infty$. Quasi-Monte Carlo methods replace random (and pseudo-random) number sequences with deterministic sequences that have better uniformity properties than random (or pseudo-random) number generators. Two of the better known quasi-random sequences are the Halton and Sobol' sequences. These sequences were first discovered by number theorists but have found application in high-dimensional numerical integration.

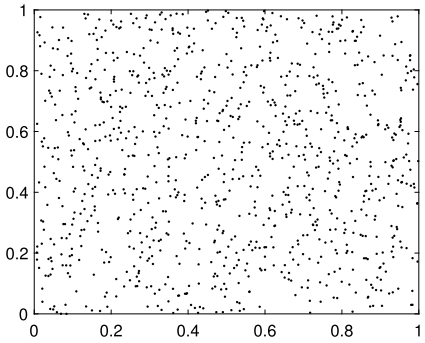
Sobol' and Halton sets are illustrated in two dimensions by Figure 5.4.4. In Figure 5.4.4, we show 1000 points from two-dimensional Sobol', Halton, and pseudo-random (using Matlab's MersenneTwister generator).



(A) Sobol′ sequence



(B) Halton sequence



(C) Pseudo-random sequence

**Fig. 5.4.4** Sobol', Halton, and pseudo-random points in two dimensions

It is evident from Figure 5.4.4 that the quasi-random points are generally "more regularly spaced" than the pseudo-random points. This is the essential nature of quasi-random sequences.

The essential measure of the "quality" of a quasi-random sequence is *discrepancy*. Typically this is measured by how well they estimate the volume of rectangular regions. Let $\text{rect}(\boldsymbol{a}, \boldsymbol{b}) = \{\, \boldsymbol{x} \mid a_i \le x_i \le b_i \text{ for all } i = 1, 2, \ldots, d \,\}$ be the rectangular solid with opposite vertices $\boldsymbol{a}$ and $\boldsymbol{b} \in \mathbb{R}^d$. Let $I^d = \text{rect}(\boldsymbol{0}, \boldsymbol{e})$ where $\boldsymbol{e} = [1,\ 1,\ \ldots,\ 1]^T \in \mathbb{R}^d$ be the unit cube in $\mathbb{R}^d$. If we have a sequence $\boldsymbol{x}_1,\ \boldsymbol{x}_2,\ \boldsymbol{x}_3,\ \ldots \in I^d$ then we can estimate the $d$-dimensional volume of a region $S$, $\text{vol}_d(S)$, by counting the number of the points $\boldsymbol{x}_i \in S$:

$$\text{vol}_d(S) \approx \frac{|\{i \mid 1 \le i \le N \ \&\ \boldsymbol{x}_i \in S\}|}{N} \qquad \text{for large } N;$$

$$R_N(S) = \left| \text{vol}_d(S) - \frac{|\{i \mid 1 \le i \le N \ \&\ \boldsymbol{x}_i \in S\}|}{N} \right|.$$

The discrepancy for the sequence $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots \in I^d$ is the error in this estimate averaged over all rectangles $\mathrm{rect}(\mathbf{0}, \boldsymbol{b})$ with $\boldsymbol{b} \in I^d$:

$$D_N^* = \sup_{\boldsymbol{b} \in I^d} R_N(\mathrm{rect}(\mathbf{0}, \boldsymbol{b})), \tag{5.4.8}$$

$$T_N^* = \left[ \int_{I^d} R_N(\mathrm{rect}(\mathbf{0}, \boldsymbol{b}))^2 \, d\boldsymbol{b} \right]^{1/2}. \tag{5.4.9}$$

We can estimate the integral $\int_{I^d} f(\boldsymbol{x}) \, d\boldsymbol{x} \approx (1/N) \sum_{i=1}^N f(\boldsymbol{x}_i)$. The error in this estimate can be bounded by the *Koksma–Hlawka inequality*:

**Theorem 5.4** *If $f : I^d \to \mathbb{R}$ is smooth, then the error in the integral*

$$\varepsilon_N[f] := \left| \int_{I^d} f(\boldsymbol{x}) \, d\boldsymbol{x} - (1/N) \sum_{i=1}^N f(\boldsymbol{x}_i) \right|$$

*is bounded by*

$$\varepsilon_N[f] \le D_N^* \, V_d[f] \quad \textit{where} \tag{5.4.10}$$

$$V_d[f] = \int_{I^d} \left| \frac{\partial^d f}{\partial x_1 \partial x_2 \cdots \partial x_d}(\boldsymbol{x}) \right| d\boldsymbol{x} + \sum_{j=1}^d V_{d-1}\left[ f \mid_{x_j = 1} \right]. \tag{5.4.11}$$

*Note that $V_0[f] = 0$.*

The quantity (5.4.11) is called the *Hardy–Krause variation*.

***Proof*** Note that

$$\int_{I^d} f(\boldsymbol{x}) \, d\boldsymbol{x} - (1/N) \sum_{i=1}^N f(\boldsymbol{x}_i) = \int_{I^d} \left[ 1 - \frac{1}{N} \sum_{i=1}^N \delta(\boldsymbol{x} - \boldsymbol{x}_i) \right] f(\boldsymbol{x}) \, d\boldsymbol{x}$$

where $\delta(\boldsymbol{z})$ is the Dirac $\delta$-function (actually a distribution; see Appendix §A.2.1). Note that for $g : [0, 1] \to \mathbb{R}$ for $0 \le u \le 1$,

$$\int_0^1 \delta(x - u) \, g(x) \, dx = g(u) = g(1) - \int_u^1 g'(x) \, dx = g(1) - \int_0^1 H(x - u) \, g'(x) \, dx$$

where $H : \mathbb{R} \to \mathbb{R}$ is the Heaviside function: $H(z) = 1$ if $z > 0$ and $H(z) = 0$ if $z < 0$. For $g : [0, 1]^2 \to \mathbb{R}$,

$$\begin{aligned}
&\int_0^1 \int_0^1 \delta(x_1 - u_1)\delta(x_2 - u_2) g(x_1, x_2)\, dx_1\, dx_2 \\
&\quad = \int_0^1 \delta(x_1 - u_1) \int_0^1 \delta(x_2 - u_2) g(x_1, x_2)\, dx_2\, dx_1 \\
&\quad = \int_0^1 \delta(x_2 - u_2) g(1, x_2)\, dx_2 - \int_0^1 H(x_1 - u_1) \int_0^1 \delta(x_2 - u_2) \frac{\partial g}{\partial x_1}(x_1, x_2)\, dx_2\, dx_1.
\end{aligned}$$

Writing

$$\int_0^1 \delta(x_2 - u_2) \frac{\partial g}{\partial x_1}(x_1, x_2)\, dx_2 = \frac{\partial g}{\partial x_1}(x_1, 1) - \int_0^1 H(x_2 - u_2) \frac{\partial^2 g}{\partial x_2 \partial x_1}(x_1, x_2)\, dx_2 \text{ and,}$$
$$\int_0^1 \delta(x_2 - u_2) g(1, x_2)\, dx_2 = g(1, 1) - \int_0^1 H(x_2 - u_2) \frac{\partial g}{\partial x_2}(1, x_2)\, dx_2,$$

we get

$$\begin{aligned}
&\int_0^1 \int_0^1 \delta(x_1 - u_1)\delta(x_2 - u_2) g(x_1, x_2)\, dx_1\, dx_2 \\
&\quad = g(1, 1) - \int_0^1 H(x_2 - u_2) \frac{\partial g}{\partial x_2}(1, x_2)\, dx_2 \\
&\qquad - \int_0^1 H(x_1 - u_1) \frac{\partial g}{\partial x_1}(x_1, 1)\, dx_1 \\
&\qquad + \int_0^1 \int_0^1 H(x_1 - u_1) H(x_2 - u_2) \frac{\partial^2 g}{\partial x_2 \partial x_1}(x_1, x_2)\, dx_2\, dx_1.
\end{aligned}$$

Generalizing to $d$ dimensions, for $\boldsymbol{z} \in \mathbb{R}^d$ we define $H(\boldsymbol{z}) = \prod_{i=1}^d H(z_i)$. Also, for $B \subset \{1, 2, \ldots, d\}$ with $B = \{i_1, i_2, \ldots, i_q\}$,

$$\frac{\partial^{|B|} g}{\partial \boldsymbol{x}^B} = \frac{\partial^q g}{\partial x_{i_1} \partial x_{i_2} \ldots \partial x_{i_d}}, \qquad \text{and}$$
$$F_B = \left\{ \boldsymbol{x} \in [0, 1]^d \mid x_j = 1 \text{ for } j \in B \right\}.$$

Then

(5.4.12)
$$\int_{[0,1]^d} \delta(\boldsymbol{x} - \boldsymbol{u})\, g(\boldsymbol{x})\, d\boldsymbol{x} = \sum_{B \subseteq \{1,2,\ldots,d\}} (-1)^{|B|} \int_{F_B} H(\boldsymbol{x} - \boldsymbol{u}) \frac{\partial^{|B|} g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\, d\boldsymbol{x}.$$

Note that if $B = \emptyset$, we have $\displaystyle\int_{F_B} H(\boldsymbol{x} - \boldsymbol{u}) \frac{\partial^{|B|} g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\, d\boldsymbol{x} = g(1, 1, \ldots, 1)$. Also, if $B = \{1, 2, \ldots, d\}$,

$$\int_{F_B} H(\boldsymbol{x}-\boldsymbol{u})\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x} = \int_{[0,1]^d} H(\boldsymbol{x}-\boldsymbol{u})\frac{\partial^d g}{\partial x_1 \partial x_2 \dots \partial x_d}(\boldsymbol{x})\,d\boldsymbol{x}.$$

On the other hand,

$$\begin{aligned}\int_{[0,1]^d} g(\boldsymbol{x})\,d\boldsymbol{x} &= \int_{[0,1]^d}\int_{[0,1]^d} \delta(\boldsymbol{x}-\boldsymbol{u})\,g(\boldsymbol{x})\,d\boldsymbol{x}\,d\boldsymbol{u} \\ &= \sum_{B\subseteq\{1,2,\dots,d\}} (-1)^{|B|}\int_{F_B}\int_{[0,1]^d} H(\boldsymbol{x}-\boldsymbol{u})\,d\boldsymbol{u}\,\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x}.\end{aligned}$$

Here $\int_{[0,1]^d} H(\boldsymbol{x}-\boldsymbol{u})\,d\boldsymbol{u} = \mathrm{vol}_d\left(\{\boldsymbol{u} \mid \boldsymbol{0}\le \boldsymbol{x}\le \boldsymbol{u}\}\right)$ where "$\boldsymbol{a}\le\boldsymbol{b}$" means "$a_i \le b_i$ for $i = 1, 2, \dots, d$". Since $\{\boldsymbol{u} \mid \boldsymbol{0}\le \boldsymbol{x}\le \boldsymbol{u}\} = \mathrm{rect}(\boldsymbol{0},\boldsymbol{x})$, this gives

$$\int_{[0,1]^d} g(\boldsymbol{x})\,d\boldsymbol{x} = \sum_{B\subseteq\{1,2,\dots,d\}} (-1)^{|B|}\int_{F_B} \mathrm{vol}_d(\mathrm{rect}(\boldsymbol{0},\boldsymbol{x}))\,\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x}. \tag{5.4.13}$$

The formulas (5.4.12, 5.4.13) mean that the integration error for $g$ is

$$\begin{aligned}&\int_{[0,1]^d} g(\boldsymbol{x})\,d\boldsymbol{x} - \frac{1}{N}\sum_{j=1}^N g(\boldsymbol{x}_j) \\ &= \sum_{B\subseteq\{1,2,\dots,d\}} (-1)^{|B|}\int_{F_B} \mathrm{vol}_d(\mathrm{rect}(\boldsymbol{0},\boldsymbol{x}))\,\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x} \\ &\quad + \sum_{B\subseteq\{1,2,\dots,d\}} (-1)^{|B|}\int_{F_B}\int_{[0,1]^d} \frac{1}{N}\sum_{j=1}^N H(\boldsymbol{x}-\boldsymbol{x}_j)\,\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x} \\ &= \sum_{B\subseteq\{1,2,\dots,d\}} (-1)^{|B|}\int_{F_B}\left[\mathrm{vol}_d(\mathrm{rect}(\boldsymbol{0},\boldsymbol{x})) - \frac{1}{N}\sum_{j=1}^N H(\boldsymbol{x}-\boldsymbol{x}_j)\right]\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\,d\boldsymbol{x}.\end{aligned}$$

The quantity $\sum_{j=1}^N H(\boldsymbol{x}-\boldsymbol{x}_j)$ is the number of $j$ where $\boldsymbol{x}\ge\boldsymbol{x}_j$; that is, $\sum_{j=1}^N H(\boldsymbol{x}-\boldsymbol{x}_j) = \left|\left\{j \mid \boldsymbol{x}_j \in \mathrm{rect}(\boldsymbol{0},\boldsymbol{x})\right\}\right|$. The discrepancy

$$D_N^* = \max_{\boldsymbol{x}\in[0,1]^d}\left|\mathrm{vol}_d(\mathrm{rect}(\boldsymbol{0},\boldsymbol{x})) - \frac{1}{N}\left|\left\{j \mid \boldsymbol{x}_j \in \mathrm{rect}(\boldsymbol{0},\boldsymbol{x})\right\}\right|\right|$$

can now be used to bound the integration error:

$$\left|\int_{[0,1]^d} g(\boldsymbol{x})\,d\boldsymbol{x} - \frac{1}{N}\sum_{j=1}^N g(\boldsymbol{x}_j)\right| \le D_N^* \sum_{B\subseteq\{1,2,\dots,d\}}\int_{F_B}\left|\frac{\partial^{|B|}g}{\partial \boldsymbol{x}^B}(\boldsymbol{x})\right|\,d\boldsymbol{x}.$$

It can be shown that $\sum_{B\subseteq\{1,2,\ldots,d\}} \int_{F_B} \left|\partial^{|B|} g/\partial \boldsymbol{x}^B(\boldsymbol{x})\right| \, d\boldsymbol{x} = V_d[f]$ in (5.4.11), giving (5.4.10), as we wanted. □

The bound

$$\varepsilon_N[g] = \left| \int_{[0,1]^d} g(\boldsymbol{x})\, d\boldsymbol{x} - \frac{1}{N}\sum_{j=1}^{N} g(\boldsymbol{x}_j) \right| \le D_N^* V_d[g]$$

on the integration error splits the error into a part that depends only on the points $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$ and a part that depends only on the regularity of $g$. In practice, the value of $V_d[g]$ tends to give great overestimates of the error, while for Halton and Sobol' points, the integration error seems to behave like $\mathcal{O}(D_N^*)$ as $N \to \infty$ [47, p. 26]. So it is natural to focus on finding sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ where the discrepancy decreases rapidly. Both Halton and Sobol' sequences in $\mathbb{R}^d$ have $D_N^* = \mathcal{O}(N^{-1}(\log N)^d)$ as $N \to \infty$ giving $\varepsilon_N[g] = \mathcal{O}(N^{-1}(\log N)^d)$ as $N \to \infty$. On the other hand, if the points $\boldsymbol{x}_j$ are sampled randomly and independently from a uniform distribution of $[0, 1]^d$,

$$\mathbb{E}[\varepsilon_N[g]] = \left[\int_{[0,1]^d} (g(\boldsymbol{x}) - \overline{g})^2 d\boldsymbol{x}\right]^{1/2} N^{-1/2},$$

where $\overline{g} = \int_{[0,1]^d} g(\boldsymbol{x})\, d\boldsymbol{x}$ [47, (5.10) and (5.11)].

Halton sequences [189, pp. 29-33] are generated by using the one-dimensional *van der Corput sequences*: given a base $b$ and positive integer $n$, we write $n = (h_k h_{k-1} \cdots h_1 h_0)_b = h_k b^k + h_{k-1} b^{k-1} + \cdots + h_1 b + h_0$ where the integers $0 \le h_j < b$ for all $j$. Then the $n$th van der Corput number in base $b$ is

$$\text{(5.4.14)} \quad \text{vdc}(n, b) = (0.h_0 h_1 \cdots h_{k-1} h_k)_b = h_0 b^{-1} + h_1 b^{-2} + \cdots + h_{k-1} b^{-k} + h_k b^{-k-1}.$$

Halton sequences in $[0, 1]^d$ use $d$ different bases $b_1, b_2, \ldots, b_d$ and

$$\text{(5.4.15)} \quad \text{halton}(n, [b_1, b_2, \ldots, b_d]) = (\text{vdc}(n, b_1), \text{vdc}(n, b_2), \ldots, \text{vdc}(n, b_d)).$$

Correlations between the different components of Halton sequences, especially near the beginning of the sequences, have been observed even if the bases $b_1, b_2, \ldots$ are distinct primes. For this reason, initial segments of a Halton sequence are often dropped, and after this, only every $m$th element in the Halton sequence used.

Sobol' sequences [235] of points $\boldsymbol{x}_k \in [0, 1]^d$ are generated separately through sequences for each $j$ as follows. The basic algorithm is outlined in [29]. For each $j$ we designate a distinct irreducible polynomial $p_j(z) = \sum_{\ell=0}^{r} a_{j,r-\ell} x^\ell \pmod 2$ for each $j$ where $r = \deg p_j = s_j$ with $a_{j,0} = 1$. We define $a \oplus b$ for integers $a$ and $b$ in terms of bits: $\text{bit}(a \oplus b, i) = \text{bit}(a, i) + \text{bit}(b, i) \pmod 2$. This can be extended to dyadic fractions ($u/2^q$ with $u, q \in \mathbb{Z}$) by allowing $\text{bit}(a, i)$ to have negative input $i$. Note that irreducibility implies that $p_j(0) = a_{j,r} \ne 0$, so that $a_{j,r} = 1 \pmod 2$. The $j$th component of $\boldsymbol{x}_k$ is given by

$$(\boldsymbol{x}_k)_j = (e_0 v_{j,0}) \oplus (e_1 v_{j,1}) \oplus \cdots \oplus (e_m v_{j,q}) \tag{5.4.16}$$

where $k = (e_q e_{q-1} \cdots e_1 e_0)_2$ is the binary representation of $k$. The quantities $v_{j,\ell}$ are binary fractions given by $v_{j,\ell} = m_{j,\ell}/2^{\ell+1}$; the integers $m_{j,\ell}$ are given by the recurrence

$$m_{j,\ell} = (2a_1 m_{j,\ell-1}) \oplus (2^2 a_2 m_{j,\ell-2}) \oplus \cdots \oplus (2^r a_r m_{j,\ell-r}) \oplus m_{j,\ell-r} \tag{5.4.17}$$

where $r = s_j$. Note that the dyadic fractions $v_{j,\ell}$ satisfy the recurrence

$$v_{j,\ell} = (a_1 v_{j,\ell-1}) \oplus (a_2 v_{j,\ell-2}) \oplus \cdots \oplus (a_r v_{j,\ell-r}) \oplus (v_{j,\ell-r}/2^r). \tag{5.4.18}$$

Sobol' sequences can be more efficiently computed [6] using *Gray codes*. Gray codes are a way of generating the number $0, 1, 2, \ldots, 2^m - 1$ where consecutive numbers in the sequence differ by exactly one bit. For example, for $m = 3$, the Gray code is $0 = (000)_2$, $1 = (001)_2$, $3 = (011)_2$, $2 = (010)_2$, $6 = (110)_2$, $7 = (111)_2$, $5 = (101)_2$, $4 = (100)_2$. A simple formula for the Gray code of $n$ is $G(n) = n \oplus \lfloor n/2 \rfloor$ where $\lfloor x \rfloor$ is the largest integer $\le x$. Note that if $n$ is a non-negative integer, then $\mathrm{bit}(\lfloor n/2 \rfloor, i) = \mathrm{bit}(n, i+1)$; that is, $n \mapsto \lfloor n/2 \rfloor$ shifts the bits of $n$ right by one place. By re-ordering the Sobol' points using the Gray code, we obtain a more efficient implementation:

$$x_{j,G(k+1)} = x_{j,G(k)} \oplus v_{j,c(k)}$$

where $c(k)$ is the index of the only bit where $G(k+1)$ and $G(k)$ differ.

Provided we choose the initial values $m_{j,\ell}$ for $0 \le \ell \le s_j - 1$ to be odd, then every $m_{j,\ell}$ is odd. Also we require that $m_{j,\ell} \le 2^\ell$. This ensures that there are no repetitions in the numbers $x_{k,j}$ as $k = 0, 1, 2, \ldots$.

Both Halton and Sobol' points $\boldsymbol{x}_k \in [0,1]^d$ have the property that the discrepancy $D_N^* = \mathcal{O}(N^{-1}(\log N)^d)$. Other sequences have the same asymptotic order of discrepancy. *Faure sequences* have smaller discrepancies than either Halton or Sobol' sequences, but with the same asymptotic order. There is a theorem of Roth [219] that for $d \ge 2$ and *any* infinite sequence, $D_N^*$ is bounded below by $D_N^* = \Omega(N^{-1}(\log N)^{d/2})$ as $N \to \infty$.

In spite of the good asymptotic properties of the discrepancy $D_N^*$ as $N \to \infty$ for Halton, Sobol', and Faure sequences, the value of $N$ needed to approach this asymptotic behavior grows exponentially in the dimension $d$. Specifically, we need $N > 2^d$ to start seeing better behavior using quasi-Monte Carlo methods than using random or pseudo-random sequences [181]. The effectiveness of quasi-Monte Carlo methods in high dimensions is the subject of the paper by Sloan and Wozniakowski [234]. In this paper, the authors consider families of functions $f(x_1, x_2, \ldots, x_d)$ with norm

$$\|f\|_{\gamma,d} = \left[ \sum_{U \subseteq \{1,2,\ldots,d\}} \gamma_U^{-1} \int_{[0,1]^U} \left| \frac{\partial^{|U|} f}{\partial \boldsymbol{x}_U}(\boldsymbol{x}) \right|^2 d\boldsymbol{x}_U \right]^{1/2} \quad \text{where}$$

$$\gamma_U = \prod_{j \in U} \gamma_j, \qquad \boldsymbol{x}_U = [x_j \mid j \in U], \qquad \text{with} \quad x_k = 1 \quad \text{if } k \notin U.$$

Thus $\gamma_j^{1/2}$ represents the roughness of $f$'s dependence on $x_j$. If $s_d(\boldsymbol{\gamma}) := \sum_{j=1}^{d} \gamma_j$ is bounded as $d \to \infty$, the number of function evaluations by a quasi-Monte Carlo method needed for an error of $\leq \epsilon$ is independent of $d$ and polynomial in $\epsilon^{-1}$; if $s_d(\boldsymbol{\gamma})/\ln d$ is bounded as $d \to \infty$ the number of function evaluations by a quasi-Monte Carlo method needed for an error of $\leq \epsilon$ is polynomial in $d$ and $\epsilon^{-1}$. Otherwise the bounds are exponential in $d$, no matter the quasi-Monte Carlo method used. Since these are bounds, they do not cover all possible cases in which quasi-Monte Carlo methods are successful, but the results of [234] certainly give good guidance as to when quasi-Monte Carlo methods are most successful.

**Exercises.**

(1) Try out the Buffon needle problem of Section 7.4.1.3 with the length of the needle $\ell$ equal to the spacing $s$ between the lines. Use $N = 2^k$ samples from a pseudo-random number generator for $k = 1, 2, \ldots, 20$, and plot the error against $N$. What is the empirical estimate of the error in the form error $\approx C\, N^{-\alpha}$?

(2) Repeat the previous Exercise using Halton numbers instead of using a built-in pseudo-random number generator.

(3) Repeat the Buffon needle problem of Exercise 1, but now using antithetic variates (5.4.6) in the angle at which the needle lies to reduce the variance.

(4) A generalization of antithetic variates (5.4.6) for a non-symmetric probability density function $p(x)$ in one dimension is to use the cumulative distribution function $F(x) = \int_{-\infty}^{x} p(t)\, dt$. At each step generate a random sample $U$ from a uniform distribution on [0, 1], and then compute $\frac{1}{2}(f(F^{-1}(U)) + f(F^{-1}(1 - U)))$ instead of $\frac{1}{2}(f(X) + f(-X))$ for the symmetric case. Show that $F^{-1}(U)$ and $F^{-1}(1 - U)$ both have probability density function $p(x)$. Apply this version of antithetic variables for variance reduction to estimating $\mathbb{E}[X]$ where $X$ has probability density function $p(x) = x\, e^{-x}$ for $x \geq 0$. Use $n$ antithetic pairs of samples for $n = 2^k$, $k = 1, 2, \ldots, 20$. Plot the error in the estimate for $\mathbb{E}[X]$ against $n$ using a log–log plot. Compare with estimates without using antithetic variables.

(5) Consider the problem of estimating $\int_{\mathbb{R}^d} p(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x}$ where $p(\boldsymbol{x})$ is the probability density function for a Gaussian distribution with mean $\boldsymbol{\mu}$ and variance–covariance matrix $V$: $p(\boldsymbol{x}) = (2\pi)^{-d/2} (\det V)^{-1/2} \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T V^{-1} (\boldsymbol{x} - \boldsymbol{\mu}))$. Suppose that $V \boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j$ with $\|\boldsymbol{v}_j\|_2 = 1$ gives an orthonormal basis of eigenvectors of $V$. Show that the $2d + 1$ point formula

$$\int_{\mathbb{R}^d} p(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x} \approx f(\boldsymbol{\mu}) + \frac{1}{2}\sum_{j=1}^{d} \frac{f(\boldsymbol{\mu}+s_j\boldsymbol{v}_j) - 2f(\boldsymbol{\mu}) + f(\boldsymbol{\mu}-s_j\boldsymbol{v}_j)}{s_j^2}\lambda_j$$

is exact for cubic functions $f$ provided $s_j \neq 0$ for all $j$. [**Hint:** Use the change of variables $f(\boldsymbol{x}) = g(\boldsymbol{y})$ where $\boldsymbol{x} = \boldsymbol{\mu} + [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_d]\boldsymbol{y}$, and use symmetry to show that the integrals with $g(\boldsymbol{y}) = y_k, y_k y_\ell, y_k^3, y_k^2 y_\ell$, and $y_k y_\ell y_p$ with $k \neq \ell \neq p \neq k$ are all zero.]

(6) ⚠ Following the previous Exercise, create a method for estimating $\int_{\mathbb{R}^d} p(\boldsymbol{x})\, f(\boldsymbol{x})\, d\boldsymbol{x}$ that involves $\mathcal{O}(d^2)$ function evaluations and is exact for all polynomials $f$ of degree $\leq 5$. [**Hint:** Focus on $g(\boldsymbol{y}) = y_k^4$ and $y_k^2 y_\ell^2$ for $k \neq \ell$.]

(7) In this Exercise, we create a pseudo-random function $f(\boldsymbol{x}) = \sum_{j=1}^{N} \alpha_j \cos(\boldsymbol{k}_j^T \boldsymbol{x} + \psi_j)$ with $\alpha_j$ uniformly distributed over $[0, 1/j^2]$, $k_{i,j} \in \{0, 1, 2, 3\}$ chosen independently with probabilities $\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}$, and $\psi_j$ uniformly distributed over $[0, 2\pi]$. Show that the exact integral is $(2\pi)^d \sum_{j\in\mathcal{J}} \alpha_j$ where $\mathcal{J} = \left\{ j \mid \boldsymbol{k}_j = \boldsymbol{0} \right\}$. Use quasi-Monte Carlo integration (choose between Halton, Sobol', and Faure sequences) to estimate $\int_{[0,2\pi]^d} f(\boldsymbol{x})\, d\boldsymbol{x}$ for a specific function generated this way, for $N = 100$ and $d = 2, 4, 8, 16, 32$. Plot the error against $n$, the number of samples for $n = 2^k$, $k = 1, 2, \ldots, 20$. Use a log–log plot. How does the error behave for increasing $d$?

(8) Repeat the previous Exercise using pseudo-random number generators instead of quasi-Monte Carlo generators. How does the error behave for increasing $d$?

(9) ⚠ In this Exercise, we aim to generalize the approach of antithetic variables for probability distributions that are invariant under a group of transformations of $\mathbb{R}^d$. In (5.4.6), it is assumed that the probability density function $p(\boldsymbol{x})$ is symmetric under the transformation $\boldsymbol{x} \mapsto -\boldsymbol{x}$, so that $p(-\boldsymbol{x}) = p(\boldsymbol{x})$. Instead, suppose that there is a group of transformations $G = \left\{ \gamma \colon \mathbb{R}^d \to \mathbb{R}^d \mid \gamma \in G \right\}$ under which $p(\boldsymbol{x})$ is invariant: that is, $p(\gamma(\boldsymbol{x})) = p(\boldsymbol{x})$ for each $\boldsymbol{x}$ and $\gamma \in G$. Assuming $G$ is finite, instead of computing $\frac{1}{2}(f(\boldsymbol{X}) + f(-\boldsymbol{X}))$ for each sample $\boldsymbol{X}$, we compute $|G|^{-1} \sum_{\gamma:\gamma\in G} f(\gamma(\boldsymbol{X}))$ for each sample $\boldsymbol{X}$. Further assume that each $\gamma$ is represented by an orthogonal matrix $U_\gamma$: $\gamma(\boldsymbol{x}) = U_\gamma \boldsymbol{x}$. Using the analysis techniques of Section 5.4.1.2 or otherwise, determine the reduction of the variance for smooth $f$.

## 5.5 Numerical Differentiation

Computation of derivatives is useful for many applications, such as ordinary and partial differential equations and computing gradients for optimization. Most of this section will be focused on using discrete values $(x_k,\ f(x_k))$ to estimate $f'(x)$ at some $x$. However, in optimization, we still want to compute gradients of known functions. These functions may be computed by computer code that is complex and where symbolic computation is impracticable. For these situations, there are methods

referred to as *automatic differentiation*, which can transform a given code into a new code that can compute not only the function values as the original code did but also the derivatives in an efficient way.

## 5.5.1 Discrete Derivative Approximations

In this section, we consider approximations of the form

$$f'(x) \approx \frac{1}{h} \sum_{j=1}^{m} c_j \, f(x + h\,\xi_j) \qquad \text{or}$$

$$f''(x) \approx \frac{1}{h^2} \sum_{j=1}^{m} d_j \, f(x + h\,\eta_j), \qquad \text{etc.}$$

These use discrete function values $f(x + h\,\xi_j)$, often at equally spaced points because of the context in which they are used. The derivation and analysis of these approximations often proceed via interpolation error estimates, or Taylor series with remainder.

### 5.5.1.1 One-Sided and Centered Differences

The simplest formula for differentiation, the *one-sided difference* formula, comes directly from

$$\begin{aligned} f'(x) &= \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \\ &\approx \frac{f(x+h) - f(x)}{h} \qquad \text{for } h \approx 0. \end{aligned} \tag{5.5.1}$$

We can estimate the error most simply using Taylor series with remainder (1.6.1):

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2} f''(c_h)h^2,$$

for some $c_h$ between $x$ and $x + h$. Then

$$\frac{f(x+h) - f(x)}{h} = f'(x) + \frac{1}{2} f''(c_h)h,$$

and the error is $\mathcal{O}(h)$.

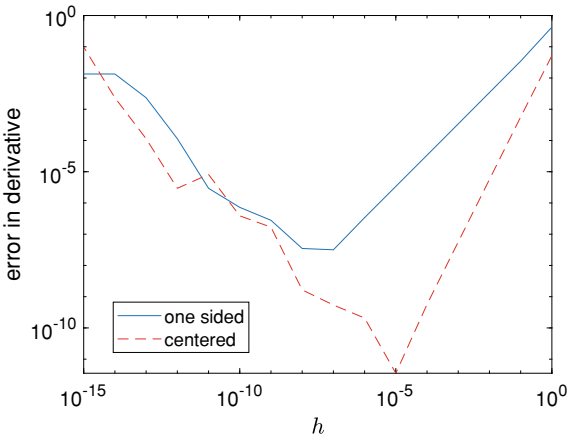



**Fig. 5.5.1** Results of one-sided and centered differences for $f(x) = e^x/(1+x)$ at $x = 1$

The centered difference rule can also be derived using Taylor series with remainder:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)\,h + \frac{1}{2}f''(x)\,h^2 + \frac{1}{3!}f'''(c_h)\,h^3, \\ f(x-h) &= f(x) - f'(x)\,h + \frac{1}{2}f''(x)\,h^2 - \frac{1}{3!}f'''(d_h)\,h^3, \quad \text{and so} \\ f(x+h) - f(x-h) &= 2f'(x)\,h + \frac{1}{3!}\left(f'''(c_h) + f'''(d_h)\right)h^3. \end{aligned}$$

That is,

$$\begin{aligned} \frac{f(x+h) - f(x-h)}{2h} &= f'(x) + \frac{1}{12}\left(f'''(c_h) + f'''(d_h)\right)h^2 \\ &= f'(x) + \frac{1}{6}f'''(\widetilde{c}_h)\,h^2 \end{aligned} \tag{5.5.2}$$

for some $\widetilde{c}_h$ between $x-h$ and $x+h$. This gives an error of $\mathcal{O}(h^2)$.

If these are used for computing, for example, gradients of a given function, we can choose $h$. Clearly, we get smaller error in our estimates for smaller $h$. So why not make $h$ as small as we can get away with? Why not make $h$ a fraction of unit roundoff? This should give derivatives that are accurate to unit roundoff. Unfortunately, this does not work. The analysis here assumes that the arithmetic is exact.

Figure 5.5.1 shows the error in computing the derivative of $f(x) = e^x/(1+x)$ at $x = 1$ using (5.5.1) and (5.5.2) for different values of $h$.

For larger values of $h$ (on the right side), we can clearly see steep reductions in the error as $h$ is decreased (going left) until a critical point where further reducing $h$ seems mainly to increase the error, but in an erratic way. The erratic behavior of the error is indicative of roundoff, which is exactly the source of the problem. Using the formal model of floating point arithmetic (1.3.1), we can roughly bound

the roundoff error (assuming that $f$ is well-implemented) by $2\mathbf{u}\,|f(x)|\,/h$ for the one-sided formula and $\mathbf{u}\,|f(x)|\,/h$ for the centered difference formula where $\mathbf{u}$ is the unit roundoff for the floating point arithmetic. The total error can then be roughly bounded by

$$\begin{aligned} &\frac{1}{2}\left|f''(x)\right||h| + \frac{2\mathbf{u}\,|f(x)|}{|h|} && \text{for the one-sided formula, and} \\ &\frac{1}{6}\left|f''(x)\right|h^2 + \frac{\mathbf{u}\,|f(x)|}{|h|} && \text{for the centered formula.} \end{aligned}$$

Minimizing these bounds gives recommended values for $h$:

$$\begin{aligned} h^* &= 2\mathbf{u}^{1/2}\left|\frac{f(x)}{f''(x)}\right|^{1/2} && \text{for the one-sided formula, and} \\ h^* &= 3^{1/3}\mathbf{u}^{1/3}\left|\frac{f(x)}{f'''(x)}\right|^{1/3} && \text{for the centered formula.} \end{aligned}$$

As a rough order of magnitude estimate, we should take $h^* \approx \mathbf{u}^{1/2}$ for the one-sided formula and $h^* \approx \mathbf{u}^{1/3}$ for the centered formula. For double precision, these are roughly $10^{-8}$ and $10^{-5}$, respectively, corresponding to the minimum values of the error in Figure 5.5.1.

If we are using one of these methods for estimating gradients $\nabla f(\boldsymbol{x})$ for $f : \mathbb{R}^n \to \mathbb{R}$, we need $n + 1$ function evaluations using the one-sided difference formula and $2n$ function evaluations using the centered difference formula.

#### 5.5.1.2 Higher Order Methods and Other Variants

By expanding by Taylor series with remainder, we can obtain higher order methods. If we start with the centered difference formula

$$\begin{aligned} \frac{f(x+h) - f(x-h)}{2h} &= f'(x) + \frac{1}{6}f'''(x)\,h^2 + \mathcal{O}(h^4), && \text{then} \\ \frac{f(x+2h) - f(x-2h)}{4h} &= f'(x) + \frac{4}{6}f'''(x)\,h^2 + \mathcal{O}(h^4). \end{aligned}$$

Combining them with

$$\begin{aligned} &\alpha\frac{f(x+h) - f(x-h)}{2h} + (1-\alpha)\frac{f(x+2h) - f(x-2h)}{4h} \\ &= f'(x) + \left[\alpha\frac{1}{6} + (1-\alpha)\frac{4}{6}\right]f'''(x)\,h^2 + \mathcal{O}(h^4), \end{aligned}$$

we get a fourth order method if we choose $\alpha$ so that $[\alpha + 4(1-\alpha)]\,/6 = 0$; that is, $\alpha = 4/3$. Then

(5.5.3)
$$f'(x) = \frac{-f(x+2h) + 8\,f(x+h) - 8\,f(x-h) + f(x-2h)}{12\,h} + \mathcal{O}(h^4).$$

More generally, we can use interpolation to estimate derivatives: if $p_n$ is the polynomial interpolant of degree $\le n$ with interpolation points $x_0,\ x_1,\ \ldots,\ x_n$, then from (4.1.6),

$$f(x) = p_n(x) + f[x, x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_n).$$

Taking derivatives,

$$\begin{aligned} f'(x) &= p_n'(x) + \frac{d}{dx}\left(f[x, x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_n)\right) \\ &= p_n'(x) + f[x, x, x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_n) \\ &\quad + f[x, x_0, x_1, \ldots, x_n] \sum_{j=0}^{n} \prod_{k:k\neq j} (x - x_k). \end{aligned}$$

If we fix the pattern of the interpolation with $x_j = a + h\,\xi_j$ we get

$$\begin{aligned} f'(a) &= p_n'(a) + f[a, a, a + h\xi_0, \ldots, a + h\xi_n]\, h^{n+1}(-1)^{n+1} \prod_{j=0}^{n} \xi_j \\ &\quad + f[a, a + h\xi_0, \ldots, a + h\xi_n]\, h^n (-1)^n \sum_{j=0}^{n} \prod_{k:k\neq j} \xi_k. \end{aligned} \tag{5.5.4}$$

If $\sum_{j=0}^{n} \prod_{k:k\neq j} \xi_k \neq 0$ then we get $\mathcal{O}(h^n)$ error in the derivative. However, sometimes we can get an improvement of the order by one if $\sum_{j=0}^{n} \prod_{k:k\neq j} \xi_k = 0$, which is the case for the centered difference method: $\xi_0 = -1$ and $\xi_1 = +1$. It is also the case of the fourth order method (5.5.3) with $\boldsymbol{\xi} = [-2,\ -1,\ +1,\ +2]^T$. It might appear that we can get zero error by making $\prod_{j=0}^{n} \xi_j = 0$ and $\sum_{j=0}^{n} \prod_{k:k\neq j} \xi_k = 0$. To do so, we would need $\xi_j = 0$ for *two* values of $j$, which means that we would be using a version of Hermite interpolation, interpolating $f'(a)$. This of course requires knowing the value of $f'(a)$, which is what we are trying to compute.

### 5.5.1.3 Higher Order Derivatives

Second order derivatives can be estimated numerically, and the best known method for doing this is the *three-point stencil*:

$$f''(x) \approx \frac{f(x+h) - 2\,f(x) + f(x-h)}{h^2}. \tag{5.5.5}$$

We can analyze this method by using Taylor series with remainder: the important point is to make as many low-order terms cancel except for the desired $f''(x)$:

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(x)h^3 + \frac{1}{24}f^{(4)}(c_{x,h})h^4 \\ f(x) &= f(x) \\ f(x-h) &= f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(x)h^3 + \frac{1}{24}f^{(4)}(c_{x,-h})h^4, \end{aligned}$$

so

$$\frac{f(x+h) - 2\,f(x) + f(x-h)}{h^2} = f''(x) + \frac{1}{24}\left[f^{(4)}(c_{x,h}) + f^{(4)}(c_{x,-h})\right]h^2$$

giving an error of $\mathcal{O}(h^2)$. We can create a fourth order method by combining this with the $2h$ three point stencil:

$$\frac{f(x+2h) - 2\,f(x) + f(x-2h)}{(2h)^2} = f''(x) + \frac{1}{24}\left[f^{(4)}(c_{x,2h}) + f^{(4)}(c_{x,-2h})\right](2h)^2.$$

The formula we can use is

$$\frac{4}{3}\frac{f(x+h) - 2\,f(x) + f(x-h)}{h^2} - \frac{1}{3}\frac{f(x+2h) - 2\,f(x) + f(x-2h)}{(2h)^2} = f''(x) + \mathcal{O}(h^4),$$

as can be verified by using Taylor series with fourth order remainder. That is,

(5.5.6)
$$f''(x) = \frac{-f(x+2h) + 16\,f(x+h) - 30\,f(x) + 16\,f(x-h) - f(x-2h)}{12\,h^2} + \mathcal{O}(h^4).$$

For differentiating-the-interpolant approaches, we can estimate the error in second derivatives in a way similar to (5.5.7):

$$\begin{aligned} f''(x) - p_n''(x) &= \frac{d^2}{dx^2}\left(f[x, x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\cdots(x - x_n)\right) \\ &= f[x, x, x, x_0, x_1, \ldots, x_n]\prod_{j=0}^{n}(x - x_j) \\ &\quad + 2\,f[x, x, x_0, x_1, \ldots, x_n]\sum_{k=0}^{n}\prod_{j:j\neq k}(x - x_j) \\ &\quad + f[x, x_0, x_1, \ldots, x_n]\sum_{k\neq\ell=0}^{n}\prod_{j:j\neq k,\ell}(x - x_j). \end{aligned}$$

Putting $x_j = a + h\,\xi_j$ for our interpolation pattern,

$$\begin{aligned} f''(a) - p_n''(a) = {} & f[a,a,a,x_0,x_1,\ldots,x_n]h^{n+1}(-1)^{n+1}\prod_{j=0}^{n}\xi_j \\ & + 2\,f[a,a,x_0,x_1,\ldots,x_n]h^{n}(-1)^{n}\sum_{k=0}^{n}\prod_{j:j\neq k}\xi_j \\ & + f[a,x_0,x_1,\ldots,x_n]h^{n-1}(-1)^{n-1}\sum_{k\neq\ell=0}^{n}\prod_{j:j\neq k,\ell}\xi_j . \end{aligned}$$

For the standard three point stencil we have $n = 2$, $\xi_0 = -1$, $\xi_1 = 0$, and $\xi_2 = +1$. In this case $\sum_{k\neq\ell=0}^{n}\prod_{j:j\neq k,\ell}\xi_j = \xi_0 + \xi_1 + \xi_2 = 0$ but

$$\sum_{k=0}^{n}\prod_{j:j\neq k}\xi_j = \xi_0\xi_1 + \xi_0\xi_2 + \xi_1\xi_2 = \xi_0\xi_2 = -1$$

giving $\mathcal{O}(h^n) = \mathcal{O}(h^2)$.

In applications to optimization, we might want to compute second derivatives to estimate the Hessian matrix $\operatorname{Hess} f(\boldsymbol{x}) = \left[\partial^2 f/\partial x_k \partial x_\ell(\boldsymbol{x})\right]_{k,\ell=1}^{n}$ for a function $f\colon \mathbb{R}^n \to \mathbb{R}$. The formula (5.5.5) can be used to compute the diagonal entries of the Hessian matrix $\partial^2 f/\partial x_k^2(\boldsymbol{x})$. The mixed derivatives $\partial^2 f/\partial x_k \partial x_\ell(\boldsymbol{x})$ require a different approach:

$$\begin{aligned} \frac{\partial^2 f}{\partial x_k \partial x_\ell}(\boldsymbol{x}) \approx \frac{1}{4\,h^2}\,[\,& f(\boldsymbol{x} + h\boldsymbol{e}_k + h\boldsymbol{e}_\ell) - f(\boldsymbol{x} - h\boldsymbol{e}_k + h\boldsymbol{e}_\ell) \\ & - f(\boldsymbol{x} + h\boldsymbol{e}_k - h\boldsymbol{e}_\ell) + f(\boldsymbol{x} - h\boldsymbol{e}_k - h\boldsymbol{e}_\ell)\,]\,. \end{aligned} \tag{5.5.7}$$

Using multivariate Taylor series with fourth order remainder, we get cancellation of all terms inside the square brackets except the $\partial^2 f/\partial x_k \partial x_\ell(\boldsymbol{x})$ term and the $\mathcal{O}(h^4)$ terms. This gives an error in the formula (5.5.7) that is $\mathcal{O}(h^2)$. Thus the entire $n \times n$ Hessian matrix can be estimated to within $\mathcal{O}(h^2)$ using $2n^2 + 1$ function evaluations.

### 5.5.2 *Automatic Differentiation*

If a function is given, even implicitly, by a formula, then we can compute its derivative symbolically. The usual ways of doing this are:

- "by hand" where someone who knows calculus performs the symbolic calculations by pen/pencil and paper; or

- the formula is first determined (whether "by hand" or by some automated process), and then the derivatives are computed by a symbolic mathematics package, such as Axiom, Maple, Mathematica, Maxima, or SageMath.

Neither of these approaches is very satisfactory, especially when the function concerned is not given directly as a formula, but rather implicit in some (perhaps large) piece of software. *Automatic differentiation*, also known as *algorithmic differentiation* or *computational differentiation*, takes a more computational approach based on the basic rules of calculus with special attention paid to the chain rule [111, 186].

#### 5.5.2.1 Forward Mode

Forward mode is the simplest approach for automatic differentiation, both conceptually and in practice. This idea is sometimes implemented as *dual numbers* in programming languages that allow overloaded arithmetic operations and functions. A dual number is a pair $x = (x.v, x.d)$ where $x.v$ represents the value of the number, and $x.d$ its derivative with respect to some single parameter, say $dx/ds$. Ordinary numbers are treated as constants, and so are represented as $(v, 0)$ where $v$ is the number.

Operations on dual numbers $x$ and $y$ can be described as

$$\begin{aligned}
x + y &= (x.v + y.v,\ x.d + y.d),\\
x - y &= (x.v - y.v,\ x.d - y.d),\\
x \cdot y &= (x.v \cdot y.v,\ x.v \cdot y.d + x.d \cdot y.v),\\
x/y &= (x.v/y.v,\ (x.d \cdot y.v - x.v \cdot y.d)/(y.v)^2),\\
f(x) &= (f(x.v),\ f'(x.v) \cdot x.d).
\end{aligned}$$

This can be extended to handle higher order derivatives, such as triple numbers $x = (x.v,\ x.d,\ x.c)$ where $x.d = dx/ds$ and $x.c = d^2x/ds^2$. Then for triple numbers, for example, the arithmetic rules include

$$\begin{aligned}
x \cdot y &= (x.v \cdot y.v,\ x.v \cdot y.d + x.d \cdot y.v,\ x.v \cdot y.c + 2\,x.d \cdot y.d + x.c \cdot y.v),\\
f(x) &= (f(x.v),\ f'(x.v) \cdot x.d,\ f'(x.v)\,x.c + f''(x.v)\,(x.d)^2).
\end{aligned}$$

The derivatives computed would be exact if the underlying arithmetic were exact. Thus the only errors in the computed derivatives are due to roundoff error. This does not guarantee accurate results, but they rarely fail.

Forward mode automatic differentiation is suitable where there is one, or a small number, of independent variables with respect to which we wish to computed derivatives. If we wish to compute gradients for many inputs, we need a different method.

#### 5.5.2.2 Reverse Mode

The *reverse mode* of automatic differentiation is best suited to compute gradients of a single output function with respect to many inputs. The basic idea has been rediscovered multiple times that we know of, but the modern approach can be traced back at least to Seppo Linnainmaa in his PhD thesis that was later published [163]. For this, we need to conceptually flatten the execution of a piece of code so that it is written as a "straight-line code" with branches and loops removed. For example, the loop

```
for i = 1, 2, ..., 4
  x ← f(x)
end
```

should be written as it is executed:

$$
\begin{aligned}
x_1 &\leftarrow f(x_0)\\
x_2 &\leftarrow f(x_1)\\
x_3 &\leftarrow f(x_2)\\
x_4 &\leftarrow f(x_3)
\end{aligned}
$$

The index $j$ in $x_j$ indicates a potentially new value for the variable "$x$" for each pass through the body of the loop.

In reverse mode automatic differentiation, this execution path and the values of variables along this path must be saved, at least at strategically important points of the execution of the original code. This can be represented in a computational graph of the execution of the code. Note that in the computational graph, each variable must only be assigned a value once. If a value of a variable is over-written, then we create a new variable for the computational graph, as shown in the example of the loop above.

The code

$$
\begin{aligned}
u &\leftarrow r \cdot s\\
v &\leftarrow r^s\\
x &\leftarrow \varphi(u, v)\\
y &\leftarrow x \cdot r
\end{aligned}
$$

can be represented by the computational graph in Figure 5.5.2.

We compute the partial derivatives $\partial y/\partial z$ for $z$ each of the variables in the computational graph as we go back through the computational graph. If we consider the last node of the computational graph *and no prior operations*, then $\partial y/\partial y = 1$ and $\partial y/\partial z = 0$ for *all* other variables $z$. Consider the last operation: $y \leftarrow x \cdot r$. Considering *this operation alone*, we have $\partial y/\partial x = r$ and $\partial y/\partial r = x$. Now include the second last operation: $x \leftarrow \varphi(u, v)$. Taking this into account, we find that $\partial y/\partial u = (\partial y/\partial x)(\partial x/\partial u) = (\partial y/\partial x)(\partial \varphi/\partial u(u, v))$ while $\partial y/\partial v = (\partial y/\partial x)(\partial x/\partial v) = (\partial y/\partial x)(\partial \varphi/\partial v(u, v))$. We have $\partial y/\partial x$ from before this operation (it is equal to $r$) so we can compute $\partial y/\partial u$ and $\partial y/\partial v$.

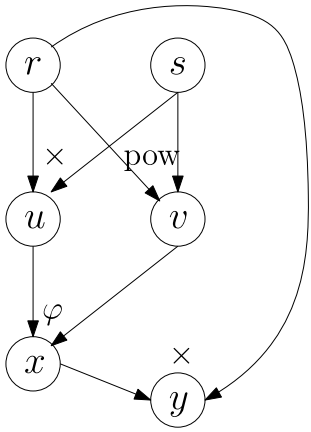



**Fig. 5.5.2** Computational graph

The previous operations are $u \leftarrow r \cdot s$ and $v \leftarrow r^s$. We already have a value for $\partial y/\partial r$ that is non-zero that does not take these operations into account. This should not be over-written. Instead, we *add* $(\partial y/\partial u)(\partial u/\partial r)$ and $(\partial y/\partial v)(\partial v/\partial r)$ to $\partial y/\partial r$. Similarly, $\partial y/\partial s = (\partial y/\partial u)(\partial u/\partial s) + (\partial y/\partial v)(\partial v/\partial s)$.

To see how the general rule works, consider

```
// position t
w ← ψ(r, s, u, v)
// position t + 1
```

Suppose we have values for $\partial y/\partial z$ for every variable at position $t+1$ in the code. At position $t$ we update these partial derivatives via

$$\frac{\partial y}{\partial z} \leftarrow \frac{\partial y}{\partial z} + \left(\frac{\partial y}{\partial w}\right)\left(\frac{\partial w}{\partial z}\right) \tag{5.5.8}$$

for each variable $z$ on the right side of the assignment. To see this more formally, suppose that starting at position $t+1$ we have $y = f_{t+1}(r, s, u, v, w)$. Then at position $t$ we have $y = f_t(r, s, u, v)$, since $w$ is only assigned to once, and so does not have a value at or before position $t$. Then

$$y = f_t(r, s, u, v) = f_{t+1}(r, s, u, v, \psi(r, s, u, v)) \qquad \text{and so}$$
$$\frac{\partial f_t}{\partial r} = \frac{\partial f_{t+1}}{\partial r} + \left(\frac{\partial f_{t+1}}{\partial w}\right)\left(\frac{\partial \psi}{\partial r}\right) = \frac{\partial f_{t+1}}{\partial r} + \left(\frac{\partial f_{t+1}}{\partial w}\right)\left(\frac{\partial w}{\partial r}\right), \quad \text{etc.}$$

This justifies the update rule

$$\frac{\partial y}{\partial r} \leftarrow \frac{\partial y}{\partial r} + \left(\frac{\partial y}{\partial w}\right)\left(\frac{\partial w}{\partial r}\right)$$

and so on, for the other variables $s$, $u$, and $v$.

For one output $y$, the number of operations to compute all the derivatives $\partial y/\partial z$ for all input variables $z$ is a modest multiple of the number of operations needed to compute $y$ from the inputs once: $\text{oper}(\nabla y) = \mathcal{O}(\text{oper}(y))$ where $\text{oper}(y)$ is the number of operations needed to compute $y$ from the inputs. Theoretically, this multiple is no more than five, provided only standard functions and arithmetic operations are allowed. In practice, the time needed for straight-line code is perhaps 20 because of the additional overhead in setting up, tracking, and recording the variables as we go through the computational graph. The biggest difficulty is that the amount of memory needed to store intermediate values of computational variables can become large.

As with the forward mode of automatic differentiation, the only errors are due to roundoff errors.

#### 5.5.2.3 Use of Automatic Differentiation

Automatic differentiation is such a wonderful technique, there is tendency to apply it indiscriminately. Some recent work, such as [131], can seem to promote this point of view. However, automatic differentiation is not infallible. To illustrate this, consider using the bisection method to solve $f(x, p) = 0$ for $x$: the solution $x$ is implicitly a function of $p$: $x = x(p)$. Provided for $p \approx p_0$ we have $f(a, p) < 0$ and $f(b, p) > 0$ for given fixed numbers $a < b$, bisection will give the solution $x(p)$ for $p \approx p_0$. However, in the bisection algorithm (Algorithm 40), we first look at $c = (a + b)/2$ and evaluate $f(c, p)$ and use the sign of this function value to determine how to update the endpoints $a$ and $b$. Since $a$ and $b$ are constant, $\partial a/\partial p = \partial a/\partial p = 0$, and so $\partial c/\partial p = 0$. Continuing through the bisection algorithm we find that the solution returned has $\partial x^*/\partial p = 0$. Which is wrong.

From the Implicit Function Theorem we have

$$0 = \frac{\partial f}{\partial x}(x, p)\frac{\partial x}{\partial p} + \frac{\partial f}{\partial p}(x, p), \qquad \text{so}$$
$$\frac{\partial x}{\partial p} = -\left(\frac{\partial f}{\partial p}(x, p)\right) / \left(\frac{\partial f}{\partial x}(x, p)\right).$$

Once the solution $x(p)$ is found, we can find the derivatives $\partial f/\partial p$ and $\partial f/\partial x$ using automatic differentiation. We can then compute $\partial x/\partial p$ using the above formula, regardless of how $x(p)$ is computed. In a multivariate setting, the computation of derivatives of the solution $\boldsymbol{x}(\boldsymbol{p})$ of equations $\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p}) = \boldsymbol{0}$ with respect to a parameter $\boldsymbol{p}$ will involve solving a linear system of equations: $\nabla_{\boldsymbol{p}}\boldsymbol{x}(\boldsymbol{p}) = -\nabla_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p})^{-1}\nabla_{\boldsymbol{p}}\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{p})$.

Automatic differentiation is also heavily used in machine learning and neural networks. The main neural network training algorithm *backpropagation* is essentially an application of the main ideas of automatic differentiation [18] combined with a version of gradient descent.

If gradients $\nabla f(\boldsymbol{x})$ can be computed in $\mathcal{O}(\text{oper}(f(\boldsymbol{x})))$ operations, what about second derivatives? Can we compute Hess $f(\boldsymbol{x})$ in $\mathcal{O}(\text{oper}(f(\boldsymbol{x})))$ operations? The answer is no. Take, for example, the function $f(\boldsymbol{x}) = (\boldsymbol{x}^T\boldsymbol{x})^2$. The computation of $f(\boldsymbol{x})$ only requires $\text{oper}(f(\boldsymbol{x})) = 2n + 1$ arithmetic operations. Then

$$\begin{aligned}\nabla f(\boldsymbol{x}) &= 4(\boldsymbol{x}^T\boldsymbol{x})\,\boldsymbol{x},\\ \text{Hess } f(\boldsymbol{x}) &= 4(\boldsymbol{x}^T\boldsymbol{x})\,I + 8\boldsymbol{x}\,\boldsymbol{x}^T.\end{aligned}$$

For general $\boldsymbol{x}$, Hess $f(\boldsymbol{x})$ has $n^2$ non-zero entries ($\frac{1}{2}n(n+1)$ independent entries), so we cannot expect to "compute" Hess $f(\boldsymbol{x})$ in $\mathcal{O}(n)$ operations.

However, we can compute

$$\begin{aligned}\text{Hess } f(\boldsymbol{x})\,\boldsymbol{d} &= \left[4(\boldsymbol{x}^T\boldsymbol{x})\,I + 8\boldsymbol{x}\,\boldsymbol{x}^T\right]\boldsymbol{d}\\ &= 4(\boldsymbol{x}^T\boldsymbol{x})\,\boldsymbol{d} + 8\boldsymbol{x}(\boldsymbol{x}^T\boldsymbol{d})\end{aligned}$$

in just $7n + 2 = \mathcal{O}(n)$ arithmetic operations. In general, we can compute Hess $f(\boldsymbol{x})\,\boldsymbol{d}$ in $\mathcal{O}(\text{oper}(f(\boldsymbol{x})))$. We can do this by applying the forward mode to compute

$$\frac{d}{ds}\,\nabla f(\boldsymbol{x} + s\boldsymbol{d})|_{s=0} = \text{Hess } f(\boldsymbol{x})\,\boldsymbol{d} \tag{5.5.9}$$

where we use the reverse mode for computing $\nabla f(\boldsymbol{z})$.

**Exercises.**

(1) If $f : \mathbb{R}^d \to \mathbb{R}$ is smooth, how many function evaluations are needed to estimate $\nabla f(\boldsymbol{x})$ using the one-sided difference formula (5.5.1)? How many function evaluations are needed if the centered difference formula (5.5.2) is used?

(2) Use one-sided (5.5.1) and centered differences (5.5.2) to estimate $f'(\pi/4)$ for $f(x) = (\cos x)/\sqrt{1+x}$ using spacing $h$. Plot the errors against $h$ for $h = 4^{-k}$, $k = 1, 2, \ldots, 10$.

(3) Repeat the previous Exercise using the 4th order symmetric difference method which uses the function values $f(x \pm h)$ and $f(x \pm 2h)$.

(4) Develop a formula to compute $\partial^2 f/\partial x\,\partial y(x, y)$ using the function values $f(x, y)$ and $f(x \pm h,\ y \pm h)$ with all possible choices of signs.

(5) Extend the previous Exercise into a method for computing the Hessian matrix Hess $f(\boldsymbol{x}) = [\partial^2 f/\partial x_k \partial x_\ell(\boldsymbol{x})]_{k,\ell=1}^d$ of 2nd order derivatives. How many function evaluations are needed?

(6) In many machine learning systems, there are weight *matrices* to be optimized. Suppose that we seek to minimize $h(W, \boldsymbol{b}) := g(W\boldsymbol{u} + \boldsymbol{b})$ given $\boldsymbol{u}$ over all possible values of $W$ and $\boldsymbol{b}$. We define $\nabla_W h(W, \boldsymbol{b})$ to be the matrix of partial derivatives $\partial h/\partial w_{k\ell}(W, \boldsymbol{b})$. Compute $\nabla_W h(W, \boldsymbol{b})$ and $\nabla_{\boldsymbol{b}} h(W, \boldsymbol{b})$ in terms of $\nabla g(W\boldsymbol{u} + \boldsymbol{b})$ and $\boldsymbol{u}$. [**Hint:** First do this in terms of the entries of $W$, then combine these expressions into a concise formula using matrix–vector operations.]

(7) Given a collection of data points $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, N$, we wish to compute the gradient of $\varphi(\boldsymbol{w}) := (2N)^{-1} \sum_{i=1}^{N} (y_i - g(\boldsymbol{x}_i; \boldsymbol{w}))^2$. Let $m$ be the dimension of $\boldsymbol{w}$. Given that there is efficient code for computing $\nabla_{\boldsymbol{w}} g(\boldsymbol{x}; \boldsymbol{w})$, give an efficient method for computing $\nabla\varphi(\boldsymbol{w})$. Given that there is efficient code for computing $\mathrm{Hess}_{\boldsymbol{w}} g(\boldsymbol{x}; \boldsymbol{w})\, \boldsymbol{d}$ given $\boldsymbol{x}$, $\boldsymbol{w}$, and $\boldsymbol{d} \in \mathbb{R}^m$ using only $\mathcal{O}(\mathrm{oper}(g) + m)$ operations, create a method to compute $\mathrm{Hess}\, \varphi(\boldsymbol{w})\, \boldsymbol{d}$ given $\boldsymbol{w}$ and $\boldsymbol{d}$ that uses only $\mathcal{O}(N\, \mathrm{oper}(g) + N\, m)$ operations.

(8) Backpropagation is a well-known technique for neural networks for computing gradients for these functions. Suppose that the cost for a given output $\boldsymbol{u}$ of a particular layer of the network is $\varphi(\boldsymbol{u})$. Now suppose that $\boldsymbol{u} = \mathrm{map}(\sigma,\ W\boldsymbol{v} + \boldsymbol{b})$ where $\boldsymbol{y} = \mathrm{map}(f, \boldsymbol{x})$ is given by the formula $y_k = f(x_k)$. Starting with $\nabla\varphi(\boldsymbol{u})$ for $\boldsymbol{u} = \mathrm{map}(\sigma,\ W\boldsymbol{v} + \boldsymbol{b})$, give efficient formulas for $\nabla_{\boldsymbol{v}}\varphi(\mathrm{map}(\sigma,\ W\boldsymbol{v} + \boldsymbol{b}))$ as well as $\nabla_W \varphi(\mathrm{map}(\sigma,\ W\boldsymbol{v} + \boldsymbol{b}))$ (see Exercise 6 for matrix derivatives) and $\nabla_{\boldsymbol{b}}\varphi(\mathrm{map}(\sigma,\ W\boldsymbol{v} + \boldsymbol{b}))$ using $\sigma'$.

(9) Use Exercises 7 and 8 to implement a backpropagation method for computing the gradient of $(y - \widehat{y})^2$ with respect to $\boldsymbol{z}$, $c$, and each $W^{(j)}$ and $\boldsymbol{b}^{(j)}$, $j = 1, 2, \ldots, m$, where

$$y = \boldsymbol{z}^T \mathrm{map}(\sigma, W^{(m)} \mathrm{map}(\sigma, W^{(m-1)} \mathrm{map}(\cdots W^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)} \cdots) + \boldsymbol{b}^{(m-1)}) + \boldsymbol{b}^{(m)}) + c.$$

(10) Given methods for computing gradients of $\varphi(\boldsymbol{u}, \boldsymbol{v})$ and the Jacobian matrices of $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{v})$, given that $\nabla_{\boldsymbol{u}} \boldsymbol{F}(\boldsymbol{u}, \boldsymbol{v})$ is invertible, give a method for computing $\nabla\psi(\boldsymbol{v})$ where $\psi(\boldsymbol{v}) = \varphi(\boldsymbol{u}, \boldsymbol{v})$ and $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{0}$. Assume that for each $\boldsymbol{v}$ there is exactly one $\boldsymbol{u}$ satisfying $\boldsymbol{F}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{0}$. Explain why this avoids trying to "differentiate the solver" where a nonlinear equation solver is used for computing $\psi(\boldsymbol{v})$. Show that trying to "differentiate the solver" does not work when applied to the bisection method.

# Chapter 6
# Differential Equations


Differential equations provide a language for representing many processes in nature, technology, and society. Ordinary differential equations have one independent variable on which all others depend. Usually this independent variable is time, although it may be position along a rod or string. Typically in these situations, the starting position or state is known at a particular time, and we wish to forecast how that will change with time. These are *initial value problems*. In other cases, partial values are known at the start and at the end, and the differential equation describes how things change between the start and end times. These are known as *boundary value problems*.

Partial differential equations have several independent variables, usually representing spatial co-ordinates, and sometimes including time as an additional independent variable. Where a partial differential equation has spatial as well as temporal independent variables, often the problem is discretized with respect to the spatial variables, leaving an ordinary differential equation remaining for the spatially discretized variables.

## 6.1 Ordinary Differential Equations — Initial Value Problems

The form of problem we consider here is, given $\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ and $\boldsymbol{x}_0 \in \mathbb{R}^n$, to find the function $\boldsymbol{x}(\cdot)$ where

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0. \tag{6.1.1}$$

This is the general form of an initial value problem (IVP). By "finding the function $\boldsymbol{x}(\cdot)$" computationally we actually mean "finding $\boldsymbol{x}_k \approx \boldsymbol{x}(t_k)$, $k = 0, 1, 2, 3, \ldots$ for



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_6

$t_k$'s sufficiently close together". Interpolation can then be used to give good approximations for $\boldsymbol{x}(t)$ where $t$ is between the $t_k$'s.

### 6.1.1 Basic Theory

We start with an equivalent expression of the initial value problem (6.1.1):

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{x}(s))\, ds \qquad \text{for all } t. \tag{6.1.2}$$

Peano proved the existence and uniqueness of solutions to the initial value problem using a fixed point iteration [200] named in his honor:

$$\boldsymbol{x}_{k+1}(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{x}_k(s))\, ds \qquad \text{for all } t \text{ for } k = 0, 1, 2, \ldots, \tag{6.1.3}$$

with $\boldsymbol{x}_0(t) = \boldsymbol{x}_0$ for all $t$. To show that the iteration (6.1.3) is well defined and converges, we need to make some assumptions about the right-hand side function $\boldsymbol{f}$.

Most specifically we assume that $\boldsymbol{f}(t, \boldsymbol{x})$ is continuous in $(t, \boldsymbol{x})$ and Lipschitz continuous in $\boldsymbol{x}$: there must be a constant $L$ where

$$\|\boldsymbol{f}(t, \boldsymbol{u}) - \boldsymbol{f}(t, \boldsymbol{v})\| \le L\, \|\boldsymbol{u} - \boldsymbol{v}\| \qquad \text{for all } t,\ \boldsymbol{u},\ \text{and } \boldsymbol{v}. \tag{6.1.4}$$

Caratheodory extended Peano's existence theorem to allow for $\boldsymbol{f}(t, \boldsymbol{x})$ continuous in $\boldsymbol{x}$ and measurable in $t$ with a bound $\|\boldsymbol{f}(t, \boldsymbol{x})\| \le m(t)\, \varphi(\|\boldsymbol{x}\|)$ with $m(t) \ge 0$ integrable in $t$ over $[t_0, T]$, $\varphi$ continuous, and $\int_1^\infty dr/\varphi(r) = \infty$. Uniqueness holds if the Lipschitz continuity condition (6.1.4) holds with an integrable function $L(t)$:

$$\|\boldsymbol{f}(t, \boldsymbol{u}) - \boldsymbol{f}(t, \boldsymbol{v})\| \le L(t)\, \|\boldsymbol{u} - \boldsymbol{v}\| \qquad \text{for all } t,\ \boldsymbol{u},\ \boldsymbol{v}. \tag{6.1.5}$$

We will focus on the case where $\boldsymbol{f}(t, \boldsymbol{x})$ is continuous in $t$ and Lipschitz in $\boldsymbol{x}$ (6.1.4) since numerical estimation of integrals of general measurable functions is essentially impossible.

**Theorem 6.1** *Suppose $\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ is continuous and (6.1.4) holds. Then the initial value problem (6.1.1) has a unique solution $\boldsymbol{x}(\cdot)$.*

***Proof*** We use the Peano iteration (6.1.3) to show the solution to the integral form (6.1.2) of (6.1.1) has a unique solution. To do that we show that the iteration (6.1.3) is a contraction mapping (Theorem 3.3) on the space of continuous functions $[t_0,\ t_0 + \delta] \to \mathbb{R}^n$ for $\delta = 1/(2\,L)$. This establishes the existence and uniqueness of the solution $\boldsymbol{x} : [t_0,\ t_0 + \delta] \to \mathbb{R}^n$. To show existence and uniqueness beyond this, let $t_1 = t_0 + \delta$ and $\boldsymbol{x}_1 = \boldsymbol{x}(t_0 + \delta)$. Then applying the argument to

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}), \qquad \boldsymbol{x}(t_1) = \boldsymbol{x}_1$$

we obtain a unique solution $\boldsymbol{x}^{(1)}\colon [t_1,\ t_1 + \delta] \to \mathbb{R}^n$, and we extend our solution $\boldsymbol{x}(\cdot)$ to $[t_1, t_1 + \delta]$ by $\boldsymbol{x}(t) = \boldsymbol{x}^{(1)}(t)$ for $t_1 \le t \le t_1 + \delta$. We can continue in this way to obtain a unique solution $\boldsymbol{x}\colon [t_0,\ T] \to \mathbb{R}^n$ for any $T > t_0$. By reversing the time direction, we can show existence and uniqueness on any time interval $[\widetilde{T},\ t_0]$ for any $\widetilde{T} < t_0$.

To show the (6.1.3) is a contraction mapping using the norm

$$\|\boldsymbol{u}\|_\infty = \max_{t\in[t_0,t_0+\delta]} \|\boldsymbol{u}(t)\|\,,$$

we note that

$$\begin{aligned}
&\max_{t\in[t_0,t_0+\delta]} \left\| \left(\boldsymbol{x}_0 + \int_{t_0}^t \boldsymbol{f}(s,\boldsymbol{u}(s))\,ds\right) - \left(\boldsymbol{x}_0 + \int_{t_0}^t \boldsymbol{f}(s,\boldsymbol{v}(s))\,ds\right)\right\| \\
&\quad = \max_{t\in[t_0,t_0+\delta]} \left\| \int_{t_0}^t \boldsymbol{f}(s,\boldsymbol{u}(s))\,ds - \int_{t_0}^t \boldsymbol{f}(s,\boldsymbol{v}(s))\,ds \right\| \\
&\quad = \max_{t\in[t_0,t_0+\delta]} \left\| \int_{t_0}^t \left[\boldsymbol{f}(s,\boldsymbol{u}(s)) - \boldsymbol{f}(s,\boldsymbol{v}(s))\right] ds \right\| \\
&\quad \le \max_{t\in[t_0,t_0+\delta]} \int_{t_0}^t \|\boldsymbol{f}(s,\boldsymbol{u}(s)) - \boldsymbol{f}(s,\boldsymbol{v}(s))\|\ ds \\
&\quad \le \max_{t\in[t_0,t_0+\delta]} \int_{t_0}^t L\ \|\boldsymbol{u}(s) - \boldsymbol{v}(s)\|\ ds \quad (\text{as } \boldsymbol{f} \text{ Lipschitz}) \\
&\quad \le \max_{t\in[t_0,t_0+\delta]} \int_{t_0}^t L\ \|\boldsymbol{u} - \boldsymbol{v}\|_\infty\ ds \\
&\quad \le \max_{t\in[t_0,t_0+\delta]} L\ \|\boldsymbol{u} - \boldsymbol{v}\|_\infty\,(t - t_0) = L\delta\ \|\boldsymbol{u} - \boldsymbol{v}\|_\infty \\
&\quad \le \frac{1}{2}\,\|\boldsymbol{u} - \boldsymbol{v}\|_\infty\,.
\end{aligned}$$

The iteration (6.1.3) is therefore a contraction mapping and so has a unique fixed point. The remainder of the argument follows as described above, giving existence and uniqueness of solutions on any interval $[t_0, t_1]$ with $t_1 > t_0$. For $t < t_0$ we can simply the apply the above arguments to the reversed time differential equation $d\widetilde{\boldsymbol{x}}/d\tau = -\boldsymbol{f}(\widetilde{\boldsymbol{x}}, -\tau)$ with $\widetilde{\boldsymbol{x}}(-t_0) = \boldsymbol{x}_0$ for $\widetilde{\boldsymbol{x}}(\tau) = \boldsymbol{x}(-\tau)$. □

If $\boldsymbol{f}$ is differentiable, then

$$\begin{aligned}
\boldsymbol{f}(t,\boldsymbol{y}) - \boldsymbol{f}(t,\boldsymbol{x}) &= \int_0^1 \nabla\boldsymbol{f}(t, \boldsymbol{x} + s(\boldsymbol{y} - \boldsymbol{x}))(\boldsymbol{y} - \boldsymbol{x})\,ds, \qquad \text{so} \\
\|\boldsymbol{f}(t,\boldsymbol{y}) - \boldsymbol{f}(t,\boldsymbol{x})\| &\le \int_0^1 \|\nabla\boldsymbol{f}(t, \boldsymbol{x} + s(\boldsymbol{y} - \boldsymbol{x}))\|\ \|\boldsymbol{y} - \boldsymbol{x}\|\ ds.
\end{aligned}$$

If $\|\nabla \boldsymbol{f}(t,z)\| \le B$ for all $z$ then $\boldsymbol{f}$ is Lipschitz as

$$\|\boldsymbol{f}(t,\boldsymbol{y}) - \boldsymbol{f}(t,\boldsymbol{x})\| \le \int_0^1 B \ \|\boldsymbol{y}-\boldsymbol{x}\| \ ds = B \ \|\boldsymbol{y}-\boldsymbol{x}\| .$$

On the other hand, provided $\nabla \boldsymbol{f}(t,z)$ is continuous in $z$, then for any $\boldsymbol{x}$ there is a $\boldsymbol{d} \neq \boldsymbol{0}$ where $\|\nabla \boldsymbol{f}(t,\boldsymbol{x})\boldsymbol{d}\| = \|\nabla \boldsymbol{f}(t,\boldsymbol{x})\| \, \|\boldsymbol{d}\|$. So if $\boldsymbol{f}$ is Lipschitz continuous with constant $L$, then

$$\begin{aligned} L \, \|\boldsymbol{d}\| = \lim_{s \downarrow 0} \frac{L \, s \ \|\boldsymbol{d}\|}{s} &\ge \lim_{s \downarrow 0} \frac{\|\boldsymbol{f}(t,\boldsymbol{x}+s\boldsymbol{d}) - \boldsymbol{f}(t,\boldsymbol{x})\|}{s} \\ &= \|\nabla \boldsymbol{f}(t,\boldsymbol{x}) \, \boldsymbol{d}\| = \|\nabla \boldsymbol{f}(t,\boldsymbol{x})\| \, \|\boldsymbol{d}\| \end{aligned}$$

so $L$ has to be at least $\|\nabla \boldsymbol{f}(t,\boldsymbol{x})\|$. So $\boldsymbol{f}$ is Lipschitz and differentiable if and only if $\|\nabla \boldsymbol{f}(t,z)\|$ is bounded.

**Example 6.2** The condition of Lipschitz continuity is important both for existence and uniqueness. Regarding existence, consider the initial value problem

$$\frac{dy}{dt} = 1 + y^2, \qquad y(0) = 0.$$

Using separation of variables we get

$$\begin{aligned} \tan^{-1} y = \int \frac{dy}{1+y^2} = \int dt = t + C \qquad &\text{so} \\ y(t) = \tan(t + C). & \end{aligned}$$

The initial condition $y(0) = 0$ means that we should take $C = 0$. That is, $y(t) = \tan t$. This is a reasonable solution for $-\pi/2 < t < +\pi/2$ but "blows up" as $t \to +\pi/2$ for example. This is an example of *local* existence: for $t \approx t_0$ the solution $\boldsymbol{x}(t)$ exists, but the solution cannot be extended to *all* $t$.

Note that the function $y \mapsto 1 + y^2$ is *not* Lipschitz as $(d/dy)(1+y^2) = 2y$ which is not bounded as $y \to \pm\infty$.

**Example 6.3** Regarding uniqueness, consider the initial value problem

$$\frac{dy}{dt} = |y|^{1/2}, \qquad y(0) = 0.$$

For $t \ge 0$ we have to have $y(t) \ge 0$ because the right-hand side is $\ge 0$. Using separation of variables we have for $t > 0$

$$2y^{1/2} = \int y^{-1/2}\,dy = \int dt = t + C \qquad \text{so}$$
$$y(t) = \frac{1}{4}(t + C)^2.$$

The initial value $y(0) = 0$ implies $C = 0$, giving $y(t) = t^2/4$. The problem is that this is not the only solution. Another solution is $y(t) = 0$ for all $t$. In fact, for any $t^* \geq 0$ there is the solution

$$y(t) = \begin{cases} 0, & \text{if } t \leq t^*, \\ (t - t^*)^2/4, & \text{if } t \geq t^*. \end{cases}$$

We can see the right-hand side function is not Lipschitz as

$$\frac{d}{dy}(y^{1/2}) = \frac{1}{2}y^{-1/2},$$

which is unbounded as $y \downarrow 0$.

**Remark 6.4** There are equations for which solutions exist and are unique that are not Lipschitz. If $\boldsymbol{f}(t, z)$ is continuously differentiable then there is *local* existence: consider the modified equation

$$\frac{d\boldsymbol{x}_R}{dt} = \begin{cases} \boldsymbol{f}(t, \boldsymbol{x}_R), & \text{if } \|\boldsymbol{x}_R\|_2 \leq R, \\ \boldsymbol{f}(t, R\,\boldsymbol{x}_R/\,\|\boldsymbol{x}_R\|_2), & \text{if } \|\boldsymbol{x}_R\|_2 \geq R. \end{cases} \tag{6.1.6}$$

It can be shown that the right-hand side function is Lipschitz with Lipschitz constant $L = \max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2 \leq R} \|\nabla \boldsymbol{f}(t, \boldsymbol{x})\|$ and has the same solutions provided $\|\boldsymbol{x}_R(t)\|_2 \leq R$ for all $t$. Thus if $\|\boldsymbol{x}_0\|_2 < R$, then there is an $\epsilon > 0$ depending only on $\|\boldsymbol{x}_0\|_2$, $R$ and $\max_{\boldsymbol{x}:\|\boldsymbol{x}\|_2 \leq R} \|\boldsymbol{f}(t, \boldsymbol{x})\|_2$ where there is a solution $\boldsymbol{x}(t)$ to $d\boldsymbol{x}/dt = \boldsymbol{f}(t, \boldsymbol{x}(t))$, $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$ for $t_0 \leq t \leq t + \epsilon$. If the solution can be guaranteed to remain bounded, then we can prove existence for all $t$.

**Example 6.5** An example are the Euler equations for the rotation of rigid body. If $\boldsymbol{\omega}$ is the angular velocity vector of a rigid body relative to co-ordinates fixed in the body, then

$$\frac{d\boldsymbol{\omega}}{dt} = J^{-1}\left[\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{\tau}\right] \tag{6.1.7}$$

where $J$ is the moment of inertia matrix, a symmetric positive definite $3 \times 3$ matrix, and $\boldsymbol{\tau}$ is the external torque in fixed-body co-ordinates. The "$\times$" is the cross product for three dimensional vectors. The right-hand side function $\boldsymbol{\omega} \mapsto J^{-1}\left[\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{\tau}\right]$ is *not* Lipschitz: it is quadratic in $\boldsymbol{\omega}$ and so the Jacobian matrix $\nabla_{\boldsymbol{\omega}}\left(J^{-1}\left[\boldsymbol{\omega} \times J\boldsymbol{\omega} + \boldsymbol{\tau}\right]\right)$ is a non-zero linear function of $\boldsymbol{\omega}$, and so is unbounded as $\|\boldsymbol{\omega}\| \to \infty$.

On the other hand,

$$\frac{d}{dt}\left(\frac{1}{2}\boldsymbol{\omega}^T J\,\boldsymbol{\omega}\right) = \boldsymbol{\omega}^T J\,\frac{d\boldsymbol{\omega}}{dt} = \boldsymbol{\omega}^T\,[\boldsymbol{\omega}\times J\boldsymbol{\omega}+\boldsymbol{\tau}]$$
$$= (J\boldsymbol{\omega})^T\,[\boldsymbol{\omega}\times\boldsymbol{\omega}] + \boldsymbol{\omega}^T\boldsymbol{\tau} = \boldsymbol{\omega}^T\boldsymbol{\tau},$$

using the rules for cross products that $\boldsymbol{a}^T(\boldsymbol{b}\times\boldsymbol{c}) = \boldsymbol{b}^T(\boldsymbol{c}\times\boldsymbol{a})$ and $\boldsymbol{a}\times\boldsymbol{a} = \boldsymbol{0}$. If $\boldsymbol{\tau}(t) = \boldsymbol{0}$ for all $t$, then $\frac{1}{2}\boldsymbol{\omega}^T J\,\boldsymbol{\omega}$ is constant. Since $J$ is positive definite, this means that $\boldsymbol{\omega}(t)$ is bounded for all $t$. The Jacobian matrix is bounded on this bounded set $\left\{\,\boldsymbol{\omega} \mid \frac{1}{2}\boldsymbol{\omega}^T J\,\boldsymbol{\omega} = \text{constant}\,\right\}$, and so there exists a unique solution to the Euler equations for all $t$.

#### 6.1.1.1 Gronwall Lemmas: Continuous and Discrete

A way of showing boundedness of solutions is to use a Gronwall lemma. The original Gronwall lemma is due to Gronwall [112] and was extended by Bellman [19], LaSalle [156], and Bihari [22]. These extended results can be summarized in the following theorem.

**Lemma 6.6** *(Generalized Gronwall lemma) If*

$$\frac{dr}{dt}(t) \le \varphi(r)\,\psi(t), \qquad \textit{for all } t, \textit{ and} \qquad r(t_0) = r_0$$

*where $\varphi$ is continuous positive function and $\psi$ integrable, then for all $t \ge t_0$ we have*

$$r(t) \le \rho(t) \qquad \textit{for all } t \ge t_0, \textit{ where}$$
$$\frac{d\rho}{dt} = \varphi(\rho)\,\psi(t), \qquad \rho(t_0) = r_0.$$

***Proof*** Let $G(r) = \int_{r_0}^{r} ds/\varphi(s)$. This is a differentiable increasing function. Then

$$\frac{d}{dt}G(r(t)) = \frac{1}{\varphi(r(t))}\frac{dr}{dt}(t) \le \psi(t),$$
$$\frac{d}{dt}G(\rho(t)) = \frac{1}{\varphi(\rho(t))}\frac{d\rho}{dt}(t) = \psi(t).$$

Integrating the right-hand side gives

$$G(r(t)) - G(r_0) \le \int_{t_0}^{t} \psi(s)\,ds = G(\rho(t)) - G(r_0), \qquad \text{so}$$
$$r(t) \le \rho(t) \qquad \text{for all } t \ge t_0,$$

as we wanted. □

For numerical methods, we have iterations where we want to show that the iterates are bounded in some way. For this, there are discrete versions of Theorem 6.6 such as the following.

**Lemma 6.7** *Suppose that*

$$r_k \le r_{k+1} \le r_k + \varphi(r_k)\left[\Psi(t_{k+1}) - \Psi(t_k)\right] \qquad \text{for } k = 0, 1, 2, 3, \ldots$$

*where $\varphi$ is continuous positive non-decreasing function and $\Psi' = \psi \ge 0$ is integrable, with $t_0 < t_1 < t_2 < \cdots$. Then*

$$r_k \le \rho(t_k) \qquad \text{where}$$
$$\frac{d\rho}{dt} = \varphi(\rho)\,\psi(t), \qquad \rho(t_0) = r_0.$$

***Proof*** Let $\widehat{r}(s)$ be the piecewise linear interpolant of $\widehat{r}(\Psi(t_k)) = r_k$. Then $\widehat{r}$ is absolutely continuous and for $\Psi(t_k) < s < \Psi(t_{k+1})$,

$$\frac{d\widehat{r}}{dt}(s) = \frac{r_{k+1} - r_k}{\Psi(t_{k+1}) - \Psi(t_k)} \le \varphi(r_k) \le \varphi(\widehat{r}(s))$$

as $r_k \le \widehat{r}(s)$ for $s \ge \Psi(t_k)$. Note that if $\Psi(t_{k+1}) = \Psi(t_k)$ then $r_{k+1} = r_k$, and we can ignore the interval $[t_k, t_{k+1}]$. Applying Lemma 6.6 gives

$$\widehat{r}(s) \le \widehat{\rho}(s) \qquad \text{where}$$
$$\frac{d\widehat{\rho}}{ds} = \varphi(\widehat{\rho}), \qquad \widehat{\rho}(\Psi(t_0)) = r_0.$$

Note that $\rho(t) = \widehat{\rho}(\Psi(t))$ is the solution to

$$\frac{d\rho}{dt} = \varphi(\rho)\,\psi(t), \qquad \rho(t_0) = r_0,$$

which gives the desired result. □

The discrete Gronwall lemma (Lemma 6.7) is useful for bounding numerical solutions for differential equations.

### 6.1.2 Euler's Method and Its Analysis

Euler's method for the initial value problem

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

is based on the approximation

$$\frac{\boldsymbol{x}(t+h)-\boldsymbol{x}(t)}{h} \approx \boldsymbol{f}(t,\boldsymbol{x}(t)).$$

Set $t_k = t_0 + k\,h$; then the method is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\,\boldsymbol{f}(t_k,\boldsymbol{x}_k), \qquad k = 0, 1, 2, \ldots \tag{6.1.8}$$

for computing $\boldsymbol{x}_k \approx \boldsymbol{x}(t_k)$.

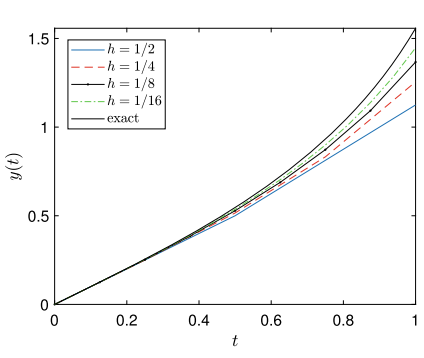
We expect to get greater accuracy if we reduce the step size $h$, but doing so means we need to use more steps and more function evaluations. To integrate from $t = a$ to $t = b$ we need $n = \lceil (b-a)/h \rceil$ steps. Note that $\lceil z \rceil$ is the smallest integer $\geq z$, also known as the *ceiling* of $z$. Each step contributes to the total error; we hope that the contribution that each step makes to the total error goes to zero faster than $(\text{constant}/n)$.

As an example, take the numerical solutions of $dy/dt = 1 + y^2$ for $y(0) = 0$ over the interval $[0, 1]$ using Euler's method. Some are shown in Figure 6.1.1.

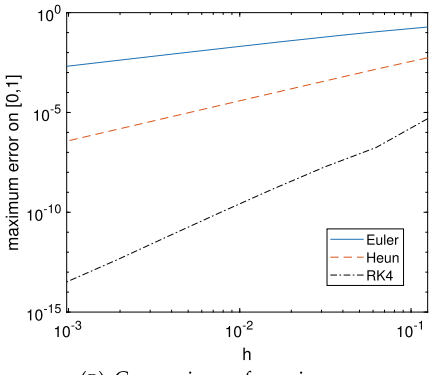
An important variant of Euler's method is the *implicit Euler method*, or *backward Euler method*:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\,\boldsymbol{f}(t_{k+1},\boldsymbol{x}_{k+1}), \qquad k = 0, 1, 2, \ldots \tag{6.1.9}$$

for computing $\boldsymbol{x}_k \approx \boldsymbol{x}(t_k)$. As with other implicit methods, we have to solve an equation for $\boldsymbol{x}_{k+1}$, which may be nonlinear. In spite of the computational difficulties involved with this, the implicit Euler method has some excellent stability properties that are discussed more in Section 6.1.6.




(A) Euler's method




(B) Comparison of maximum errors

**Fig. 6.1.1** Solutions via Euler's method for $dy/dt = 1 + y^2$, $y(0) = 0$

### 6.1.2.1 Error Analysis for Euler's Method

To estimate the error, we apply Taylor series with second-order remainder in integral form to the exact solution:

$$\boldsymbol{x}(t_{k+1}) = \boldsymbol{x}(t_k) + \frac{d\boldsymbol{x}}{dt}(t_k)\, h + \int_{t_k}^{t_{k+1}} (t_{k+1} - t) \frac{d^2\boldsymbol{x}}{dt^2}(t)\, dt. \tag{6.1.10}$$

The remainder term $\int_{t_k}^{t_{k+1}} (t_{k+1} - t)\boldsymbol{x}''(t)\, dt$ is the error we would incur with Euler's method if $\boldsymbol{x}_k = \boldsymbol{x}(t_k)$ were exact. It is called the *local truncation error* (LTE). The size of the LTE can be bounded by

$$\begin{aligned}
\left\| \int_{t_k}^{t_{k+1}} (t_{k+1} - t) \frac{d^2\boldsymbol{x}}{dt^2}(t)\, dt \right\| &\le \int_{t_k}^{t_{k+1}} \left\| (t_{k+1} - t) \frac{d^2\boldsymbol{x}}{dt^2}(t) \right\| dt \\
&\le \int_{t_k}^{t_{k+1}} (t_{k+1} - t) \max_{t_k \le c \le t_{k+1}} \left\| \frac{d^2\boldsymbol{x}}{dt^2}(c) \right\| dt \\
&= \frac{1}{2} h^2 \max_{t_k \le c \le t_{k+1}} \left\| \frac{d^2\boldsymbol{x}}{dt^2}(c) \right\|.
\end{aligned}$$

So we can write the LTE as

$$\int_{t_k}^{t_{k+1}} (t_{k+1} - t) \frac{d^2\boldsymbol{x}}{dt^2}(t)\, dt = \frac{1}{2} h^2 \boldsymbol{\eta}_k \quad \text{with } \|\boldsymbol{\eta}_k\| \le \max_{t_k \le c \le t_{k+1}} \left\| \frac{d^2\boldsymbol{x}}{dt^2}(c) \right\|.$$

Then (6.1.10) can be written as

$$\boldsymbol{x}(t_{k+1}) = \boldsymbol{x}(t_k) + h \frac{d\boldsymbol{x}}{dt}(t_k) + \frac{1}{2} h^2 \boldsymbol{\eta}_k.$$

Euler's method is

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\, \boldsymbol{f}(t_k, \boldsymbol{x}_k).$$

Subtracting and using the error $\boldsymbol{e}_j = \boldsymbol{x}(t_j) - \boldsymbol{x}_j$ we get

$$\boldsymbol{e}_{k+1} = \boldsymbol{e}_k + h \left[ \boldsymbol{f}(t_k, \boldsymbol{x}(t_k)) - \boldsymbol{f}(t_k, \boldsymbol{x}_k) \right] + \frac{1}{2} h^2 \boldsymbol{\eta}_k.$$

Taking norms and using the usual inequalities for norms we get

$$\begin{aligned}
\|\boldsymbol{e}_{k+1}\| &\le \|\boldsymbol{e}_k\| + h \|\boldsymbol{f}(t_k, \boldsymbol{x}(t_k)) - \boldsymbol{f}(t_k, \boldsymbol{x}_k)\| + \frac{1}{2} h^2 \|\boldsymbol{\eta}_k\| \\
&\le \|\boldsymbol{e}_k\| + h\, L\, \|\boldsymbol{x}(t_k) - \boldsymbol{x}_k\| + \frac{1}{2} h^2 \|\boldsymbol{\eta}_k\| \\
&= (1 + h\, L)\, \|\boldsymbol{e}_k\| + \frac{1}{2} h^2 M
\end{aligned} \tag{6.1.11}$$

where $M = \max_t \left\| \boldsymbol{x}''(t) \right\|$ with the maximum taken over the interval of integration.

Assuming the initial value $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$ is exactly correct, we get $\boldsymbol{e}_0 = \mathbf{0}$. Then we can apply (6.1.11) inductively to get a bound for many steps:

$$\begin{aligned}\|\boldsymbol{e}_k\| &\le (1 + h\,L)^k \,\|\boldsymbol{e}_0\| + \sum_{j=0}^{k-1} (1 + h\,L)^j \,\frac{1}{2} h^2 M \\ &= \frac{(1 + h\,L)^k - 1}{(1 + h\,L) - 1} \frac{1}{2} h^2 M = \frac{1}{2} h \left[(1 + h\,L)^k - 1\right] \frac{M}{L}.\end{aligned}$$

Using the bound

$$1 + h\,L \le 1 + h\,L + \frac{1}{2!}(h\,L)^2 + \frac{1}{3!}(h\,L)^3 + \cdots = e^{h\,L}$$

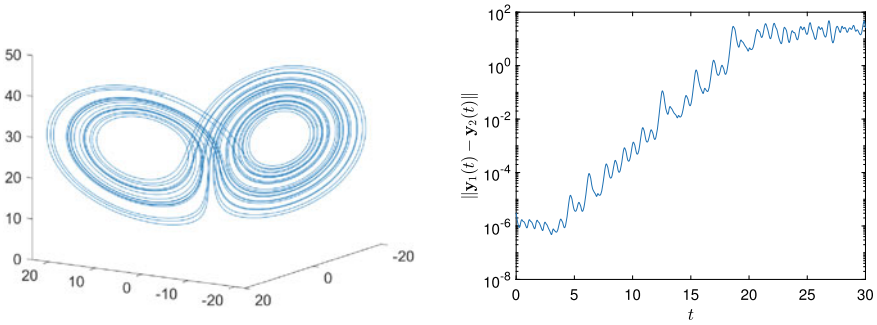
we obtain

$$\|\boldsymbol{e}_k\| \le \frac{1}{2} h \left[(e^{h\,L})^k - 1\right] \frac{M}{L} = \frac{1}{2} h \left[e^{L(t_k - t_0)} - 1\right] \frac{M}{L}. \tag{6.1.12}$$

Provided we consider integrating the differential equation over a fixed interval $[t_0, T]$, then we can say that the error is $\mathcal{O}(h)$.

For the differential equations $dy/dt = 1 + y^2$ for $y(0) = 0$, the errors at $t = 1$ for Euler's method for different values of $h$ are shown in Figure 6.1.1(b) along with the errors for some methods discussed in the following sections Heun's method and the standard 4th order Runge–Kutta method. The slope on the log–log plot for Euler's method from $h = 2^{-3}$ to $h = 2^{-10}$ is about 0.934, which is entirely consistent with an error of $\mathcal{O}(h)$.

However, integrating a differential over a long time period can result in exponential growth in the error due to the factor $e^{L(t_k - t_0)}$. This exponential growth is obvious in unstable differential equations such as $dx/dt = \lambda\, x$ with $\lambda > 0$. In this example, the size of the error also grows proportionate to the size of the solution. But there are many differential equations, often described as being "chaotic", where the size of the solution remains bounded, but the differential equation is persistently unstable. Examples of such equations include the equations E. Lorenz [166] developed to understand weather. The realization came that there are persistently unstable systems where a small perturbation at one place and time can be amplified over time to cause large changes in the solution later. It has been summarized as the *butterfly effect*: perhaps the beat of a butterfly's wings in South America could be amplified over months to result in a hurricane in Florida a year later. No-one, of course, has been able to demonstrate this as there are a vast number of small perturbations, not to mention billions of butterflies, that trying to track their influence on a large and unstable system like the weather is essentially impossible. However, the fact of exponential growth of perturbations cannot be avoided. This means that numerical errors can also grow exponentially after they have occurred.




(A) Trajectory (B) Amplification of differences

**Fig. 6.1.2** Solutions of Lorenz' equations

To illustrate the effects of persistent instability, Figure 6.1.2 shows a numerical trajectory for Lorenz' equations, and the difference between solutions with two initial values that differ by about $3 \times 10^{-6}$. While the amplification of errors is somewhat erratic on small time scales, the overall trend of exponential growth of the difference between two solutions until the differences are of order one is clear.

Lorenz' equations are

$$\frac{dx}{dt} = \sigma(-x + y), \tag{6.1.13}$$

$$\frac{dy}{dt} = -xz + \rho x - y, \tag{6.1.14}$$

$$\frac{dz}{dt} = xy - \beta z, \tag{6.1.15}$$

with standard values $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

### *6.1.3 Improving on Euler: Trapezoidal, Midpoint, and Heun*

One way of thinking about improving on Euler's method is to consider the equation

$$\boldsymbol{x}(t_{k+1}) = \boldsymbol{x}(t_k) + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}(t))\, dt,$$

and try to find a more accurate way to approximate the integral. Euler's method essentially uses the rectangle rule to estimate the integral. By using a second-order integration method, we should obtain greater accuracy. There are two main ways to do this:

$$\int_{t_k}^{t_{k+1}} f(t)\,dt \approx \frac{t_{k+1}-t_k}{2}\left[f(t_k)+f(t_{k+1})\right] \qquad \text{trapezoidal rule}$$
$$\int_{t_k}^{t_{k+1}} f(t)\,dt \approx (t_{k+1}-t_k)\, f\left(\frac{t_k+t_{k+1}}{2}\right) \qquad \text{mid-point rule.}$$

These give two *implicit methods*, the implicit trapezoidal rule and the implicit mid-point rule:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h}{2}\left[\boldsymbol{f}(t_k,\boldsymbol{x}_k)+\boldsymbol{f}(t_{k+1},\boldsymbol{x}_{k+1})\right] \qquad \text{trapezoidal rule}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\,\boldsymbol{f}(\frac{1}{2}(t_k+t_{k+1}),\frac{1}{2}(\boldsymbol{x}_k+\boldsymbol{x}_{k+1})) \qquad \text{mid-point rule}$$

The difficulty with these methods is that they are *implicit*; that is, the solution value at the end of the time step $\boldsymbol{x}_{k+1}$ is in the *right-hand side* as an argument to $\boldsymbol{f}$ as well as in the left-hand side. This means that there is a system of equations to solve rather than just a function to evaluate. Methods that just require function evaluations $\boldsymbol{f}(t,\boldsymbol{x})$ and forming linear combinations are called *explicit methods*. Euler's method is an explicit method.

Implicit methods are more complex to implement than explicit methods. In general, implicit methods require solvers, which may be simple fixed-point iterations, versions of Newton's method, or some hybrid of these. Other issues that arise include error tolerances for solvers, how to incorporate knowledge of the problems including preconditioners and specialized iterative methods.

Because of these difficulties, it is often easier to develop an explicit method that captures the desired order without resorting to implicit equations to be solved. Heun [123] found a way to keep the order by replacing $\boldsymbol{f}(t_{k+1},\boldsymbol{x}_{k+1})$ with $\boldsymbol{f}(t_{k+1},\boldsymbol{z}_{k+1})$ where $\boldsymbol{z}_{k+1}$ is a *first-order* approximation to $\boldsymbol{x}_{k+1}$. This first-order approximation can come from Euler's method. This gives Heun's method:

$$\boldsymbol{z}_{k+1} = \boldsymbol{x}_k + h\,\boldsymbol{f}(t_k,\boldsymbol{x}_k), \tag{6.1.16}$$
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \frac{h}{2}\left[\boldsymbol{f}(t_k,\boldsymbol{x}_k)+\boldsymbol{f}(t_{k+1},\boldsymbol{z}_{k+1})\right]. \tag{6.1.17}$$

There is one more second-order method that we will mention here: the *leap-frog method*. This uses the previous two values $\boldsymbol{x}_{k-1}$ and $\boldsymbol{x}_k$ to compute $\boldsymbol{x}_{k+1}$ by a version of the mid-point rule:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_{k-1} + 2h\,\boldsymbol{f}(t_k,\boldsymbol{x}_k). \tag{6.1.18}$$

This is also an explicit method. This method is rarely used because of stability issues, which will be discussed later.

Figure 6.1.1(b) shows that maximum error for Heun's method for $dy/dt = 1 + y^2$, $y(0) = 0$.

Three of these methods, the implicit mid-point rule, the implicit trapezoidal rule, and Heun's method, are all *Runge–Kutta methods*. The leap-frog method, on the other hand, is a multistep method. We will discuss the Runge–Kutta family of methods more systematically in Section 6.1.4 and multistep methods in Section 6.1.5.

### 6.1.4 Runge–Kutta Methods

Carl Runge [222] and Martin Wilhelm Kutta [150] developed a fourth order explicit method for solving differential equations:

$$
\begin{aligned}
\boldsymbol{v}_{k,1} &= \boldsymbol{f}(t_k, \boldsymbol{x}_k) \\
\boldsymbol{v}_{k,2} &= \boldsymbol{f}(t_k + \frac{1}{2}h,\ \boldsymbol{x}_k + \frac{1}{2}h\boldsymbol{v}_{k,1}), \\
\boldsymbol{v}_{k,3} &= \boldsymbol{f}(t_k + \frac{1}{2}h,\ \boldsymbol{x}_k + \frac{1}{2}h\boldsymbol{v}_{k,2}), \\
\boldsymbol{v}_{k,4} &= \boldsymbol{f}(t_k + \quad h,\ \boldsymbol{x}_k + \quad h\boldsymbol{v}_{k,3}), \\
\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \frac{1}{6}h\left[\boldsymbol{v}_{k,1} + 2\boldsymbol{v}_{k,2} + 2\boldsymbol{v}_{k,3} + \boldsymbol{v}_{k,4}\right].
\end{aligned}
\tag{6.1.19}
$$

Along with Euler's method, the implicit mid-point rule, trapezoidal rule rules, and Heun's method, this is a *single-step method* as computing $\boldsymbol{x}_{k+1}$ only requires $\boldsymbol{x}_k$, unlike multistep methods which use more prior values $\boldsymbol{x}_k, \boldsymbol{x}_{k-1}, \ldots, \boldsymbol{x}_{k-p}$.

Butcher [44, 46] developed a framework for the analysis of these methods. First, we need a consistent way of representing these methods. Butcher used *Butcher tableaus*: the Runge–Kutta method

$$
\boldsymbol{v}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{k,i}) \qquad j = 1, 2, \ldots, s,
\tag{6.1.20}
$$

$$
\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\sum_{j=1}^{s} b_j\,\boldsymbol{v}_{k,j}
\tag{6.1.21}
$$

is represented by the tableau

$$
\begin{array}{c|cccc}
c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\
c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\
\hline
 & b_1 & b_2 & \cdots & b_s
\end{array}
\quad \text{or} \quad
\begin{array}{c|c}
\boldsymbol{c} & A \\
\hline
 & \boldsymbol{b}^T
\end{array}.
\tag{6.1.22}
$$

The integer $s$ is the number of *stages* of the Runge–Kutta method.

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array}$$

(A) Euler's method

$$\begin{array}{c|cc} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \hline & & 1 \end{array}$$

(B) Implicit mid-point rule

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & \frac{1}{2} & \frac{1}{2} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(C) Implicit trapezoidal method

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

(D) Heun's method

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & & \frac{1}{2} & & \\ 1 & & & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

(E) Standard 4th order Runge–Kutta method

**Fig. 6.1.3** Butcher tableaus of well-known Runge–Kutta methods

The single-step methods we have already seen can be put in this form. Figure 6.1.3 shows them.

One property that is very easily determined from the Butcher tableau is if the method is implicit: the Butcher tableau of an explicit method has $A$ strictly lower triangular, possibly after a permutation of the stages.

#### 6.1.4.1 Solvability of the Runge–Kutta Equations

For the implicit Runge–Kutta methods, we would like to see that there are solutions of the Runge–Kutta equations (6.1.20), at least for sufficiently small $h$ provided $\boldsymbol{f}$ is Lipschitz. Consider using the simple fixed-point iteration for (6.1.20):

$$\boldsymbol{v}_{k,j}^{(p+1)} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{v}_{k,i}^{(p)}) \qquad j = 1, 2, \ldots, s,$$

where $p$ is the iteration index. To see if this is a contraction mapping (see Theorem 3.3), we consider

$$\boldsymbol{u}_{k,j}^{(p+1)} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{u}_{k,i}^{(p)}) \qquad j = 1, 2, \ldots, s;$$

we wish to bound $\left\| \boldsymbol{v}_{k,j}^{(p+1)} - \boldsymbol{u}_{k,j}^{(p+1)} \right\|$ in terms of $\left\| \boldsymbol{v}_{k,i}^{(p)} - \boldsymbol{u}_{k,i}^{(p)} \right\|$:

$$\left\| \boldsymbol{v}_{k,j}^{(p+1)} - \boldsymbol{u}_{k,j}^{(p+1)} \right\| \le \sum_{i=1}^{s} L\,h\ |a_{ji}| \left\| \boldsymbol{v}_{k,i}^{(p)} - \boldsymbol{u}_{k,i}^{(p)} \right\|$$
$$\le L\,h\ \|A\|_\infty \max_i \left\| \boldsymbol{v}_{k,i}^{(p)} - \boldsymbol{u}_{k,i}^{(p)} \right\|.$$

Provided $L\,h\ \|A\|_\infty < 1$ the mapping is a contraction mapping, and so there is a unique solution to (6.1.20) for sufficiently small $h$. Substituting this solution into (6.1.21) gives $\boldsymbol{x}_{k+1}$ in terms of $\boldsymbol{v}_{k,j}$ for $j = 1, 2, \ldots, s$, which in turn are implicitly determined by $\boldsymbol{x}_k$. This gives $\boldsymbol{x}_{k+1} = \boldsymbol{\Phi}_h(\boldsymbol{x}_k)$. The function $\boldsymbol{\Phi}_h$ clearly depends on the Runge–Kutta method and $\boldsymbol{f}$ as well as the step size $h$.

We will want to solve these equations even if $L\,h\ \|A\|_\infty > 1$. There will be more on this case in Section 6.1.6 on stiff differential equations.

### 6.1.4.2 Error Analysis

The *local truncation error* (LTE) of a Runge–Kutta method is the difference between the result of applying one step of the method to $\boldsymbol{x}_k$ and the true solution of the differential equation with $\boldsymbol{x}(t_k) = \boldsymbol{x}_k$ at time $t_{k+1}$: $\boldsymbol{\tau}_k(\boldsymbol{x}_k) := \boldsymbol{x}(t_{k+1}; \boldsymbol{x}_k, t_k) - \boldsymbol{\Phi}_h(\boldsymbol{x}_k)$ where

$$\frac{d}{dt}\boldsymbol{x}(t; \boldsymbol{x}_k, t_k) = \boldsymbol{f}(t, \boldsymbol{x}(t; \boldsymbol{x}_k, t_k)), \qquad \boldsymbol{x}(t_k; \boldsymbol{x}_k, t_k) = \boldsymbol{x}_k.$$

We can estimate the LTE using the method developed by J. Butcher [45, 46] using so-called Butcher trees, as we will see below. First we will look at the amplification of errors in the Runge–Kutta method. Essentially we want to estimate the Lipschitz constant of $\boldsymbol{\Phi}_h$. Consider the Runge–Kutta equations (6.1.20) for two starting points $\boldsymbol{x}_k$ and $\boldsymbol{y}_k$:

$$\boldsymbol{v}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{k,i}) \qquad j = 1, 2, \ldots, s,$$
$$\boldsymbol{w}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{y}_k + h \sum_{i=1}^{s} a_{ji}\boldsymbol{w}_{k,i}) \qquad j = 1, 2, \ldots, s, \qquad \text{so}$$
$$\left\| \boldsymbol{v}_{k,j} - \boldsymbol{w}_{k,j} \right\| \le L \left[ \left\| \boldsymbol{x}_k - \boldsymbol{y}_k \right\| + h \sum_{i=1}^{s} |a_{ji}| \left\| \boldsymbol{v}_{k,i} - \boldsymbol{w}_{k,i} \right\| \right].$$
$$\text{Then } \max_j \left\| \boldsymbol{v}_{k,j} - \boldsymbol{w}_{k,j} \right\| \le \frac{L}{1 - hL\,\|A\|_\infty} \left\| \boldsymbol{x}_k - \boldsymbol{y}_k \right\|.$$

The results of one step of the Runge–Kutta method

$$\boldsymbol{\Phi}_h(\boldsymbol{x}_k) = \boldsymbol{x}_k + h \sum_{j=1}^{s} b_j \boldsymbol{v}_{k,j},$$
$$\boldsymbol{\Phi}_h(\boldsymbol{y}_k) = \boldsymbol{y}_k + h \sum_{j=1}^{s} b_j \boldsymbol{w}_{k,j},$$

can be compared:

$$\begin{aligned}\left\| \boldsymbol{\Phi}_h(\boldsymbol{x}_k) - \boldsymbol{\Phi}_h(\boldsymbol{y}_k) \right\| &\le \left\| \boldsymbol{x}_k - \boldsymbol{y}_k \right\| + h \sum_{j=1}^{s} \left| b_j \right| \left\| \boldsymbol{v}_{k,j} - \boldsymbol{w}_{k,j} \right\| \\ &\le \left( 1 + \frac{h\, L\, \|\boldsymbol{b}\|_1}{1 - h\, L\, \|A\|_\infty} \right) \left\| \boldsymbol{x}_k - \boldsymbol{y}_k \right\|.\end{aligned}$$

Provided $h\, L\, \|A\|_\infty \le 1/2$ we have the bound

$$\left\| \boldsymbol{\Phi}_h(\boldsymbol{x}_k) - \boldsymbol{\Phi}_h(\boldsymbol{y}_k) \right\| \le (1 + 2L\, \|\boldsymbol{b}\|_1\, h) \left\| \boldsymbol{x}_k - \boldsymbol{y}_k \right\|. \tag{6.1.23}$$

#### 6.1.4.3 Butcher Trees and the Local Truncation Error

The keys to estimating the local truncation error (LTE) are Taylor series: the Taylor series of $\boldsymbol{f}(t + c\,h, \boldsymbol{x} + h\,\boldsymbol{v})$, and the Taylor series of the solution $\boldsymbol{x}(t + h)$. To avoid having to deal with derivatives of $\boldsymbol{f}(t + c\,h, \boldsymbol{x} + h\,\boldsymbol{v})$ with respect to both $t$ and $\boldsymbol{x}$, we replace the differential equation with an autonomous equation:

$$\frac{d}{dt} \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix} = \begin{bmatrix} \boldsymbol{f}(t, \boldsymbol{x}) \\ 1 \end{bmatrix}, \qquad \left. \begin{bmatrix} \boldsymbol{x} \\ t \end{bmatrix} \right|_{t=t_0} = \begin{bmatrix} \boldsymbol{x}_0 \\ t_0 \end{bmatrix} \tag{6.1.24}$$

gives an initial value problem $d\boldsymbol{z}/dt = \widetilde{\boldsymbol{f}}(\boldsymbol{z})$, $\boldsymbol{z}(t_0) = \boldsymbol{z}_0$ where $\boldsymbol{z}(t) = [\boldsymbol{x}(t)^T,\ t]^T$. Note that matching the Runge–Kutta method for (6.1.24) to the Runge–Kutta method for the original differential equation gives

$$\sum_{i=1}^{s} a_{ji} = c_j \qquad \text{for } j = 1, \ldots, s. \tag{6.1.25}$$

Another requirement on the Butcher tableau (6.1.22) to obtain correct solutions for $\boldsymbol{f}(t, \boldsymbol{x}) =$ constant is that

$$\sum_j b_j = 1. \tag{6.1.26}$$

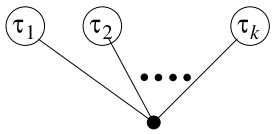
We drop the tilde in what follows, so we assume the initial value problem is in the form $d\boldsymbol{x}/dt = \boldsymbol{f}(\boldsymbol{x})$, $\boldsymbol{x}(t_0) = \boldsymbol{x}_0$. Using the notation (1.6.2) from Section 1.6.2 we can efficiently represent these Taylor series:

$$\boldsymbol{f}(\boldsymbol{x} + h\,\boldsymbol{v}) = \boldsymbol{f}(\boldsymbol{x}) + \sum_{k=1}^{m} \frac{1}{k!} h^k\, D^k \boldsymbol{f}(\boldsymbol{x})[\boldsymbol{v}, \boldsymbol{v}, \ldots, \boldsymbol{v}] + \mathcal{O}(h^{m+1}).$$

For the Taylor series expansion of $\boldsymbol{x}(t+h)$ we first note that $d\boldsymbol{x}/dt = \boldsymbol{f}(\boldsymbol{x})$; then

$$\begin{aligned}
\frac{d^2\boldsymbol{x}}{dt^2} &= \frac{d}{dt}\left(\frac{d\boldsymbol{x}}{dt}\right) = \frac{d}{dt}\left(\boldsymbol{f}(\boldsymbol{x})\right) = D^1\boldsymbol{f}(\boldsymbol{x})[\frac{d\boldsymbol{x}}{dt}] = D^1\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x})],\\
\frac{d^3\boldsymbol{x}}{dt^3} &= \frac{d}{dt}\left(\frac{d^2\boldsymbol{x}}{dt^2}\right) = \frac{d}{dt}\left(D^1\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x})]\right)\\
&= D^2\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x}),\ \boldsymbol{f}(\boldsymbol{x})] + D^1\boldsymbol{f}[D^1\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x})]], \qquad \text{etc.}
\end{aligned}$$

Butcher's insight was to represent these expressions using trees: let $expr(\tau)$ be the expression represented by tree $\tau$. We start with $expr(\bullet) = \boldsymbol{f}(\boldsymbol{x})$. The tree




where each $\tau_j$ is a tree, is recursively denoted $[\tau_1,\ \tau_2,\ \ldots,\ \tau_k]$. We recursively define

$$expr([\tau_1,\ \tau_2,\ \ldots,\ \tau_k]) = D^k\boldsymbol{f}(\boldsymbol{x})[expr(\tau_1),\ expr(\tau_2),\ \ldots,\ expr(\tau_k)]. \tag{6.1.27}$$

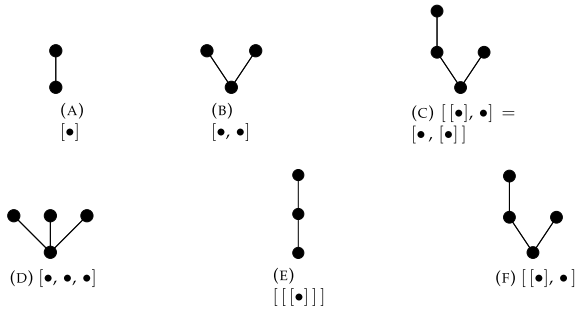
Since $D^k\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k] = D^k\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$ whenever $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k$ is a permutation of $\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k$, it follows that if $\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_k$ is a permutation of $\tau_1,\ \tau_2,\ \ldots,\ \tau_k$ then we can identify $[\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_k] = [\tau_1,\ \tau_2,\ \ldots,\ \tau_k]$. Identifying the tree with the expression, we can write

$$\frac{d}{dt}[\tau_1,\ \tau_2,\ \ldots,\ \tau_k] = [\bullet,\ \tau_1,\ \tau_2,\ \ldots,\ \tau_k] + \sum_{j=1}^{k} [\tau_1, \ldots,\ \frac{d\tau_j}{dt},\ \ldots,\ \tau_k].$$

And so,

$$\begin{aligned}
\frac{d\boldsymbol{x}}{dt} &= \boldsymbol{f}(\boldsymbol{x}) = \bullet,\\
\frac{d^2\boldsymbol{x}}{dt^2} &= \frac{d}{dt}\left(\boldsymbol{f}(\boldsymbol{x})\right) = D^1\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{f}(\boldsymbol{x})] = [\bullet],\\
\frac{d^3\boldsymbol{x}}{dt^3} &= \frac{d}{dt}\left([\bullet]\right) = [\bullet,\ \bullet] + [\,[\bullet]\,],\\
\frac{d^4\boldsymbol{x}}{dt^4} &= \frac{d}{dt}\left([\bullet,\ \bullet] + [\,[\bullet]\,]\right)
\end{aligned}$$

**Fig. 6.1.4** Butcher trees




$$= [\bullet,\ \bullet,\ \bullet] + [\,[\bullet],\ \bullet] + [\bullet,\ [\bullet]\,] + [\bullet,\ [\bullet]\,] + [\bullet,\ ([\bullet,\ \bullet] + [\,[\bullet]\,])\,]$$
$$= [\bullet,\ \bullet,\ \bullet] + 3\,[\bullet,\ [\bullet]\,] + [\bullet,\ [\bullet,\ \bullet]\,] + [\,[\,[\bullet]\,]\,] \qquad \text{etc.}$$

Readers with a stronger visual sense may find the diagrams in Figure 6.1.4 easier to understand.

We want the results of the Runge–Kutta methods to match the Taylor series expansions of the solution. So we need to expand $\boldsymbol{f}(\boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_i)$ in terms of the Butcher trees. Note that since the expressions $D^k\boldsymbol{f}(\boldsymbol{x})[\boldsymbol{w}_1, \ldots, \boldsymbol{w}_k]$ are linear in each $\boldsymbol{w}_j$, the summations $\sum_i a_{ji}\boldsymbol{v}_i$ can be expanded in terms of trees. Also, modifying the bracket notation for the trees to allow items that are not themselves Butcher trees, we write

$$\begin{aligned}\boldsymbol{f}(\boldsymbol{x}_k + h\boldsymbol{w}) &= \boldsymbol{f}(\boldsymbol{x}_k) + D^1\boldsymbol{f}(\boldsymbol{x})[h\boldsymbol{w}] + \frac{1}{2!}D^2\boldsymbol{f}(\boldsymbol{x}_k)[h\boldsymbol{w},\ h\boldsymbol{w}] + \\ &\quad + \frac{1}{3!}D^3\boldsymbol{f}(\boldsymbol{x}_k)[h\boldsymbol{w},\ h\boldsymbol{w},\ h\boldsymbol{w}] + \cdots \\ &= \ \bullet\ + h[\,\boldsymbol{w}\,] + \frac{1}{2!}h^2[\,\boldsymbol{w},\ \boldsymbol{w}\,] + \frac{1}{3!}h^3[\,\boldsymbol{w},\ \boldsymbol{w},\ \boldsymbol{w}\,] + \cdots .\end{aligned}$$

The Runge–Kutta equations (6.1.20) then become

$$\begin{aligned}\boldsymbol{v}_j &= \bullet + h[\sum_i a_{ji}\boldsymbol{v}_i\,] + \frac{1}{2}h^2[\sum_i a_{ji}\boldsymbol{v}_i,\ \sum_k a_{jk}\boldsymbol{v}_k\,] \\ &\quad + \frac{1}{6}h^3[\sum_i a_{ji}\boldsymbol{v}_i,\ \sum_k a_{jk}\boldsymbol{v}_k,\ \sum_\ell a_{j\ell}\boldsymbol{v}_\ell] + \cdots \\ &= \bullet + h\sum_i a_{ji}[\,\boldsymbol{v}_i\,] + \frac{1}{2}h^2\sum_i a_{ji}\sum_k a_{jk}[\,\boldsymbol{v}_i,\ \boldsymbol{v}_k\,] \\ &\quad + \frac{1}{6}h^3\sum_i a_{ji}\sum_k a_{jk}\sum_\ell a_{j\ell}[\,\boldsymbol{v}_i,\ \boldsymbol{v}_k,\ \boldsymbol{v}_\ell] + \cdots .\end{aligned}$$

Substituting recursively into this expression for $\boldsymbol{v}_j$ enables us to expand this expression to any order of $h$. To get the expansion to third order, we expand $\boldsymbol{v}_i$ to second order in the term with factor $h$, expand $\boldsymbol{v}_i$ and $\boldsymbol{v}_k$ to first order in the term with factor

$h^2$, and to zeroth order in the term with factor $h^3$. This gives

$$
\begin{aligned}
\boldsymbol{v}_j &= \bullet + h\sum_i a_{ji}\,[\,\bullet + h\sum_k a_{ik}[\,\boldsymbol{v}_k\,] + \frac{1}{2}h^2\sum_{k,\ell} a_{ik}a_{i\ell}[\,\boldsymbol{v}_k,\ \boldsymbol{v}_\ell\,]\,] \\
&\quad + \frac{1}{2}h^2\sum_{i,k} a_{ji}a_{jk}\,[\,\bullet + h\sum_p a_{ip}[\,\boldsymbol{v}_p\,],\ \bullet + h\sum_q a_{kq}[\,\boldsymbol{v}_q\,]\,] \\
&\quad + \frac{1}{6}h^3\sum_{i,k,\ell} a_{ji}a_{jk}a_{j\ell}\,[\,\bullet,\ \bullet,\ \bullet\,] + \mathcal{O}(h^4) \\
&= \bullet + h\sum_i a_{ji}\,[\,\bullet\,] + h^2\sum_{i,k} a_{ji}a_{ik}[\,[\,\boldsymbol{v}_k\,]\,] + \frac{1}{2}h^3\sum_{i,k,\ell} a_{ji}a_{ik}a_{i\ell}[\,[\,\boldsymbol{v}_k,\ \boldsymbol{v}_\ell\,]\,] \\
&\quad + \frac{1}{2}h^2\sum_{i,k} a_{ji}a_{jk}\,[\,\bullet,\ \bullet\,] + \frac{1}{2}h^3\sum_{i,k,p} a_{ji}a_{jk}a_{ip}[\,[\,\boldsymbol{v}_p\,],\ \bullet\,] \\
&\quad + \frac{1}{2}h^3\sum_{i,k,q} a_{ji}a_{jk}a_{iq}[\,\bullet,\ [\,\boldsymbol{v}_q\,]\,] + \frac{1}{6}h^3\sum_{i,k,\ell} a_{ji}a_{jk}a_{j\ell}\,[\,\bullet,\ \bullet,\ \bullet\,] + \mathcal{O}(h^4) \\
&= \bullet + h\sum_i a_{ji}\,[\,\bullet\,] + h^2\sum_{i,k} a_{ji}a_{ik}[\,[\,\bullet\,]\,] + h^3\sum_{i,k,\ell} a_{ji}a_{ik}a_{k\ell}[\,[\,[\,\bullet\,]\,]\,] \\
&\quad + \frac{1}{2}h^3\sum_{i,k,\ell} a_{ji}a_{ik}a_{i\ell}[\,[\,\bullet,\ \bullet\,]\,] + \frac{1}{2}h^2\sum_{i,k} a_{ji}a_{jk}\,[\,\bullet,\ \bullet\,] \\
&\quad + \frac{1}{2}h^3\sum_{i,k,p} a_{ji}a_{jk}a_{ip}[\,[\,\bullet\,],\ \bullet\,] + \frac{1}{2}h^3\sum_{i,k,q} a_{ji}a_{jk}a_{iq}[\,\bullet,\ [\,\bullet\,]\,] \\
&\quad + \frac{1}{6}h^3\sum_{i,k,\ell} a_{ji}a_{jk}a_{j\ell}\,[\,\bullet,\ \bullet,\ \bullet\,] + \mathcal{O}(h^4) \\
&= \bullet + h\sum_i a_{ji}\,[\,\bullet\,] + h^2\left\{\sum_{i,k} a_{ji}a_{ik}[\,[\,\bullet\,]\,] + \frac{1}{2}\sum_{i,k} a_{ji}a_{jk}\,[\,\bullet,\ \bullet\,]\right\} \\
&\quad + h^3\left\{\sum_{i,k,\ell} a_{ji}a_{ik}a_{k\ell}[\,[\,[\,\bullet\,]\,]\,] + \frac{1}{2}\sum_{i,k,\ell} a_{ji}a_{ik}a_{i\ell}[\,[\,\bullet,\ \bullet\,]\,]\right. \\
&\qquad \left. + \sum_{i,k,p} a_{ji}a_{jk}a_{ip}[\,[\,\bullet\,],\ \bullet\,] + \frac{1}{6}\sum_{i,k,\ell} a_{ji}a_{jk}a_{j\ell}\,[\,\bullet,\ \bullet,\ \bullet\,]\right\} + \mathcal{O}(h^4).
\end{aligned}
$$

Now we note that

$$
\begin{aligned}
\boldsymbol{x}_{m+1} &= \boldsymbol{x}_m + h\sum_{j=1}^{s} b_j \boldsymbol{v}_j \\
&= \boldsymbol{x}_m + h\sum_{j=1}^{s} b_j \bullet + h^2 \sum_{i,j} b_j a_{ji} [\bullet] \\
&\quad + h^3 \left\{ \sum_{i,j,k} b_j a_{ji} a_{ik} [[\bullet]] + \frac{1}{2} \sum_{i,j,k} b_j a_{ji} a_{jk} [\bullet, \bullet] \right\} \\
&\quad + h^4 \left\{ \sum_{i,j,k,\ell} b_j a_{ji} a_{ik} a_{k\ell} [[[\bullet]]] + \frac{1}{2} \sum_{i,j,k,\ell} b_j a_{ji} a_{ik} a_{i\ell} [[\bullet, \bullet]] \right. \\
&\quad \left. + \sum_{i,j,k,p} b_j a_{ji} a_{jk} a_{ip} [[\bullet], \bullet] + \frac{1}{6} \sum_{i,j,k,\ell} b_j a_{ji} a_{jk} a_{j\ell} [\bullet, \bullet, \bullet] \right\} + \mathcal{O}(h^5).
\end{aligned} \tag{6.1.28}
$$

There are some rules we can use to simplify these results. The main rule we use is (6.1.25): $c_j = \sum_i a_{ji}$. This simplifies the previous (long) Taylor expansion above to

$$
\begin{aligned}
\boldsymbol{x}_{m+1} &= \boldsymbol{x}_m + h\sum_{j=1}^{s} b_j \bullet + h^2 \sum_{j} b_j c_j [\bullet] \\
&\quad + h^3 \left\{ \sum_{i,j} b_j a_{ji} c_i [[\bullet]] + \frac{1}{2} \sum_{j} b_j c_j^2 [\bullet, \bullet] \right\} \\
&\quad + h^4 \left\{ \sum_{i,j,k} b_j a_{ji} a_{ik} c_k [[[\bullet]]] + \frac{1}{2} \sum_{i,j,k,\ell} b_j a_{ji} c_i^2 [[\bullet, \bullet]] \right. \\
&\quad \left. + \sum_{i,j} b_j a_{ji} c_j c_i [[\bullet], \bullet] + \frac{1}{6} \sum_{j} b_j c_j^3 [\bullet, \bullet, \bullet] \right\} + \mathcal{O}(h^5).
\end{aligned}
$$

Comparing with the Taylor series of the exact solution, we can identify the conditions for obtaining a given order for the local truncation error up to $\mathcal{O}(h^5)$:

$$
\begin{aligned}
\boldsymbol{x}(t_{m+1}) = \boldsymbol{x}(t_m) &+ h \bullet + \frac{1}{2} h^2 [\bullet] + \frac{1}{6} h^3 \{[\bullet, \bullet] + [[\bullet]]\} \\
&+ \frac{1}{24} h^4 \{[\bullet, \bullet, \bullet] + 3[[\bullet], \bullet] + [[\bullet, \bullet]] + [[[\bullet]]]\} \\
&+ \mathcal{O}(h^5).
\end{aligned}
$$

This gives the conditions in Table 6.1.1. If we use the *Hadamard* (or componentwise) *product*: $\boldsymbol{u} \circ \boldsymbol{v} = [u_1 v_1, u_2 v_2, \ldots, u_s v_s]^T$, these conditions can be written in a simplified way, as shown in Table 6.1.1.

**Table 6.1.1** Order conditions for Runge–Kutta methods

| Condition(s) | | Tree | Order |
|---|---|---|---|
| $\sum_j b_j = 1$ | $\boldsymbol{e}^T\boldsymbol{b} = 1$ | $\bullet$ | 1st |
| $\sum_j b_j c_j = \frac{1}{2}$ | $\boldsymbol{b}^T\boldsymbol{c} = \frac{1}{2}$ | $[\bullet]$ | 2nd |
| $\sum_{j,i} b_j a_{ji} c_i = \frac{1}{6}$ | $\boldsymbol{b}^T A\boldsymbol{c} = \frac{1}{6}$ | $[[\bullet]]$ | 3rd |
| $\sum_j b_j c_j^2 = \frac{1}{3}$ | $\boldsymbol{b}^T(\boldsymbol{c}\circ\boldsymbol{c}) = \frac{1}{3}$ | $[\bullet, \bullet]$ | |
| $\sum_{j,i} b_j a_{ji} c_i^2 = \frac{1}{12}$ | $\boldsymbol{b}^T A(\boldsymbol{c}\circ\boldsymbol{c}) = \frac{1}{12}$ | $[[\bullet, \bullet]]$ | 4th |
| $\sum_{i,j} b_j a_{ji} c_j c_i = \frac{1}{8}$ | $(\boldsymbol{b}\circ\boldsymbol{c})^T A\boldsymbol{c} = \frac{1}{8}$ | $[[\bullet], \bullet]$ | |
| $\sum_{j,i,k} b_j a_{ji} a_{ik} c_k = \frac{1}{24}$ | $\boldsymbol{b}^T A^2\boldsymbol{c} = \frac{1}{24}$ | $[[[\bullet]]]$ | |
| $\sum_j b_j c_j^3 = \frac{1}{24}$ | $\boldsymbol{b}^T(\boldsymbol{c}\circ\boldsymbol{c}\circ\boldsymbol{c}) = \frac{1}{24}$ | $[\bullet, \bullet, \bullet]$ | |

John Butcher analyzed the combinatorics of the Butcher trees and their application in [46, Chap. 3]. We start with $|\tau|$ which is the number of nodes of the Butcher tree $\tau$. Note that if $\tau = [\tau_1, \tau_2, \ldots, \tau_k]$ then $|\tau| = 1 + \sum_{j=1}^{k} |\tau_j|$, while $|\bullet| = 1$.

The Taylor series with remainder of order $m+1$ of the exact solution is

$$\boldsymbol{x}(t+h) = \boldsymbol{x}(t) + \sum_{\tau:|\tau|\le p} \frac{h^{|\tau|}}{\sigma(\tau)\,\gamma(\tau)}\, expr(\tau)|_{\boldsymbol{x}=\boldsymbol{x}(t)} + \mathcal{O}(h^{p+1}) \tag{6.1.29}$$

where $\sigma(\tau)$ and $\gamma(\tau)$ are combinatorial quantities that can be computed recursively by

$$\sigma([\underbrace{\tau^{(1)},\ldots,\tau^{(1)}}_{m_1},\underbrace{\tau^{(2)},\ldots,\tau^{(2)}}_{m_2},\ldots,\underbrace{\tau^{(r)},\ldots,\tau^{(r)}}_{m_r}]) = \prod_{i=1}^{r}\left(m_i!\,\sigma(\tau^{(i)})\right),$$

$$\gamma([\underbrace{\tau^{(1)},\ldots,\tau^{(1)}}_{m_1},\underbrace{\tau^{(2)},\ldots,\tau^{(2)}}_{m_2},\ldots,\underbrace{\tau^{(r)},\ldots,\tau^{(r)}}_{m_r}]) = |\tau| \prod_{i=1}^{r}\gamma(\tau^{(i)})^{m_i}, \qquad \text{and}$$

$$\sigma(\bullet) = \gamma(\bullet) = 1.$$

Note that $\sigma(\tau)$ is the order of the group of automorphisms of the Butcher tree. Since Butcher trees are rooted trees, every automorphism of $\tau$ must map the root to the root. For $\tau = [\underbrace{\bullet, \bullet, \ldots, \bullet}_{p-1 \text{ times}}]$ (a wide shrub that is only one edge high with $p$ nodes), $\sigma(\tau) = (p-1)!$ while $\gamma(\tau) = p$, giving $\sigma(\tau)\,\gamma(\tau) = p!$. On the other hand, for $\tau = \underbrace{[[\ldots,[}_{p-1 \text{ times}} \bullet \underbrace{]]\ldots,]}_{p-1 \text{ times}}$ (which also has $p$ nodes but is a tall branchless tree), $\sigma(\tau) = 1$ and $\gamma(\tau) = p!$, again giving $\sigma(\tau)\,\gamma(\tau) = p!$.

On the other hand, applying the Runge–Kutta method (6.1.20, 6.1.21) gives

$$\boldsymbol{x}_{m+1} = \boldsymbol{x}_m + \sum_{\tau:|\tau|\le p} h^{|\tau|} \frac{1}{\sigma(\tau)} \Phi(\tau\,;A,\boldsymbol{b})\ \ expr(\tau)|_{\boldsymbol{x}=\boldsymbol{x}_m} + \mathcal{O}(h^{p+1}). \tag{6.1.30}$$

Here $\Phi(\tau\,;A,\boldsymbol{b})$ is an *elementary weight* in the sense of Butcher, and corresponds to the expressions in (6.1.28) such as $\Phi([[\bullet],\bullet]\,;A,\boldsymbol{b}) = \sum_{i,j,k,p} b_j a_{ji} a_{jk} a_{ip}$. The general formula is

$$\Phi(\tau\,;A,\boldsymbol{b}) = \sum_{i_1,i_2,\ldots,i_p=1}^{s} b_{i_1} \prod_{(j,k)\in E(\tau)} a_{i_j i_k} \tag{6.1.31}$$

where $E(\tau)$ is the set of edges of $\tau$ directed away from the root; nodes of $\tau$ are labeled $1, 2, 3, \ldots, p$ where $p = |\tau|$, with node 1 being the root of $\tau$.

By matching the Taylor series expansions of the exact solution (6.1.29) and the Runge–Kutta method (6.1.30), we can see that the method has order $p$ if

$$\Phi(\tau\,;A,\boldsymbol{b}) = 1/\gamma(\tau) \text{ for all Butcher trees } \tau \text{ with } |\tau| \le p. \tag{6.1.32}$$

#### 6.1.4.4 Butcher's Simplifying Assumptions and Designing Runge–Kutta Methods

Butcher's condition (6.1.32) is a wonderful way of testing whether a given Runge–Kutta method has a given order of accuracy. However, it can be used as a *design* tool as well. Butcher gave *simplifying* assumptions that are sufficient, but not necessary, for obtaining a given order of accuracy. The conditions are still flexible enough to be used as the basis for creating new methods. In particular, these simplifying assumptions allow us to create high-order Runge–Kutta methods based on Gaussian quadrature methods (see Section 5.2).

The simplifying assumptions are:

$$B(p): \quad \sum_{i=1}^{s} b_i c_i^{k-1} = \frac{1}{k} \qquad \text{for } k = 1, 2, \ldots, p, \tag{6.1.33}$$

$$C(q): \quad \sum_{j=1}^{s} a_{ij} c_j^{k-1} = \frac{1}{k} c_i^k \qquad \text{for } k = 1, 2, \ldots, q, \quad i = 1, 2, \ldots, s, \tag{6.1.34}$$

$$D(r): \quad \sum_{i=1}^{s} b_i c_i^{k-1} a_{ij} = \frac{1}{k} b_j (1 - c_j^k) \qquad \text{for } k = 1, 2, \ldots, r, \quad j = 1, 2, \ldots, s. \tag{6.1.35}$$

The most important use of these is to show a given order for a method satisfying these conditions.

**Theorem 6.8** *If a Butcher tableau containing $A$, $\boldsymbol{b}$ and $\boldsymbol{c}$ with $c_i = \sum_{j=1}^{s} a_{ij}$ satisfies conditions $B(p)$, $C(q)$, and $D(r)$ with $p \le q + r + 1$ and $p \le 2q + 2$, then the method is of order $p$.*

**Table 6.1.2** Gauss methods

| | |
|---|---|
| $1/2$ | $1/2$ |
| | $1$ |

(A) 1 stage; order is two

| | | |
|---|---|---|
| $(3-\sqrt{3})/6$ | $1/4$ | $(3-2\sqrt{3})/12$ |
| $(3+\sqrt{3})/6$ | $(3+2\sqrt{3})/12$ | $1/4$ |
| | $1/2$ | $1/2$ |

(B) 2 stage; order is four

| | | | |
|---|---|---|---|
| $(5-\sqrt{15})/10$ | $5/36$ | $(2/9-\sqrt{15}/5)$ | $(5/36-\sqrt{15}/30)$ |
| $1/2$ | $(5/36+\sqrt{15}/24)$ | $2/9$ | $(5/36-\sqrt{15}/24)$ |
| $(5+\sqrt{15})/10$ | $(5/36+\sqrt{15}/30)$ | $(2/9+\sqrt{15}/5)$ | $5/36$ |
| | $5/18$ | $4/9$ | $5/18$ |

(C) 3 stage; order is six

The proof uses the manipulation of Butcher trees and the order conditions (6.1.32). Condition $B(p)$ is *necessary* to obtain a method of order $p$ as $B(p)$ is necessary to integrate $\boldsymbol{f}(t,\boldsymbol{x}) = \boldsymbol{f}(t)$ with order $p$ accuracy. Condition $C(q)$ ensures that the integration formula

$$\int_{t_m}^{t_m+c_i h} f(t)\,dt \approx h\sum_{j=1}^{s} a_{ij}\, f(t_m + c_j h)$$

is exact for all polynomials of degree $\leq q-1$. A proof of Theorem 6.8 is given in [45, Thm. 7]. Just as important for applications, Butcher showed that:

**Theorem 6.9** *Conditions $B(p+q)$ and $C(q)$ imply $D(p)$, provided the $c_j$'s are distinct.*

The simplifying assumptions can be used to create methods that have high order. For example, suppose we start with the Gauss–Legendre integration method: $\int_{t_k}^{t_{k+1}} f(t)\,dt \approx h\sum_{j=1}^{s} b_j\, f(t_k + c_j h)$ (see Section 5.2) that is exact for all polynomials of degree $\leq 2s$ where $s$ is the number of function evaluations. The values $c_j$ are the roots of the Legendre polynomial of degree $s$; that is, $(d/dx)^s\left[(x^2-1)^s\right] = 0$ at $x = c_j$. This choice of $c_j$'s and $b_j$'s satisfies $B(2s)$. To determine the coefficients $a_{ji}$ we need $s^2$ equations. Satisfying condition $C(s)$ gives those additional $s^2$ equations. By Theorem 6.9, condition $D(s)$ also holds. By Theorem 6.8, the resulting method has order $2s$. These are the *Gauss methods* or *Gauss–Kuntzmann methods*. The Gauss methods of one, two and three stages are shown in Table 6.1.2. Note that the one-stage Gauss method is the implicit mid-point rule.

**Table 6.1.3** Radau IIA methods

| | | |
|---|---|---|
| $1/3$ | $5/12$ | $-1/12$ |
| $1$ | $3/4$ | $1/4$ |
| | $3/4$ | $1/4$ |

(A) 2 stages; order is three

| | | | |
|---|---|---|---|
| $(4-\sqrt{6})/10$ | $(88-7\sqrt{6})/360$ | $(296-169\sqrt{6})/1800$ | $(-2+3\sqrt{6})/225$ |
| $(4+\sqrt{6})/10$ | $(296+169\sqrt{6})/1800$ | $(88+7\sqrt{6})/360$ | $(-2-3\sqrt{6})/225$ |
| $1$ | $(16-\sqrt{6})/36$ | $(16+\sqrt{6})/36$ | $1/9$ |
| | $(16-\sqrt{6})/36$ | $(16+\sqrt{6})/36$ | $1/9$ |

(B) 3 stages; order is five

Another family of methods that are important are the *Radau methods*, particularly the *Radau IIA* methods. The Radau methods use the $c_j$'s being the zeros of

$$\frac{d^{s-1}}{dx^{s-1}}\left[x^{s-1}(1-x)^s\right].$$

This gives an integration method of order $2s-1$, but we have $c_s = 1$. Radau methods satisfy simplifying assumptions $B(2s-1)$; the values of $a_{ji}$ are determined by satisfying $C(s)$ giving the necessary $s^2$ equations. Theorem 6.9 then implies $D(s-1)$ also holds, and so by Theorem 6.8, the $s$-stage Radau IIA method has order $2s-1$. The one-stage Radau IIA method is the implicit or backward Euler method. The Radau IIA methods of two and thee stages are shown in Table 6.1.3. Note that the last line of the tableau ($\boldsymbol{b}^T$) is the same as the second last line of the tableau ($\boldsymbol{e}_s^T A$). This is not a random fact; it is important for some properties of these methods, as is discussed in Section 6.1.6.

Another important family of methods are the *diagonally implicit Runge–Kutta methods* or *DIRK* methods. In DIRK methods, the matrix $A$ is lower triangular, but with non-zero entries on the diagonal. Typically, DIRK methods have the same non-zero value in each diagonal entry. These methods have the benefit, that when solving the Runge–Kutta equations (6.1.20) for $\boldsymbol{x}(t) \in \mathbb{R}^n$, a series of $s$ $n$-dimensional nonlinear equations can be solved in turn, while for a general implicit Runge–Kutta method, one $sn$-dimensional nonlinear system of equations must be solved. Even if these equations are linear (or we use the Newton method), we expect that solving one $n \times n$ system of equations takes about $1/s^3$ times as much computational work as solving one $(sn) \times (sn)$ system. So solving $s$ systems each $n \times n$ takes about $1/s^2$ times as much computational work. There are many DIRK methods, but here we highlight two of them, both three stage methods. The method of Alexander [3] gives third order accuracy, while the method of Crouzeix & Raviart [62] gives fourth order accuracy. These methods are shown in Table 6.1.4.

**Table 6.1.4** DIRK methods of Alexander and Crouzeix & Raviart

$$\begin{array}{c|ccc} \alpha & \alpha & & \\ \tau_2 & \tau_2-\alpha & \alpha & \\ 1 & b_1 & b_2 & \alpha \\ \hline & b_1 & b_2 & \alpha \end{array}$$

The values of the parameters are given by

$$\alpha = \text{root of } x^3 - 3x^2 + \tfrac{3}{2}x - \tfrac{1}{6} \text{ in } (\tfrac{1}{6},\ \tfrac{1}{2}),$$
$$\alpha \approx 0.43586652150846$$
$$\tau_2 = \tfrac{1}{2}(1+\alpha),$$
$$b_1 = -\tfrac{1}{4}(6\alpha^2 - 16\alpha + 1),$$
$$b_2 = +\tfrac{1}{4}(6\alpha^2 - 20\alpha + 5).$$

(A) Alexander's DIRK method

$$\begin{array}{c|ccc} \gamma & \gamma & & \\ 1/2 & 1/2-\gamma & \gamma & \\ 1-\gamma & 2\gamma & 1-4\gamma & \gamma \\ \hline & \delta & 1-2\delta & \delta \end{array}$$

The values of the parameters are given by

$$\gamma = \frac{1}{\sqrt{3}}\cos(\frac{\pi}{18}) + \frac{1}{2},$$
$$\delta = \frac{1}{6\,(2\gamma-1)^2}.$$

(B) Method of Crouzeix & Raviart

## 6.1.5 *Multistep Methods*

Multistep methods use previous solution values $\boldsymbol{x}_k,\ \boldsymbol{x}_{k-1},\ \ldots,\ \boldsymbol{x}_{k-m}$ and interpolation to compute $\boldsymbol{x}_{k+1}$. The earliest of these methods are the Adams methods named after John Couch Adams who published the methods as an appendix to a paper on capillary action of drops of water by Francis Bashforth [17] in 1883. These methods are split into explicit methods (the Adams–Bashforth methods) and implicit methods (the Adams–Moulton methods). The Adams–Moulton methods had the name "Moulton" attached because of Ray Forest Moulton's book [183] in which he showed that an explicit method can provide a starting value for the corresponding implicit method.

All Adams methods are based on interpolation of $\boldsymbol{f}(t, \boldsymbol{x}(t))$. The Adams–Bashforth methods use the polynomial interpolant $\boldsymbol{p}_{m,k}(t)$ of $\boldsymbol{f}(t_k, \boldsymbol{x}_k),\ \boldsymbol{f}(t_{k-1}, \boldsymbol{x}_{k-1}),\ \ldots,\ \boldsymbol{f}(t_{k-m}, \boldsymbol{x}_{k-m})$ to compute

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{p}_{m,k}(t)\,dt \approx \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{f}(t, \boldsymbol{x}(t))\,dt.$$

Assuming that the time instances $t_{k-j}$ are equally spaced: $t_{k-j} = t_k - j\,h$, we can write $\boldsymbol{p}_{m,k}$ as a linear combination of Lagrange interpolation polynomials (4.1.3)

$$\boldsymbol{p}_{m,k}(t) = \sum_{j=0}^{m} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})\, L_j(t) \qquad \text{where } \deg L_j = m \text{ and}$$

$$L_j(t_{k-\ell}) = \begin{cases} 1, & \text{if } j = \ell, \\ 0, & \text{if } j \neq \ell \text{ and } j = 0, 1, 2, \ldots, m. \end{cases}$$

We can write

$$L_j(t) = \widetilde{L}_j\left(\frac{t - t_k}{h}\right) \qquad \text{where } \deg \widetilde{L}_j = m \text{ and}$$

$$\widetilde{L}_j(-\ell) = \begin{cases} 1, & \text{if } j = \ell, \\ 0, & \text{if } j \neq \ell \text{ and } j = 0, 1, 2, \ldots, m. \end{cases}$$

These $\widetilde{L}_j$ do not depend on $h$. Then

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \boldsymbol{p}_{m,k}(t)\, dt \\ &= \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \sum_{j=0}^{m} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})\, L_j(t)\, dt \\ &= \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \sum_{j=0}^{m} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})\, \widetilde{L}_j\left(\frac{t - t_k}{h}\right) dt \\ &= \boldsymbol{x}_k + h \sum_{j=0}^{m} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j}) \int_0^1 \widetilde{L}_j(s)\, ds. \end{aligned} \tag{6.1.36}$$

The values of $\beta_j := \int_0^1 \widetilde{L}_j(s)\, ds$ for $m = 0, 1, 2, 3, 4$ are shown in Table 6.1.5.

The Adams–Moulton methods, on the other hand, are implicit, and use interpolation at $t = t_{k+1}, t_k, \ldots, t_{k-m+1}$:

$$\boldsymbol{q}_{m,k}(t) = \sum_{j=-1}^{m-1} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})\, M_j(t) \qquad \text{where } \deg M_j = m \text{ and}$$

$$M_j(t_{k-\ell}) = \begin{cases} 1, & \text{if } j = \ell \\ 0, & \text{if } j \neq \ell \text{ and } j = -1, 0, 1, \ldots, m-1. \end{cases} \tag{6.1.37}$$

This interpolation uses only one future value $\boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1})$, which directly involves the unknown $\boldsymbol{x}_{k+1}$ to be computed.

Assuming equally spaced interpolation points with spacing $h$, the Lagrange interpolation polynomials are $M_j(t) = \widetilde{M}_j((t - t_k)/h)$ where $\deg \widetilde{M}_j = m$ and

$$\widetilde{M}_j(-\ell) = \begin{cases} 1, & \text{if } j = \ell, \\ 0, & \text{if } j \neq \ell \text{ and } j = -1, 0, 1, \ldots, m-1. \end{cases}$$

Note that $\widetilde{M}_j$ does not depend on $h$. This leads to the Adams–Moulton method

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \int_{t_k}^{t_{k+1}} \sum_{j=-1}^{m-1} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})\, M_j(t)\, dt \\ &= \boldsymbol{x}_k + h \sum_{j=-1}^{m-1} \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j}) \int_0^1 \widetilde{M}_j(s)\, ds. \end{aligned}$$

Values of $\gamma_j := \int_0^1 \widetilde{M}_j(s)\, ds$ are also shown in Table 6.1.5.

The Adams–Bashforth method with $m = 0$ is Euler's method. The Adams–Moulton method with $m = 1$ is the implicit trapezoidal rule.

Another family of multistep methods are the *backward differentiation formulas* or BDF methods. These rely on an interpolant of the *solution* values: $\widetilde{\boldsymbol{p}}_{m,k}(t_{k-j}) = \boldsymbol{x}_{k-j}$ for $j = -1, 0, 1, \ldots, m-1$. We can then write

$$\widetilde{\boldsymbol{p}}_{m,k}(t) = \sum_{j=-1}^{m-1} \boldsymbol{x}_{k-j}\, M_j(t) \tag{6.1.38}$$

where the Lagrange interpolation polynomials $M_j$ are given by (6.1.37). The equation to be satisfied is then $\widetilde{\boldsymbol{p}}'_{m,k}(t_{k+1}) = \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1})$. This implicitly defines the equation for $\boldsymbol{x}_{k+1}$ given $\boldsymbol{x}_{k-j}$ for $j = 0, 1, \ldots, m-1$. That is,

$$\sum_{j=-1}^{m-1} \boldsymbol{x}_{k-j}\, M'_j(t_{k+1}) = \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}).$$

Writing $M_j(t) = \widetilde{M}_j((t - t_k)/h)$ we see that this method can be written as

$$\boldsymbol{x}_{k+1} + \sum_{j=0}^{m-1} \frac{\widetilde{M}'_j(1)}{\widetilde{M}'_{-1}(1)} \boldsymbol{x}_{k-j} = h\,\beta\, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}), \qquad \text{or equivalently,}$$

$$\boldsymbol{x}_{k+1} = \sum_{j=0}^{m-1} \alpha_j\, \boldsymbol{x}_{k-j} + h\,\beta\, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}). \tag{6.1.39}$$

Table 6.1.6 shows the values of $\beta$ and the coefficients $\alpha_j$ for different values of $m$.

Note that there are no usable BDF methods beyond these six. Using $m = 7$ or higher results in a method that is unstable, even for $\boldsymbol{f}(t, \boldsymbol{x}) = \boldsymbol{0}$ for all $(t, \boldsymbol{x})$. Every BDF method is implicit.

**Table 6.1.5** Adams–Bashforth and Adams–Moulton coefficients

| $\beta_j$ | $j$ | | | | | |
|---|---|---|---|---|---|---|
| $m$ | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 1 | | | | | |
| 1 | $\frac{3}{2}$ | $-\frac{1}{2}$ | | | | |
| 2 | $\frac{23}{12}$ | $-\frac{4}{3}$ | $\frac{5}{12}$ | | | |
| 3 | $\frac{55}{24}$ | $-\frac{59}{24}$ | $\frac{37}{24}$ | $-\frac{3}{8}$ | | |
| 4 | $\frac{1901}{720}$ | $-\frac{1387}{360}$ | $\frac{109}{30}$ | $-\frac{637}{360}$ | $\frac{251}{720}$ | |
| 5 | $\frac{4277}{1440}$ | $-\frac{2641}{480}$ | $\frac{4991}{720}$ | $-\frac{3649}{720}$ | $\frac{959}{480}$ | $-\frac{95}{288}$ |

(A) Adams-Bashforth

| $\gamma_j$ | $j$ | | | | | |
|---|---|---|---|---|---|---|
| $m$ | $-1$ | 0 | 1 | 2 | 3 | 4 |
| 0 | 1 | | | | | |
| 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | | | | |
| 2 | $\frac{5}{12}$ | $\frac{2}{3}$ | $-\frac{1}{12}$ | | | |
| 3 | $\frac{3}{8}$ | $\frac{19}{24}$ | $-\frac{5}{24}$ | $\frac{1}{24}$ | | |
| 4 | $\frac{251}{720}$ | $\frac{323}{360}$ | $-\frac{11}{30}$ | $\frac{53}{360}$ | $-\frac{19}{720}$ | |
| 5 | $\frac{95}{288}$ | $\frac{1427}{1440}$ | $-\frac{133}{240}$ | $\frac{241}{720}$ | $-\frac{173}{1440}$ | $\frac{3}{160}$ |

(B) Adams–Moulton

### *6.1.6 Stability and Implicit Methods*

Stability is an important concept in differential equations and their numerical solution. If we consider the basic differential equation

$$\frac{dx}{dt} = \lambda x, \qquad \text{the solution is}$$
$$x(t) = x(t_0)\, e^{\lambda(t-t_0)}.$$

**Table 6.1.6** BDF formulas

| | | $\alpha_j$ | | | | | |
|---|---|---|---|---|---|---|---|
| $m$ | $\beta$ | $j=0$ | $j=1$ | $j=2$ | $j=3$ | $j=4$ | $j=5$ |
| 1 | 1 | 1 | | | | | |
| 2 | $\frac{2}{3}$ | $\frac{4}{3}$ | $-\frac{1}{3}$ | | | | |
| 3 | $\frac{6}{11}$ | $\frac{18}{11}$ | $-\frac{9}{11}$ | $+\frac{2}{11}$ | | | |
| 4 | $\frac{12}{25}$ | $\frac{48}{25}$ | $-\frac{36}{25}$ | $+\frac{16}{25}$ | $-\frac{3}{25}$ | | |
| 5 | $\frac{60}{137}$ | $\frac{300}{137}$ | $-\frac{300}{137}$ | $+\frac{200}{137}$ | $-\frac{75}{137}$ | $+\frac{12}{137}$ | |
| 6 | $\frac{60}{147}$ | $\frac{360}{147}$ | $-\frac{450}{147}$ | $+\frac{400}{147}$ | $-\frac{225}{147}$ | $+\frac{72}{147}$ | $-\frac{10}{147}$ |

The solution $x(t) \to 0$ as $t \to +\infty$ if and only if the real part of $\lambda = \operatorname{Re}\lambda < 0$; $|x(t)| \to \infty$ as $t \to \infty$ if and only if $\operatorname{Re}\lambda > 0$. If $\operatorname{Re}\lambda = 0$, then $|x(t)|$ is constant.

We would like the stability of our numerical methods to match, as well as reasonably possible, the stability of the differential equation being solved. If the step size $h$ is small, then because of the convergence of the method we expect to have the behavior of the method's results closely matching the behavior of the exact solution. The difficulty becomes acute when the value of $L\,h$ is large where $L$ is the Lipschitz constant of $\boldsymbol{f}(t, \boldsymbol{x})$, but $h$ is modestly small. Normally we might expect that the solution would show "interesting behavior" over a time-scale of $1/L$, so it would be natural to make $L\,h$ small in order to capture this "interesting behavior".

But there are many systems of differential equations where we might indeed want to make $L\,h$ large. Consider, for example, the diffusion equation

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + f(t, x, y, u(t, x, y)) \qquad \text{inside a region } \Omega, \tag{6.1.40}$$

$$u = 0 \qquad \text{on the boundary } \partial\Omega. \tag{6.1.41}$$

This is a non-linear partial differential equation. Using the five-point stencil approximation (2.4.7) for $\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2$ we get the discrete version of the diffusion equation:

$$\frac{du_{ij}}{dt} = \frac{u_{i+1,j} + u_{i,j+1} - 4u_{ij} + u_{i-1,j} + u_{i,j-1}}{(\Delta x)^2} + f(t, x_i, y_j, u_{ij}) \qquad \text{if } (x_i, y_j) \in \Omega, \tag{6.1.42}$$

$$u_{ij} = 0 \qquad \text{if } (x_i, y_j) \notin \Omega. \tag{6.1.43}$$

By combining all the values $u_{ij}$ for $(x_i, y_j) \in \Omega$ into a single vector $\boldsymbol{u}$, we can write this as a single differential equation

$$\frac{d\boldsymbol{u}}{dt} = B_{\Delta x}\boldsymbol{u} + \boldsymbol{f}(t, \boldsymbol{u}).$$

The Lipschitz constant of the right-hand side is bounded by $\|B_{\Delta x}\| + L_{\boldsymbol{f}}$ where $L_{\boldsymbol{f}}$ is the Lipschitz constant of $\boldsymbol{f}$. Since $\|B_{\Delta x}\|_2 \approx 8/(\Delta x)^2$ and $L_{\boldsymbol{f}}$ is usually modest, for small $\Delta x$ the Lipschitz constant of the right-hand side is dominated by $\|B_{\Delta x}\|_2$. The good news is that the eigenvalues of $B_{\Delta x}$ are negative real. This makes the differential equation (6.1.42, 6.1.43) quite stable. But the numerical method has to deal with the large Lipschitz constant. The large negative eigenvalues of $B_{\Delta x}$ correspond to rapid spatial oscillation that is quickly damped out in time. But these large eigenvalues can cause difficulties for our numerical methods. Take, for example, Euler's method with $\boldsymbol{f}(t, \boldsymbol{u}) = \boldsymbol{0}$:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^k + h\, B_{\Delta x}\boldsymbol{u}^k = (I + h\, B_{\Delta x})\boldsymbol{u}^k.$$

The eigenvalues of $I + h\, B_{\Delta x}$ are $1 + h\lambda$ where $\lambda$ is an eigenvalue of $B_{\Delta x}$. If $h\lambda < -2$ then $|1 + h\lambda| > 1$ meaning that the method has become unstable. Since the minimum eigenvalue is $\approx -8/(\Delta x)^2$, then the time step $h$ should be in the range $0 < h \lesssim (\Delta x)^2/4$.

On the other hand, if we use the implicit Euler method, we get

$$\begin{aligned} \boldsymbol{u}^{k+1} &= \boldsymbol{u}^k + h\, B_{\Delta x}\boldsymbol{u}^{k+1} \qquad \text{so} \\ \boldsymbol{u}^{k+1} &= (I - h\, B_{\Delta x})^{-1}\boldsymbol{u}^k. \end{aligned}$$

The eigenvalues of $(I - h\, B_{\Delta x})^{-1}$ are $1/(1 - h\lambda)$ where $\lambda$ is an eigenvalue of $B_{\Delta x}$. Since the eigenvalues of $B_{\Delta x}$ are negative, $0 < 1/(1 - h\lambda) < 1$. This method (implicit Euler) is *absolutely stable*; that is, provided $\operatorname{Re}\lambda < 0$, the method applied to $dx/dt = \lambda x$ is stable.

Consider the test equation $dx/dt = \lambda x$; then $x(t_{k+1}) = e^{h\lambda}x(t_k)$. For a numerical method we wish to find $x_{k+1} \approx x(t_{k+1})$ in terms of $x_k \approx x(t_k)$. Provided the method is linear, we can write $x_{k+1} = R(h, \lambda)x_k$. The function $R(h, \lambda)$ is called the *stability function*; for all the methods we will consider, $R(h, \lambda) = R(h\lambda)$.

#### 6.1.6.1 Stability of Runge–Kutta Methods

To find the stability function for (6.1.20, 6.1.21) we substitute $f(t, x) = \lambda x$ to get

$$\begin{aligned} v_{k,j} &= \lambda\left[x_k + h\sum_{i=1}^{s} a_{ji}v_i\right], \qquad j = 1, 2, \ldots, s, \\ x_{k+1} &= x_k + h\sum_{j=1}^{s} b_j v_{k,j}. \end{aligned}$$

**Table 6.1.7** Stability functions for selected Runge–Kutta methods

| Method | $R(z)$ | Method | $R(z)$ |
|---|---|---|---|
| Euler | $1+z$ | 4-stage Runge–Kutta | $1+z+\frac{1}{2}z^2+\frac{1}{6}z^3+\frac{1}{24}z^4$ |
| Implicit Euler | $\dfrac{1}{1-z}$ | Gauss 2-stage | $\dfrac{1+\frac{1}{2}z+\frac{1}{12}z^2}{1-\frac{1}{2}z+\frac{1}{12}z^2}$ |
| Heun | $1+z+\frac{1}{2}z^2$ | Radau IIA 2-stage | $\dfrac{1+\frac{1}{3}z}{1-\frac{2}{3}z+\frac{1}{6}z^2}$ |
| Trapezoidal | $\dfrac{1+\frac{1}{2}z}{1-\frac{1}{2}z}$ | Alexander | $\dfrac{1+(1-3\alpha)z+\frac{1}{2}(6\alpha^2-6\alpha+1)z^2}{(1-\alpha z)^3}$ |
| Mid-point | $\dfrac{1+\frac{1}{2}z}{1-\frac{1}{2}z}$ | Crouzeix & Raviart | $\dfrac{\left(\begin{array}{c} z^3(-\frac{1}{2}\gamma^3+2\gamma^2-\frac{3}{2}\gamma+\frac{1}{6})+ \\ +z^2(3\gamma^2-3\gamma+\frac{1}{2})+z(1-3\gamma)+1 \end{array}\right)}{(1-\gamma z)^3}$ |

Expressing these equations in matrix—vector form we get

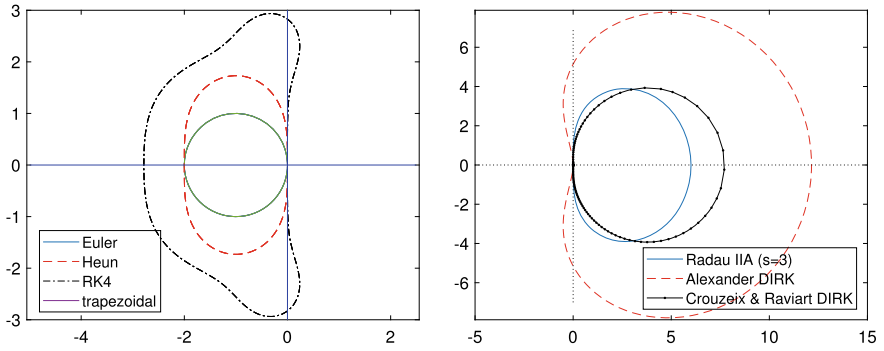
$$\begin{aligned} \boldsymbol{v}_k &= \lambda \boldsymbol{e}\, x_k + \lambda h\, A \boldsymbol{v}_k, \\ x_{k+1} &= x_k + h \boldsymbol{b}^T \boldsymbol{v}_k, \end{aligned}$$

where $\boldsymbol{e} = [1, 1, \ldots, 1]^T$. Solving these gives $(I - h\lambda A)\boldsymbol{v}_k = \lambda\, \boldsymbol{e}\, x_k$ and so

$$\begin{aligned} x_{k+1} &= x_k + h\, \boldsymbol{b}^T (I - h\lambda A)^{-1} \lambda\, \boldsymbol{e}\, x_k \\ &= \left[1 + h\lambda\, \boldsymbol{b}^T (I - h\lambda A)^{-1} \boldsymbol{e}\right] x_k \\ &= R(h\lambda)\, x_k \qquad \text{where} \\ R(h\lambda) &= 1 + h\lambda\, \boldsymbol{b}^T (I - h\lambda A)^{-1} \boldsymbol{e}. \end{aligned}$$

Stability functions $R(z)$ for $z = h\lambda$ for various methods are listed in Table 6.1.7. It should be noted that for a method of order $p$, $R(z) = e^z + \mathcal{O}(z^{p+1})$ as $|z| \to 0$. If a method is explicit, then $R(z)$ is a polynomial with the degree equal to the number of stages $s$; in general, $R(z)$ is a rational function of $z$. If $A$ is an invertible matrix, then $R(z) \to 1 - \boldsymbol{b}^T A^{-1} \boldsymbol{e}$ as $|z| \to \infty$. In this case, and $1 - \boldsymbol{b}^T A^{-1} \boldsymbol{e} \neq 0$, then the degree of the numerator and denominators of $R(z)$ are the same. Otherwise, if $1 - \boldsymbol{b}^T A^{-1} \boldsymbol{e} = 0$, the degree of the numerator of $R(z)$ is less than the degree of the denominator of $R(z)$.

The stability function $R(z)$ can often be related to *Padé approximations* (see Exercise 4.6.10). A Padé approximation of a function $f$ is a rational function $r(z) = n(z)/d(z)$ where $f^{(k)}(0) = r^{(k)}(0)$ for $k = 0, 1, 2, \ldots, \deg n + \deg d + 1$. The order of the approximation is $m = \deg n + \deg d + 1$. We call $r(z)$ a $(\deg n,\ \deg d)$ Padé approximation. Then $R(z)$ for the trapezoidal and mid-point rules are both the $(1, 1)$ Padé approximations to $e^z$, while $R(z)$ for the 2-stage Gauss method is a $(2, 2)$




(A) Explicit and trapezoidal methods (B) Radau IIA ($s = 2$) and DIRK methods

**Fig. 6.1.5** Stability regions of Runge–Kutta methods. In (a) the stability regions are inside the curves; in (b) they are outside the curves

Padé approximation to $e^z$, while $R(z)$ for the Radau IIA method is a $(1, 3)$ Padé approximation to $e^z$. In fact, for linear equations $d\boldsymbol{x}/dt = B\boldsymbol{x}$, we have $\boldsymbol{x}_{k+1} = R(hB)\boldsymbol{x}_k$, so finding a suitable method can be reduced to the problem of finding a Padé approximation $R(z) \approx e^z$ for $z \approx 0$ with suitable stability properties [28].

The *stability region* is the set $\{ z \in \mathbb{C} : |R(z)| < 1 \}$. Figure 6.1.5 shows the stability regions for the Euler, Heun, standard fourth-order Runge–Kutta, trapezoidal, 2-stage Radau IIA methods as well the DIRK methods of Alexander, and of Crouzeix & Raviart. Note that the stability regions of the trapezoidal, mid-point, and 2-stage Gauss methods are all exactly the left-half complex plane.

There are several stability conditions that can be easily checked from the stability conditions: a method is called *A-stable* if the stability region includes all the left-half complex plane; that is, $|R(z)| < 1$ for all $z \in \mathbb{C}$ where $\operatorname{Re} z < 0$. This is a stronger condition than is needed to ensure the method is stable for the diffusion equation. Even stronger is the condition of being *L-stable*: the method must be A-stable and $\lim_{|z|\to\infty} R(z) = 0$. Gauss methods, Radau IIA methods, and the DIRK methods of Alexander, and of Crouzeix & Raviart, are all A-stable. Of these, the Radau IIA and the DIRK methods are L-stable.

There is a stability condition for nonlinear equations called *B-stability*. A method is B-stable if for the differential equations

$$d\boldsymbol{x}/dt = \boldsymbol{f}(t, \boldsymbol{x}),$$
$$d\boldsymbol{y}/dt = \boldsymbol{f}(t, \boldsymbol{y}),$$

with

$$(\boldsymbol{f}(t, \boldsymbol{u}) - \boldsymbol{f}(t, \boldsymbol{v}))^T (\boldsymbol{u} - \boldsymbol{v}) \le 0 \tag{6.1.44}$$

for all $\boldsymbol{u}$ and $\boldsymbol{v}$ and $t$ implies that the results of applying the given method results in

$$\left\|\boldsymbol{x}_{n+1} - \boldsymbol{y}_{n+1}\right\|_2 \le \left\|\boldsymbol{x}_n - \boldsymbol{y}_n\right\|_2 .$$

To see the relevance of this condition, the exact solutions satisfy

$$\frac{d}{dt}\left(\|\boldsymbol{x}(t) - \boldsymbol{y}(t)\|_2^2\right) = 2(\boldsymbol{f}(t, \boldsymbol{x}(t)) - \boldsymbol{f}(t, \boldsymbol{y}(t)))^T(\boldsymbol{x}(t) - \boldsymbol{y}(t)) \le 0$$

so $\|\boldsymbol{x}(t+r) - \boldsymbol{y}(t+r)\|_2 \le \|\boldsymbol{x}(t) - \boldsymbol{y}(t)\|_2$ for any $r > 0$. This may appear to be a difficult condition to check as it appears to require checking all functions satisfying (6.1.44). However, both Burrage & Butcher [41] and Crouzeix [63] found a way of identifying B-stable Runge–Kutta methods: a Runge–Kutta method is B-stable if and only if $b_i > 0$ for all $i$, and

$$M := B\,A + A^T B^T - \boldsymbol{b}\boldsymbol{b}^T \qquad \text{where } B = \operatorname{diag}(b_1, \ldots, b_s), \tag{6.1.45}$$

is a positive semi-definite matrix. Note that B-stability implies A-stability. The Gauss and Radau IIA methods are B-stable methods, but the above DIRK methods are not.

There is a problem that arises with Runge–Kutta methods that does not occur with multistep methods called *order reduction* for stiff equations [209]. The order of a method is achieved for sufficiently small $h$, usually when $L\,h \ll 1$ where $L$ is the Lipschitz constant. For stiff but stable differential equations, we might have $h > 0$ small, but $L\,h$ is not. It is possible that in the range $1/L \ll h \ll 1$, the error in not behaving like $C\,h^p$ where $p$ is the asymptotic order of the method. To illustrate this phenomenon, consider the initial value problem

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{g}'(t) - D\left[\boldsymbol{x} - \boldsymbol{g}(t)\right], \qquad \boldsymbol{x}(0) = \boldsymbol{g}(0) \tag{6.1.46}$$

where $D$ is a large diagonal matrix with at least some large diagonal entries, and $\boldsymbol{g}(t)$ a slowly varying function. The solution to (6.1.46) is $\boldsymbol{x}(t) = \boldsymbol{g}(t)$. This is a slowly varying function, and so we should be able to get accurate solutions even though the entries in $D$ may be large. Consider the modified stability equation

$$\frac{dx}{dt} = g'(t) + \lambda\,[x - g(t)] \qquad \text{with } \operatorname{Re}\lambda < 0.$$

The exact solution with $x(t_0) = g(t_0)$ is $x(t) = g(t)$ for all $t$.

To analyze the numerical method applied to this equation, we introduce some extra quantities that connect the method to $g$:

$$\Delta_{k,j} = g(t_k + c_j h) - g(t_k) - h\sum_{i=1}^{s} a_{ji}\,g'(t_k + c_i h), \qquad j = 1, 2, \ldots, s,$$

$$\widehat{\Delta}_k = g(t_k + \quad h) - g(t_k) - h\sum_{j=1}^{s} b_j\,g'(t_k + c_j h).$$

We combine the components $\Delta_{k,i}$ into a vector $\boldsymbol{\Delta}_k$. If $\boldsymbol{\Delta}_k = \mathcal{O}(h^{q+1})$ we say that the method has *stage order* $q$, whereas if $\widehat{\Delta}_k = \mathcal{O}(h^{p+1})$ we say that the method has *quadrature order* $p$. Note that these are equivalent to Butcher's simplifying assumptions $C(q)$ and $B(p)$ respectively. After some calculations, we obtain the formula

$$x_{k+1} - g(t_{k+1}) = R(h\lambda)\ [x_k - g(t_k)] - h\lambda \boldsymbol{b}^T (I - h\lambda\, A)^{-1} \boldsymbol{\Delta}_k - \widehat{\Delta}_k.$$

The $s$-stage Gauss method satisfies the simplifying assumptions $B(2s)$ and $C(s)$, so that the stage order is $s$ while the quadrature order is $2s$. The Radau IIA methods satisfy $B(2s-1)$ and $C(s)$ If $1/|\lambda| \ll h \ll 1$ we have

$$\begin{aligned} -h\lambda \boldsymbol{b}^T (I - h\lambda\, A)^{-1} \boldsymbol{\Delta}_k &= -\boldsymbol{b}^T ((h\lambda)^{-1} I - A)^{-1} \boldsymbol{\Delta}_k \\ &\approx \boldsymbol{b}^T A^{-1} \boldsymbol{\Delta}_k \end{aligned}$$

for $A$ invertible. If the method is stiffly accurate in the sense that $\boldsymbol{b}^T = \boldsymbol{e}_s^T A$ (that is, $\boldsymbol{b}$ is the last row of $A$), then for $h\lambda$ large we get

$$\begin{gathered} -h\lambda \boldsymbol{b}^T (I - h\lambda\, A)^{-1} \boldsymbol{\Delta}_k \approx \boldsymbol{e}_s^T\, \boldsymbol{\Delta}_k = \Delta_{k,s} = \widehat{\Delta}_k, \qquad \text{so} \\ x_{k+1} - g(t_{k+1}) \approx R(h\lambda)\ [x_k - g(t_k)] \end{gathered}$$

to high order. We still need $|R(h\lambda)| \le 1$ for stability, of course. If $0 < h \ll 1$ but $h\lambda$ is not small, then

$$\begin{aligned} x_{k+1} - g(t_{k+1}) &= R(h\lambda)\ [x_k - g(t_k)] + \Delta_{k,s} + \frac{1}{h\lambda} \boldsymbol{b}^T \left(\frac{1}{h\lambda} I - A\right)^{-1} \boldsymbol{\Delta}_k - \widehat{\Delta}_k \\ &= R(h\lambda)\ [x_k - g(t_k)] + \frac{1}{h\lambda} \boldsymbol{b}^T \left(\frac{1}{h\lambda} I - A\right)^{-1} \boldsymbol{\Delta}_k \\ &= R(h\lambda)\ [x_k - g(t_k)] + \mathcal{O}(h^{q+1}/\,|h\lambda|). \end{aligned}$$

If $R(h\lambda) \ne 1$ for $h\lambda$ not small, even if $|R(h\lambda)| = 1$, then there is no accumulation of error over many steps: $\boldsymbol{\Delta}_k$ varies only slowly with $k$ so

$$\sum_{j=0}^{k} R(h\lambda)^j (h\lambda)^{-1} ((h\lambda)^{-1} I - A)^{-1} \boldsymbol{\Delta}_{k-j} = \mathcal{O}(h^{q+1}/\,|h\lambda|),$$

independently of $k$. While this may still be worse than $\mathcal{O}(h^p)$, this shows that there is benefit to be had from using stiffly accurate methods such as the Radau IIA and Alexander DIRK methods over the Gauss and Crouzeix & Raviart DIRK methods respectively.

### 6.1.6.2 Stability of Multistep Methods

The stability theory of multistep methods takes a slightly different form because of the different structure of these methods. We will assume a form that encompasses both the Adams and BDF methods:

$$\boldsymbol{x}_{k+1} = \sum_{j=0}^{p} \alpha_j \boldsymbol{x}_{k-j} + h \sum_{j=-1}^{p} \beta_j \, \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j}). \tag{6.1.47}$$

Applied to the differential equation $dx/dt = \lambda x$ we get the linear recurrence

$$(1 - h\lambda\beta_{-1})x_{k+1} = \sum_{k=0}^{p} \left(\alpha_j + h\lambda\,\beta_j\right) x_{k-j}.$$

The stability of this recurrence depends on the roots of the characteristic equation

$$(1 - h\lambda\beta_{-1})r^{p+1} - \sum_{j=0}^{p} \left(\alpha_j + h\lambda\,\beta_j\right) r^{p-j} = 0. \tag{6.1.48}$$

If we write $\alpha(r) = r^{p+1} - \sum_{j=0}^{p} \alpha_j r^{p-j}$ and $\beta(r) = \sum_{j=-1}^{p} \beta_j r^{p-j}$ then we can write the characteristic equation more succinctly as

$$\alpha(r) - h\lambda\,\beta(r) = 0. \tag{6.1.49}$$

One difference with the case of Runge–Kutta methods is that the case of $\lambda = 0$ can still be a problem. This means that applying the method to the differential equation $d\boldsymbol{x}/dt = \boldsymbol{0}$ is unstable. Such methods do not converge, as errors are amplified by a significant amount *with each step*.

More specifically, setting $\lambda = 0$ gives the characteristic equation $\alpha(r) = 0$. The fundamental stability conditions that are *necessary* for convergence of the method are that

$$\begin{gathered}\text{every root of } \alpha(r) = 0 \text{ has } |r| \le 1 \text{ and} \\ \text{every root } r \text{ with } |r| = 1 \text{ has multiplicity one.}\end{gathered} \tag{6.1.50}$$

To see why this is necessary, consider the recurrence $x_{k+1} = \sum_{j=0}^{p} \alpha_j x_j$. If the *distinct* roots of $\alpha(r) = 0$ are $r_1, \ldots, r_m$ with multiplicities $\nu_1, \ldots, \nu_m$ respectively, then the general solution of the recurrence is

$$x_k = \sum_{i=1}^{m} q_i(k)\, r_i^k, \qquad \deg q_i \le \nu_i - 1 \text{ for all } i, \tag{6.1.51}$$

where each $q_i$ is a polynomial. If $|r_i| > 1$ for any $i$, then solutions can grow exponentially fast *in the number of steps*, and not the time $t_k - t_0$. Even if $|r_i| = 1$, then the solutions can grow polynomially fast *in the number of steps*, if the multiplicity $\nu_i > 1$. This can result in errors growing like $\mathcal{O}(1/h^{\nu_i - 1})$ as $h \downarrow 0$ in this case.

The Adams methods (6.1.36) have the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h \sum_{j=-1}^{p} \beta_j \, \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j})$$

so for the Adams methods, $\alpha(r) = r^{p+1} - r^p$. The roots are one, which is a simple root, and zero which has multiplicity $p$. The Adams methods clearly satisfy the basic stability condition (6.1.50). BDF methods, on the other hand, have the form

$$\boldsymbol{x}_{k+1} = \sum_{j=0}^{p} \alpha_j \boldsymbol{x}_j + h \, \beta_{-1} \, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}).$$

It is far from being a foregone conclusion that BDF methods satisfy (6.1.50). The BDF method for $p = 1$ is

$$\boldsymbol{x}_{k+1} = \frac{4}{3}\boldsymbol{x}_k - \frac{1}{3}\boldsymbol{x}_{k-1} + h \, \frac{2}{3} \, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1});$$

for this method $\alpha(r) = r^2 - \frac{4}{3}r + \frac{1}{3}$ which has roots $r = (\frac{4}{3} \pm \sqrt{(\frac{4}{3})^2 - 4 \times 1 \times \frac{1}{3}})/2 = \frac{2}{3} \pm \frac{1}{3}$ of which one root is equal to one and the other is $1/3$, and so satisfies the basic stability condition (6.1.50). Every BDF method for $p = 0, 1, 2, \ldots, 5$ satisfies (6.1.50), but the BDF method for $p = 6$ fails (6.1.50). Larger values of $p$ also fail (6.1.50). This is why there are only six BDF methods.

There is also a consistency condition that we need: if $dx/dt = 0$ and $x(0) = 1$, the solution should be constant. If we want this to hold for our method, then we need

$$1 = \sum_{j=0}^{p} \alpha_j. \tag{6.1.52}$$

To obtain a consistency order of $m \geq 1$ we need

$$\sum_{j=0}^{p} \alpha_j (-j)^q + q \sum_{j=-1}^{p} \beta_j (-j)^{q-1} = 1 \qquad \text{for } q = 1, 2, \ldots, m. \tag{6.1.53}$$

In our derivations of the Adams and BDF methods, we did not need to establish the consistency conditions (6.1.52, 6.1.53) directly. Rather we used the accuracy properties of polynomial interpolation to provide the consistency properties directly.

The combination of stability (6.1.50) and consistency conditions (6.1.52, 6.1.53) imply convergence of the method. To see this, we note that the consistency conditions imply that the local truncation error is

$$\boldsymbol{\tau}_k = \boldsymbol{x}(t_{k+1}) - \sum_{j=0}^{p} \alpha_j \, \boldsymbol{x}(t_{k-j}) - h \sum_{j=-1}^{p} \beta_j \, \boldsymbol{x}'(t_{k-j}) = \mathcal{O}(h^{m+1}),$$

by expanding $\boldsymbol{x}(t_k + s)$ using Taylor series with remainder of order $m + 1$ in $s$. We can then write

$$\begin{aligned} \boldsymbol{x}(t_{k+1}) &= \sum_{j=0}^{p} \alpha_j \, \boldsymbol{x}(t_{k-j}) + h \sum_{j=-1}^{p} \beta_j \, \boldsymbol{f}(t_{k-j}, \boldsymbol{x}(t_{k-j})) + \boldsymbol{\tau}_k \\ \boldsymbol{x}_{k+1} &= \sum_{j=0}^{p} \alpha_j \, \boldsymbol{x}_{k-j} + h \sum_{j=-1}^{p} \beta_j \, \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j}). \end{aligned}$$

Subtracting and setting the error $\boldsymbol{e}_j = \boldsymbol{x}(t_j) - \boldsymbol{x}_j$ we get

$$\boldsymbol{e}_{k+1} = \sum_{j=0}^{p} \alpha_j \, \boldsymbol{e}_{k-j} + h \sum_{j=-1}^{p} \beta_j \left[ \boldsymbol{f}(t_{k-j}, \boldsymbol{x}(t_{k-j})) - \boldsymbol{f}(t_{k-j}, \boldsymbol{x}_{k-j}) \right] + \boldsymbol{\tau}_k.$$

Assuming that $\boldsymbol{f}(t, \boldsymbol{x})$ is Lipschitz in $\boldsymbol{x}$ with Lipschitz constant $L$, we get

$$\begin{aligned} \boldsymbol{e}_{k+1} &= \sum_{j=0}^{p} \alpha_j \, \boldsymbol{e}_{k-j} + \boldsymbol{\eta}_k, \qquad \text{with} \\ \left\| \boldsymbol{\eta}_k \right\| &\le h \, L \sum_{j=-1}^{p} \left| \beta_j \right| \left\| \boldsymbol{e}_{k-j} \right\| + \mathcal{O}(h^{m+1}). \end{aligned}$$

To find the solution we first find the solution to the linear recurrence

$$\begin{aligned} \theta_{k+1} &= \sum_{j=0}^{p} \alpha_j \, \theta_{k-j} + \begin{cases} 1, & \text{if } k = 0, \\ 0, & \text{otherwise,} \end{cases} \qquad \text{where} \\ \theta_k &= 0 \qquad \text{for } k \le 0. \end{aligned}$$

From the basic stability condition (6.1.50), there is a constant $M$ where $|\theta_k| \le M$ for all $k$. We can use a discrete convolution to write $\boldsymbol{e}_k$ in terms of the $\boldsymbol{\eta}_j$ plus a bounded term that comes from the starting error values $\boldsymbol{e}_0, \, \boldsymbol{e}_1, \, \ldots, \, \boldsymbol{e}_{p-1}$. Assuming that $\boldsymbol{e}_0 = \boldsymbol{e}_1 = \cdots = \boldsymbol{e}_{p-1} = \boldsymbol{0}$,

$$\boldsymbol{e}_k = \sum_{j=p}^{k-1} \theta_{k-j} \boldsymbol{\eta}_j .$$

Then we can obtain bounds

$$\begin{aligned} \|\boldsymbol{e}_{k+1}\| &\le \sum_{j=p}^{k} \left|\theta_{k+1-j}\right| \left\|\boldsymbol{\eta}_j\right\| \\ &\le \sum_{j=p}^{k} M \left( h\, L \sum_{\ell=-1}^{p} |\beta_\ell| \left\|\boldsymbol{e}_{j-\ell}\right\| + \mathcal{O}(h^{m+1}) \right). \end{aligned}$$

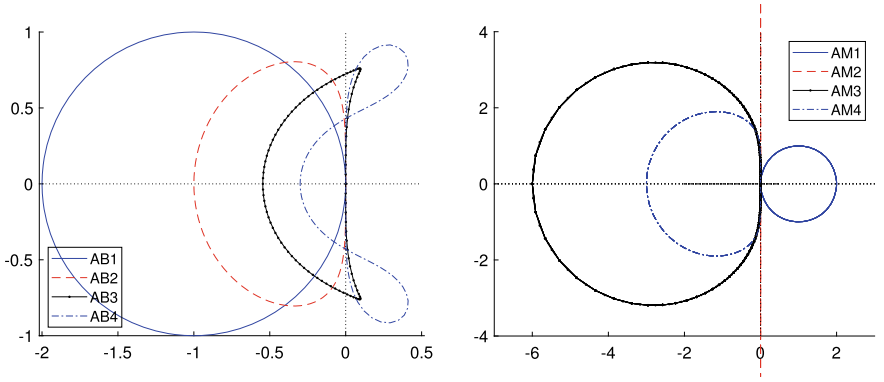
If $\psi_k = \max_{j \le k} \left\|\boldsymbol{e}_j\right\|$ then $\ell \le k$ implies $\psi_\ell \le \psi_k$, so

$$\begin{aligned} \psi_{k+1} &\le \sum_{j=p}^{k} M \left( h\, L \sum_{\ell=-1}^{p} |\beta_\ell|\, \psi_{j-\ell} + \mathcal{O}(h^{m+1}) \right) \\ &\le \sum_{j=p}^{k} M \left( h\, L\, \|\boldsymbol{\beta}\|_1\, \psi_{j+1} + \mathcal{O}(h^{m+1}) \right) \qquad \text{and} \\ \psi_{k+1} &\le \frac{1}{1 - M\, L\, h\, \|\boldsymbol{\beta}\|_1} \sum_{j=p}^{k-1} M \left( h\, L\, \|\boldsymbol{\beta}\|_1\, \psi_{j+1} + \mathcal{O}(h^{m+1}) \right). \end{aligned}$$

This gives a bound on $\psi_k$ of the form $\exp(c(t_k - t_0))\, \mathcal{O}(h^m)$ for a suitable constant $c$. This implies that $\|\boldsymbol{e}_k\| \le \exp(c(t_k - t_0))\, \mathcal{O}(h^m)$.

Stability regions for multistep methods as for Runge–Kutta methods are defined as the set of $h\lambda$ values in the complex plane for which the multistep method applied to $dx/dt = \lambda x$ gives a stable recurrence. This amounts to showing that all roots $r$ of the characteristic equation $\alpha(r) - h\lambda\,\beta(r) = 0$ (6.1.49) lie within the unit circle: $|r| < 1$. The boundaries of these regions are given by $h\lambda = \alpha(r)/\beta(r)$ for $|r| = 1$. The stability regions for the Adams methods are shown in Figure 6.1.6. Note that the lobes of the AB4 method protruding into the right-half plane should be excluded from the stability region; otherwise the interior of the curves shown is the stability region. The stability regions for the BDF methods are shown in Figure 6.1.7; the regions to the left of, and outside, the curves shown are the stability regions.

It should be noted that of these multistep methods, only implicit Euler (AM1 and BDF1), the implicit trapezoidal rule (AM2), and the second-order BDF method (BDF2) are A-stable; that is their stability region includes the entire left-half complex plane. The third-order BDF method is close to being A-stable, but its stability region misses a small part close to the imaginary axis. No multistep method with order higher than two is A-stable, as was shown by Dahlquist [67]; this is an example of an *order barrier*. The excellent stability properties of the BDF methods make




(A) Adams–Bashforth methods (B) Adams–Moulton methods

**Fig. 6.1.6** Stability regions for Adams methods

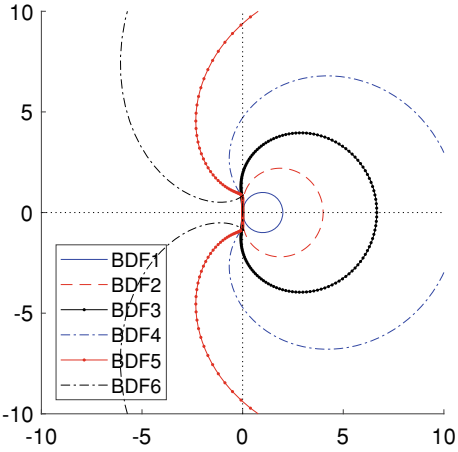



**Fig. 6.1.7** Stability regions for BDF methods

them useful methods for many applications, such as *differential algebraic equations* (DAEs) as well as partial differential equations in time.

### *6.1.7 Practical Aspects of Implicit Methods*

Implicit methods require solving a system of equations in general, and in general, these equations are non-linear. As we have seen in Chapter 3, there are a number of ways of solving systems of non-linear equations, such as fixed-point iteration and variants of Newton's method. Using Newton's method involves solving systems of

linear equations, and there is much about how to do this in Chapter 2, whether using direct or iterative methods. Knowing something of the structure of the equations can help design good methods. Ultimately, the methods chosen will depend on the system of differential equations being solved, and so the designer of the software for implicit methods needs to allow the user some flexibility as to how to do this.

Most of the implicit methods we have considered lead to the problem of finding $\boldsymbol{x}$ satisfying

$$\boldsymbol{x} = \boldsymbol{u} + h\,\beta\,\boldsymbol{f}(t, \boldsymbol{w} + \alpha\boldsymbol{x}) \tag{6.1.54}$$

given $\boldsymbol{u}$, $\boldsymbol{w}$, $\alpha$, $\beta$, and of course $\boldsymbol{f}$. Exceptions to this rule are the Runge–Kutta methods where the equations (6.1.20) are fully implicit. DIRK methods (where the $A$ matrix of the Butcher tableau is lower triangular) can be decomposed into $s$ equations like (6.1.54) to be solved sequentially. Even fully implicit Runge–Kutta methods can be streamlined to avoid the full complexity of the general Runge–Kutta system.

Fixed-point iteration can be applied to (6.1.54):

$$\boldsymbol{x}^{(m+1)} = \boldsymbol{u} + h\,\beta\,\boldsymbol{f}(t, \boldsymbol{w} + \alpha\boldsymbol{x}^{(m)}), \qquad m = 0, 1, 2, \ldots \tag{6.1.55}$$

This will converge provided $h\,|\alpha\,\beta|\;L < 1$ where $L$ is the Lipschitz constant of $\boldsymbol{f}$. Unfortunately, this strategy will likely fail when applied to stiff differential equations with $L$ large and $h$ small, but $L\,h > 1/\,|\alpha\,\beta|$. Often in practical situations we can write $\boldsymbol{f}(t, z) = Az + \boldsymbol{g}(t, z)$ where $A$ is a large matrix both in terms of its norm and in terms of the number of rows and columns. This often arises in dealing with partial differential equations, for example. Suppose that $L_{\boldsymbol{g}}$ is the Lipschitz constant for $\boldsymbol{g}$ and $L_{\boldsymbol{g}} \ll \|A\|$. Provided the large eigenvalues of $A$ have negative real part, we can use this together with some suitable linear solvers to obtain efficient methods: (6.1.54) is then equivalent to

$$\boldsymbol{x} = \boldsymbol{u} + h\,\beta\left[A(\boldsymbol{w} + \alpha\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{w} + \alpha\boldsymbol{x})\right]$$

and we can re-write the system as

$$(I - h\,\beta\,\alpha\,A)\boldsymbol{x} = \boldsymbol{u} + h\beta\left[A\boldsymbol{w} + \boldsymbol{g}(\boldsymbol{w} + \alpha\boldsymbol{x})\right]$$

for which we have a modified fixed-point iteration

$$\boldsymbol{x}^{(m+1)} = (I - h\,\beta\,\alpha\,A)^{-1}\left\{\boldsymbol{u} + h\beta\left[A\boldsymbol{w} + \boldsymbol{g}(\boldsymbol{w} + \alpha\boldsymbol{x}^{(m)})\right]\right\}$$

which converges provided $\left\|(I - h\,\beta\,\alpha\,A)^{-1}\right\|\,h\,|\alpha\beta|\,L_{\boldsymbol{g}} < 1$. We do not expect $\left\|(I - h\,\beta\,\alpha\,A)^{-1}\right\|$ to be small, but we can reasonably hope for it to be modest. For example, if $-A$ is positive definite, then $\left\|(I - h\,\beta\,\alpha\,A)^{-1}\right\|_2 < 1$ provided $\alpha\beta > 0$.

Flexibility from the software may be desired to allow the user to provide efficient means of solving the equation $(I - h\,\beta\,\alpha\,A)\boldsymbol{x} = \boldsymbol{y}$ for $\boldsymbol{x}$ given $\boldsymbol{y}$. The user can provide or designate specific iterative methods and preconditioners tailored for the

user's problems. Exactly how this should be implemented depends on the programming language and software standards in use.

One approach is for the user to pass an *nlsolver* function to the *odesolver* function for solving the differential equation. The *nlsolver* function takes as input the data for the problem, plus whatever other data the user can provide to help solve the equations efficiently:

```
function nlsolver(u, w, α, hβ, f, p, ..., ϵ)
  ...
  return x
end
```

The $\boldsymbol{p}, \ldots$ inputs can be parameters, functions, links to preconditioners etc. The ODE solver then has the form

```
function odesolver(f, x_0, error tolerances etc., nlsolve, p, ...)
  ...
  ... // now to solve x = u + hβ f(t, w + αx)
  ϵ ← ... // accuracy needed for solving equations
  x ← nlsolve(u, w, α, hβ, f, p, ..., ϵ)
  ...
end
```

In this way, *odesolver* does not have to worry at all about how the solution is found, only the accuracy needed for it. If $\epsilon_0$ is the overall accuracy desired, then we should have $\epsilon \approx \epsilon_0 h$. If the solver fails, there should be a mechanism for passing the fact of the failure, plus useful information about the cause of the failure, back to the end-user. Many programming languages have error/exception mechanisms for doing this without requiring cumbersome error code returns that have been the mainstay of scientific computing in Fortran, for example.

Fully implicit Runge–Kutta methods pose a slightly different challenge. The Runge–Kutta equations (6.1.20)

$$\boldsymbol{v}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{v}_{k,i}) \qquad j = 1, 2, \ldots, s,$$

can be solved by Newton's method and its variants. To do this involves computing the updates $\boldsymbol{w}_{k,j}$ for $\boldsymbol{v}_{k,j}$, $j = 1, 2, \ldots, s$, by means of the linear systems

$$\begin{aligned} \boldsymbol{w}_{k,j} - \sum_{i=1}^{s} \nabla_{\boldsymbol{x}} \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{v}_{k,i})\, h\, a_{ji}\, \boldsymbol{w}_{k,i} \\ = -\left\{ \boldsymbol{v}_{k,j} - \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{v}_{k,i}) \right\}, \qquad j = 1, 2, \ldots, s. \end{aligned}$$

If we let $J_j = \nabla_{\boldsymbol{x}} \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h \sum_{i=1}^{s} a_{ji} \boldsymbol{v}_{k,i})$ be the Jacobian matrix of $\boldsymbol{f}$ for stage $j$ of the method, then we can write the linear system more succinctly as

$$\boldsymbol{w}_{k,j} - h \sum_{i=1}^{s} a_{ji}\, J_j \boldsymbol{w}_{k,i} = \boldsymbol{\delta}_j, \qquad j = 1, 2, \ldots, s.$$

If each $J_j$ is $n \times n$ this is an $ns \times ns$ linear system. Using a direct solver, like LU or Cholesky or QR factorization, will typically result in $\mathcal{O}(n^3 s^3)$ operations, which can be rather expensive. (A three-stage method would result in the time to solve increased by a factor of about $3^3 = 27$.) This linear system can be written out in a more extensive form:

$$\begin{bmatrix} I - ha_{11}J_1 & -ha_{12}J_1 & \cdots & -ha_{1s}J_1 \\ -ha_{21}J_2 & I - ha_{22}J_2 & \cdots & -ha_{2s}J_2 \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{21}J_s & -ha_{s2}J_s & \cdots & I - ha_{ss}J_s \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_{k,1} \\ \boldsymbol{w}_{k,2} \\ \vdots \\ \boldsymbol{w}_{k,s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta}_{k,1} \\ \boldsymbol{\delta}_{k,2} \\ \vdots \\ \boldsymbol{\delta}_{k,s} \end{bmatrix}.$$

While there are few evident shortcuts for solving such a linear system, at least the structure is clearer. We can, however, look for good approximations for which we can solve the system quickly. If $J_j \approx J_0 \approx \nabla_{\boldsymbol{x}} \boldsymbol{f}(t_k, \boldsymbol{x}_k)$, then we have the approximate linear system

$$\begin{bmatrix} I - ha_{11}J_0 & -ha_{12}J_0 & \cdots & -ha_{1s}J_0 \\ -ha_{21}J_0 & I - ha_{22}J_0 & \cdots & -ha_{2s}J_0 \\ \vdots & \vdots & \ddots & \vdots \\ -ha_{21}J_0 & -ha_{s2}J_0 & \cdots & I - ha_{ss}J_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{w}_{k,1} \\ \boldsymbol{w}_{k,2} \\ \vdots \\ \boldsymbol{w}_{k,s} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\delta}_{k,1} \\ \boldsymbol{\delta}_{k,2} \\ \vdots \\ \boldsymbol{\delta}_{k,s} \end{bmatrix}. \tag{6.1.56}$$

This can actually be solved much faster using *Kronecker* or *tensor product* methods: for $A$ $(r \times s)$ and $B$ $(m \times n)$, $A \otimes B$ is the $rm \times sn$ matrix

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1s}B \\ a_{21}B & a_{22}B & \cdots & a_{2s}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{r1}B & a_{r2}B & \cdots & a_{rs}B \end{bmatrix}. \tag{6.1.57}$$

Tensor products have the following properties:

- $\alpha(A \otimes B) = (\alpha A) \otimes B = A \otimes (\alpha B)$ for any scalar $\alpha$;
- $A \otimes B + A \otimes C = A \otimes (B + C)$ and $A \otimes C + B \otimes C = (A + B) \otimes C$;
- $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$, so $(A \otimes B)^{-1} = A^{-1} \otimes B^{-1}$;
- $(A \otimes B)^T = A^T \otimes B^T$;
- if $A$ and $B$ are upper triangular, then so is $A \otimes B$.

The eigenvalues of $A \otimes B$ are the products $\lambda\mu$ where $\lambda$ is an eigenvalue of $A$ and $\mu$ is an eigenvalue of $B$. A quick way to see this is to note that if $\overline{U}^T A\, U = S$ and $\overline{V}^T B\, V = T$ are the Schur decompositions (2.5.7) of $A$ and $B$ respectively, then

$$\overline{U \otimes V}^T (A \otimes B)(U \otimes V) = S \otimes T \qquad \text{upper triangular.}$$

Since $U$ and $V$ are unitary, $U^{-1} = \overline{U}^T$ and $V^{-1} = \overline{V}^T$, so $(U \otimes V)^{-1} = U^{-1} \otimes V^{-1} = \overline{U}^T \otimes \overline{V}^T = \overline{U \otimes V}^T$. As $S \otimes T$ is upper triangular, its eigenvalues (which are the eigenvalues of $A \otimes B$) are the diagonal entries of $S \otimes T$, which are products of diagonal entries of $S$ and diagonal entries of $T$. That is, each eigenvalue of $A \otimes B$ is the product of an eigenvalue of $A$ and an eigenvalue of $B$. Furthermore, every product of an eigenvalue of $A$ and an eigenvalue of $B$ is an eigenvalue of $A \otimes B$.

The system of equations (6.1.56) can be represented using Kronecker or tensor products:

$$(I \otimes I - h\, A \otimes J_0)\boldsymbol{w} = \boldsymbol{\delta}.$$

If we have a Schur decomposition $\overline{U}^T A\, U = S$ of $A$, then

$$\begin{aligned}(U \otimes I)(I \otimes I - h\, \overline{U}^T A U \otimes J_0)(\overline{U}^T \otimes I)\boldsymbol{w} = \boldsymbol{\delta}; \qquad &\text{that is,}\\ (I \otimes I - h\, S \otimes J_0)(\overline{U}^T \otimes I)\boldsymbol{w} = (\overline{U}^T \otimes I)\boldsymbol{\delta}.&\end{aligned}$$

Note that since $S$ is upper triangular, $S \otimes J_0$ is block upper triangular, and each block is a scalar multiple of $J_0$. Block backward substitution can then be used for these equations; the crucial step is solving $(I - h\, s_{kk}\, J_0)\boldsymbol{z} = \boldsymbol{b}$ for each $k$. For many cases, this approach is problematic as it involves doing complex arithmetic and complex function evaluations. It is probably better to use the real Schur decomposition, which has a $2 \times 2$ diagonal block in $S$ for each complex conjugate eigenvalue. This means that a linear system of the form

$$\begin{bmatrix} I - h\,\alpha\, J_0 & +h\,\beta J_0 \\ -h\,\beta J_0 & I - h\,\alpha\, J_0 \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \boldsymbol{z}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \boldsymbol{b}_2 \end{bmatrix} \qquad \text{or} \qquad (I - h\, a\, J_0)\boldsymbol{z} = \boldsymbol{b}$$

needs to be solved at each stage of using block backward substitution. This means the total time for solving the linearized Runge–Kutta equations can be reduced to $\mathcal{O}(n^3 s + n^2 s^2)$. In fact, if $A$ is diagonalizable then the cost can be reduced to $\mathcal{O}(n^3 s)$. For example, the 3-stage Gauss method, which has order 6, has eigenvalues $\approx 0.16410 \pm 0.42820\, i$ and $\approx 0.17172$. The three-stage Radau IIA method, which has order 5 and is stiffly accurate, has eigenvalues $\approx 0.16256 \pm 0.18495\, i$ and $\approx 0.27489$. The $A$ matrix in both cases is diagonalizable, and there is a real invertible matrix $X$ so that $X^{-1} A X$ is real block diagonal with a $2 \times 2$ block and a $1 \times 1$ block. So instead of solving a $3n \times 3n$ linear system at each step of Newton's method, we can solve an $n \times n$ and a $2n \times 2n$ system, which can give a significant improvement in the computational cost.

### 6.1.8 Error Estimates and Adaptive Methods

In many differential equations, there are periods of relatively little activity, followed by short bursts where things change rapidly. Take, for example, the Kepler problem:

$$m\frac{d^2\boldsymbol{x}}{dt^2} = -G\,M\,m\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2^3}. \tag{6.1.58}$$

This describes the motion of a planet around a relatively large star, like the Earth around the Sun, ignoring the effects of the other planets and celestial objects. It also describes the motion of asteroids. As is well known, the solution to (6.1.58) forms elliptic orbits with the Sun (at $\boldsymbol{x} = \boldsymbol{0}$) at one of the foci of the ellipse. Asteroids can have orbits with high eccentricity, so that while they linger for long periods far from the Sun, they fall toward it, and pass by close to the Sun where they have high velocity and high acceleration that changes direction quite rapidly. Solving such differential equations poses some challenges that we have not dealt with explicitly so far.

Our asymptotic convergence analysis indicates that (except for when the body impacts the Sun) as long as the step size is "small enough" then the error is also small, and we have an asymptotic estimate for how large that error would be. The standard fourth order Runge–Kutta method has a global error of $\mathcal{O}(h^4)$ as $h \to 0$. However, this avoids the question of how small is "small enough". This is clearly problem dependent, but even with $m = 1$ and $G\,M\,m = 1$ in (6.1.58), for the initial conditions $\boldsymbol{x}_0 = [1,\ 0]^T$ and $d\boldsymbol{x}/dt(0) = [0,\ 1/10]^T$, we find that even $h = 10^{-4}$ for the standard fourth-order Runge–Kutta method is not sufficient to make the error invisible to the eye. Figure 6.1.8 shows the results for exactly this case. The plot on the left shows the orbits with the Sun ($\boldsymbol{x} = \boldsymbol{0}$) denoted by a red asterisk. The upper right of Figure 6.1.8 shows the size of the acceleration vector $\boldsymbol{a}(t) = \boldsymbol{x}''(t)$, while the lower right shows the energy function $E(t) = \frac{1}{2}m\,\|\boldsymbol{v}(t)\|_2^2 - GMm/\,\|\boldsymbol{x}(t)\|$ where $\boldsymbol{v}(t) = \boldsymbol{x}'(t)$ is the velocity vector. Exact solutions of the Kepler problem conserve energy; that is, $E(t) = E(0)$ and is constant. However, the errors incurred in the numerical method by the close transit of the orbit near the Sun are large enough to produce clearly visible changes in the energy.

The solution is clearly to use smaller step sizes! This is a wasteful use of computational resources, since for most steps the acceleration is relatively modest, and large steps can be easily accommodated. So we want large time steps when we can use them, and short time steps when we must.

Adaptive step sizes are built into most modern ODE solvers. To implement them, we need to be able to estimate the size of the errors, and then change the step size to achieve a given error target. But we should be clear about what an adaptive method can and cannot do. The exponential growth of perturbations in persistently unstable differential equation like the Lorenz equations (6.1.13–6.1.15) (see Figure 6.1.2) means that to accurately predict errors, we would need to know how much errors are amplified *before we have solved the equations*. Instead, adaptive methods use local information from one or two steps to estimate the local truncation error (LTE),

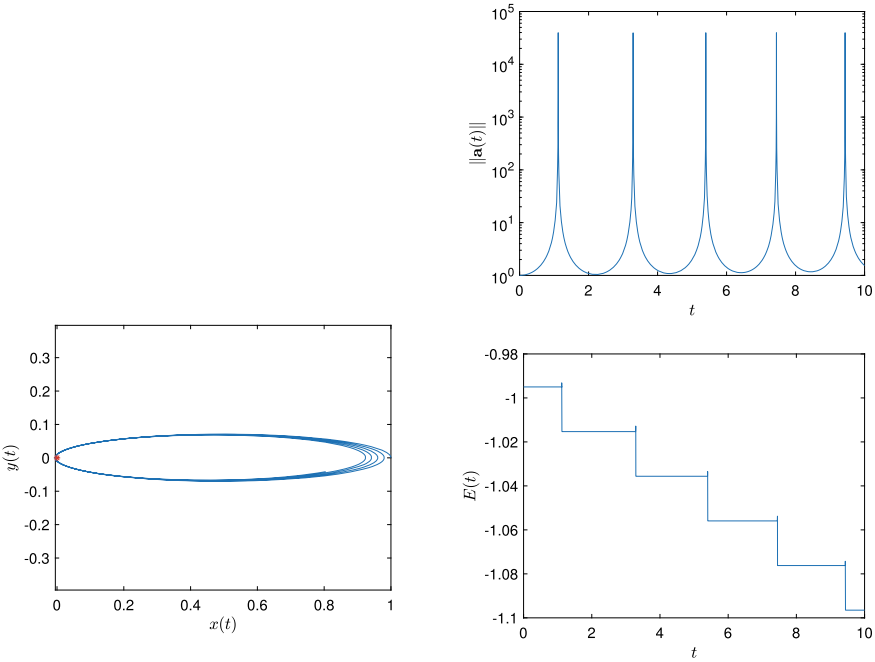



**Fig. 6.1.8** Results for Kepler problem with $\boldsymbol{x}_0 = [1,\ 0]^T$, $\boldsymbol{v}_0 = [0,\ 1/10]^T$ with fixed step size $h = 10^{-4}$ and the 4th order Runge–Kutta method

and adjust the step size $h$ so that the LTE per unit step is less than the user-specified target.

We need several new components in a successful adaptive method: a way of estimating the LTE, a way of determining the new step size, and a way to change the step size. For Runge–Kutta methods, changing the step size is trivial. For multistep methods, it is more complex. In either case, Algorithm 63 shows an outline of how the control mechanism can work. In Algorithm 63**,** $p$ is the order of the underlying ODE method. We have $h_{\min} \leq h \leq h_{\max}$ for all $h$ to achieve a target LTE per unit step $\leq \epsilon$. The parameter $0 < \gamma < 1$ is used to avoid excessive changes in the step size $h$, and to take into account the fact that the computed LTE estimate $\tau$ is only an estimate. If the $h$ update formula in line 16 was $h \leftarrow [\epsilon/(\tau/h)]^{1/p}\ h$ then the *predicted* LTE per unit step is $\epsilon$. However, any variation on this could result in the LTE per unit step estimate again exceeding $\epsilon$ on the next pass through the `while` loop. This can lead to an infinite loop where $h$ is *almost* small enough to get $\tau/h \leq \epsilon$. Instead we have a "fudge factor" $0 < \gamma < 1$ to ensure that this infinite loop does not occur. While frequent changes in step size can be easily accommodated with Runge–Kutta methods (line 18 is empty for Runge–Kutta methods), it may be damaging to multistep methods, as we will see.

**Algorithm 63** Adaptive ODE method

```
1   function adaptiveodesolver(f, x_0, t_0, t_end, h_0, h_min, h_max, ε, γ)
2     t ← t_0; x ← x_0; done ← t ≥ t_end; h ← h_0; k ← 0
3     while not done
4       if t + h ≥ t_end
5         h ← t_end − t; done ← true
6       end if
7       accepted ← false
8       compute solution estimate x̂
9       compute LTE estimate τ
10      if τ/h ≤ ε or h = h_min
11        accepted ← true; x ← x̂; t ← t + h
12      else
13        done ← false
14      end if
15      if τ/h ≤ (1/2)γε or τ/h > ε
16        h ← [γε/(τ/h)]^{1/p} h
17        h ← min(max(h, h_min), h_max)
18        change step size to h
19      end if
20      if accepted
21        k ← k + 1; x_k ← x; t_k ← t
22      end if
23    end while
24    return (x_0, x_1, ..., x_k), (t_0, t_1, ..., t_k)
25  end function
```

We now need to see how to estimate the local truncation error (LTE). For all methods of order $p$, one approach is to compute the results for one step of the method and another step of the method with half the step size. The difference in the results divided by $(1 - 2^{-p})$ is then an estimate of the LTE for the larger step size. However, this doubles the cost of each step, and for implicit methods, this can be very substantial.

For Runge–Kutta methods, a more popular approach is to have a pair of overlapping methods, one of order $p$ and the other of order $p + 1$. An example is the Runge–Kutta–Fehlberg method of orders four and five [89] is represented by the Butcher tableau

(6.1.59)

$$\begin{array}{c|cccccc}
0 & & & & & & \\
1/4 & 1/4 & & & & & \\
3/8 & 3/32 & 9/32 & & & & \\
12/13 & 1932/2197 & -7200/2197 & 7296/2197 & & & \\
1 & 439/216 & -8 & 3680/513 & -845/4104 & & \\
1/2 & -8/27 & 2 & -3544/2565 & 1859/4104 & -11/40 & \\
\hline
\boldsymbol{b} & 25/216 & 0 & 1408/2565 & 2197/4104 & -1/5 & 0 \\
\widehat{\boldsymbol{b}} & 16/135 & 0 & 6656/12825 & 28561/56430 & -9/50 & 2/55
\end{array},$$

The fourth order and fifth order methods are

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h \sum_{j=1}^{s} b_j \, \boldsymbol{v}_j,$$

$$\widehat{\boldsymbol{x}}_{k+1} = \widehat{\boldsymbol{x}}_k + h \sum_{j=1}^{s} \widehat{b}_j \, \boldsymbol{v}_j, \qquad \text{respectively.}$$

To estimate the LTE we compute with difference $\boldsymbol{\tau}_k = \widehat{\boldsymbol{x}}_{k+1} - \boldsymbol{x}_{k+1}$ with $\widehat{\boldsymbol{x}}_k = \boldsymbol{x}_k$ and set $\tau = \|\boldsymbol{\tau}_k\|$ for use in Algorithm 63. Note that $\boldsymbol{\tau}_k = h \sum_{j=1}^{s} (b_j - \widehat{b}_j) \boldsymbol{v}_j$, so $\tau / h = \left\| \sum_{j=1}^{s} (b_j - \widehat{b}_j) \boldsymbol{v}_j \right\|$.

This value $\boldsymbol{\tau}_k$ is an asymptotic estimate of the LTE *for the fourth order method*. Yet, often in practice, the fifth order method is the one that is actually used to update $\boldsymbol{x}_{k+1}$. It is perhaps strange to use the method for which the error estimate does not actually apply, but *usually* the LTE estimate for the fourth order method is much larger than the actual LTE for the fifth order method.

Multistep methods can also be used to efficiently obtain LTE estimates, by re-using information generated by the method itself. One approach is to use, say, the Adams–Bashforth method of order $p + 1$ to estimate the LTE for an order $p$ method of either the Adams–Moulton or BDF methods. The algorithm DIFSUB of Gear [100] uses a more sophisticated approach. The DIFSUB algorithm is also a variable order method. Variable order methods avoid the start-up problems for multistep methods as they start with order one methods: Euler's method or the implicit Euler method, which requires no $\boldsymbol{x}_{k-j}$ values for $j \geq 1$. Once the step size is small enough for Euler's method for a few steps, then the order used can be increased to two, allowing the step size to increase. In a few more steps, the method can be increased in order to third order, with a larger step size. This increase in order and step size can continue until the maximum order is reached, or increasing the order gives no increase in the step size.

Changing the step size for multistep methods is not a cost-free task. If the step size $h$ is changed, then the statement that $\boldsymbol{x}_{k-j} \approx \boldsymbol{x}(t_{k-j}) = \boldsymbol{x}(t_k - j\,h)$ becomes invalid. Instead, we use polynomial interpolation on the computed $\boldsymbol{x}_{k-j}$ to interpolate approximate values $\boldsymbol{x}^{+}_{k-j} \approx \boldsymbol{x}(t_k - j\,h^{+})$ for the new step size $h^{+}$. This process introduces errors in the new values $\boldsymbol{x}^{+}_{k-j}$. Indeed, repeatedly changing the step size can result in substantially larger errors than predicted by the LTE estimates. This means that it is important to set the value $\gamma$ in Algorithm 63.

More advanced variable step size and variable order/step size methods have been developed, such as the Sundials suite [127], VODE, and CVODE (based on a previous code LSODE). The MATLAB differential equations suite, written by Shampine [232], is also an example of an excellent collection of numerical ordinary differential equation solvers with step-size control.

### 6.1.9 Differential Algebraic Equations (DAEs)

*Differential algebraic equations* (DAEs) [10, 30] are a combination of a differential equation and a system of "algebraic" equations:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{y}), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \in \mathbb{R}^n, \tag{6.1.60}$$

$$\boldsymbol{0} = \boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{y}), \qquad \boldsymbol{y}(t_0) = \boldsymbol{y}_0 \in \mathbb{R}^m, \tag{6.1.61}$$

with the condition that $\boldsymbol{g}(t_0, \boldsymbol{x}_0, \boldsymbol{y}_0) = \boldsymbol{0}$. Often DAEs can be turned into differential equations, most commonly by solving (6.1.61) for $\boldsymbol{y}$ to give $\boldsymbol{y} = \boldsymbol{h}(t, \boldsymbol{x})$ so we can set

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}, \boldsymbol{h}(t, \boldsymbol{x})), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0.$$

In many cases, the equation $\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0}$ cannot be solved in this way, and even if we could (locally), we still need numerical methods to solve the system of equations. One way of doing this is to approximate the system (6.1.60, 6.1.61) by the system of differential equations

$$\frac{d\boldsymbol{x}_\epsilon}{dt} = \quad \boldsymbol{f}(t, \boldsymbol{x}_\epsilon, \boldsymbol{y}_\epsilon), \qquad \boldsymbol{x}_\epsilon(t_0) = \boldsymbol{x}_0 \in \mathbb{R}^n, \tag{6.1.62}$$

$$\epsilon \frac{d\boldsymbol{y}_\epsilon}{dt} = B\, \boldsymbol{g}(t, \boldsymbol{x}_\epsilon, \boldsymbol{y}_\epsilon), \qquad \boldsymbol{y}_\epsilon(t_0) = \boldsymbol{y}_0 \in \mathbb{R}^m, \tag{6.1.63}$$

and take $\epsilon \downarrow 0$. The matrix $B$ is chosen to be invertible and to make (6.1.63) a stable differential equation so that the equilibria are the solutions of $\boldsymbol{g}(t, \boldsymbol{x}, \boldsymbol{y}) = \boldsymbol{0}$ for given $\boldsymbol{x}$.

Since we are interested in $\epsilon \downarrow 0$, the effective stiffness of the DAE (6.1.60, 6.1.61) is infinite. We therefore seek methods designed for stiff ODEs to use here. These methods can be either implicit Runge–Kutta methods or BDF multistep methods. Of the Runge–Kutta methods, we should restrict attention to stiffly accurate methods: $\boldsymbol{b}^T = \boldsymbol{e}_s^T A$ in the Butcher tableau.

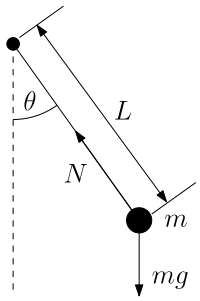
A good example of how we use DAEs comes from the equations for an idealized pendulum problem (Figure 6.1.9). There is a tension $N$ in the string connecting the mass $m$ to the pivot point $(0, 0)$. The position of the mass is $(x, y)$, and the main constraint is $x^2 + y^2 = L^2$. The tension $N$ is closely related to a Lagrange multiplier $\lambda$ for the constraint that $x^2 + y^2 = L^2$.

The differential-algebraic equations to solve are

$$m\frac{d^2x}{dt^2} = -\frac{N\,x}{\sqrt{x^2 + y^2}}, \tag{6.1.64}$$

$$m\frac{d^2y}{dt^2} = -\frac{N\,y}{\sqrt{x^2 + y^2}} - m\,g, \tag{6.1.65}$$

**Fig. 6.1.9** Pendulum




$$0 = L^2 - x^2 - y^2. \tag{6.1.66}$$

We have differential equations for $x$ and $y$; these are the "differential" variables. The other variable $N$ is defined implicitly through the "algebraic" equations. Unfortunately, we cannot solve the algebraic equations for $N$ in terms of $x$ and $y$. However, we can differentiate the algebraic equations (6.1.66) to get

$$0 = \frac{d}{dt}\left(L^2 - x^2 - y^2\right) = -2x\frac{dx}{dt} - 2y\frac{dy}{dt}.$$

This still does not give us a way to determine $N$ in terms of $x,\ y,\ dx/dt,\ dy/dt$. Differentiating one more time, using $u = dx/dt$ and $v = dy/dt$, we get

$$\begin{aligned} 0 = \frac{d}{dt}(-2x\,u - 2y\,v) &= -2\frac{dx}{dt}u - 2\frac{dy}{dt}v - 2x\frac{du}{dt} - 2y\frac{dv}{dt} \\ &= -2u^2 - 2v^2 - 2x\left(-\frac{N\,x}{m\sqrt{x^2+y^2}}\right) - 2y\left(-\frac{N\,y}{m\sqrt{x^2+y^2}} - g\right) \\ &= -2(u^2+v^2) + 2\frac{N(x^2+y^2)}{m\sqrt{x^2+y^2}} + 2gy. \end{aligned}$$

This now gives a way to solve for $N$ explicitly in terms of $x,\ y,\ u,\ v$:

$$N = \frac{m}{\sqrt{x^2+y^2}}\left[u^2 + v^2 - g\,y\right].$$

This formula for $N$ can be substituted into the other pendulum equations (6.1.64, 6.1.65) to give a complete system of differential equations. Alternatively, we can obtain a differential equation for $N$ with one more differentiation with respect to time.

This gives us multiple ways of formulating the same problem. Associated with each formulation is an *index*. This is the number of times the "algebraic" variables need to be differentiated in order to obtain differential equations for the algebraic variables. The original formulation (6.1.64–6.1.66) has index three, since three dif-

ferentiations with respect to $t$ are needed to get $dN/dt$ as a function of $x,\ y,\ u,\ v,\ N$. The formulation with algebraic equation

$$0 = -2x\,u - 2y\,v$$

has index two, while the formulation with algebraic equation

$$0 = -2(u^2 + v^2) + 2\frac{N(x^2 + y^2)}{m\sqrt{x^2 + y^2}} + 2gy$$

has index one. When we have a complete system of differential equations the system has index zero: it is just a system of differential equations.

Suppose we use the following formulation

$$\frac{dx}{dt} = u, \tag{6.1.67}$$

$$\frac{dy}{dt} = v, \tag{6.1.68}$$

$$m\frac{du}{dt} = -\frac{N\,x}{\sqrt{x^2 + y^2}}, \tag{6.1.69}$$

$$m\frac{dv}{dt} = -\frac{N\,y}{\sqrt{x^2 + y^2}} - m\,g, \tag{6.1.70}$$

$$N = \frac{m}{\sqrt{x^2 + y^2}}\left[u^2 + v^2 - g\,y\right] \tag{6.1.71}$$

and substitute (6.1.71) into (6.1.69, 6.1.70). The differential equations together with consistent initial conditions ($x_0^2 + y_0^2 = L^2$, $x_0u_0 + y_0v_0 = 0$) imply that $x(t)^2 + y(t)^2 = L^2$ for all $t$. However, our numerical methods do not give exact solutions, but rather approximate solutions. Since we obtained our formula for $N$ after taking *two* derivatives with respect to $t$, our numerical solution can be modeled as a solution to

$$\left|\frac{d^2}{dt^2}\left(x^2 + y^2 - L^2\right)\right| \le \epsilon$$

where $\epsilon = \mathcal{O}(h^p)$ is a bound on the LTE per unit time step; $p$ is the order of the method. For fixed time intervals, the error in $x^2 + y^2 - L^2$ is $\mathcal{O}(h^p)$ as expected. But for long time intervals, the error grows to be $\mathcal{O}(h^p\,(t - t_0)^2)$ provided $h^p\,(t - t_0)^2 \ll 1$. If we integrate for longer than $|t - t_0| \sim h^{-p/2}$, the mechanical energy can change sufficiently to cause an even more rapid growth in $\left|x^2 + y^2 - L^2\right|$. This drift can be clearly seen in Figure 6.1.10, which shows $x(t)^2 + y(t)^2 - L^2$ against $t$ for using Heun's method with step size $h = 10^{-2}$. (This uses $m = 1$, $L = 1/2$, and $g = 9.82$.)

By using the original DAE formulation (6.1.64–6.1.66) we can ensure that $x^2 + y^2 - L^2 \approx 0$ is maintained. However, the order of the method used must at least exceed the index of the DAE. Now we will look at some methods for solving DAEs.

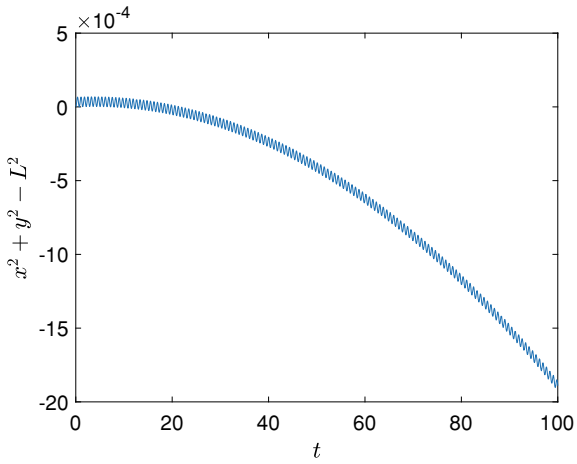



**Fig. 6.1.10** Drift in (6.1.67–6.1.71) using Heun's method with $h = 10^{-2}$

### 6.1.9.1 Runge–Kutta Methods for DAEs

We need to use implicit Runge–Kutta methods that are stiffly accurate. But we need to see how we can adapt Runge–Kutta methods to DAEs (6.1.60, 6.1.61). The standard Runge–Kutta equations (6.1.20, 6.1.21)

$$\boldsymbol{v}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{k,i}) \qquad j = 1, 2, \ldots, s,$$

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\sum_{j=1}^{s} b_j\, \boldsymbol{v}_{k,j}$$

apply to the equation $d\boldsymbol{x}/dt = \boldsymbol{f}(t, \boldsymbol{x})$. If we think of a DAE as the limit as $\epsilon \downarrow 0$ of (6.1.62, 6.1.63)

$$\begin{aligned} \frac{d\boldsymbol{x}_\epsilon}{dt} &= \quad \boldsymbol{f}(t, \boldsymbol{x}_\epsilon, \boldsymbol{y}_\epsilon), \qquad \boldsymbol{x}_\epsilon(t_0) = \boldsymbol{x}_0, \\ \epsilon\frac{d\boldsymbol{y}_\epsilon}{dt} &= B\,\boldsymbol{g}(t, \boldsymbol{x}_\epsilon, \boldsymbol{y}_\epsilon), \qquad \boldsymbol{y}_\epsilon(t_0) = \boldsymbol{y}_0, \end{aligned}$$

would give the approximate equations where $\boldsymbol{w}_{\epsilon,k,j}$ correspond to $d\boldsymbol{y}_\epsilon/dt$ values,

$$\boldsymbol{v}_{\epsilon,k,j} = \quad \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{\epsilon,k,i},\ \boldsymbol{y}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{w}_{\epsilon,k,i}) \qquad j = 1, 2, \ldots, s,$$

$$\epsilon\boldsymbol{w}_{\epsilon,k,j} = B\,\boldsymbol{g}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{\epsilon,k,i},\ \boldsymbol{y}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{w}_{\epsilon,k,i}) \qquad j = 1, 2, \ldots, s.$$

Taking $\epsilon \downarrow 0$ and using invertibility of $B$ gives the DAE Runge–Kutta equations:

(6.1.72)
$$\boldsymbol{v}_{k,j} = \boldsymbol{f}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{k,i},\ \boldsymbol{y}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{w}_{k,i}) \qquad j = 1, 2, \ldots, s,$$

(6.1.73)
$$\boldsymbol{0} = \boldsymbol{g}(t_k + c_j h,\ \boldsymbol{x}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{v}_{k,i},\ \boldsymbol{y}_k + h\sum_{i=1}^{s} a_{ji}\boldsymbol{w}_{k,i}) \qquad j = 1, 2, \ldots, s.$$

The new $\boldsymbol{x}$ and $\boldsymbol{y}$ values are

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\sum_{j=1}^{s} b_j \boldsymbol{v}_{k,j},$$
$$\boldsymbol{y}_{k+1} = \boldsymbol{y}_k + h\sum_{j=1}^{s} b_j \boldsymbol{w}_{k,j}.$$

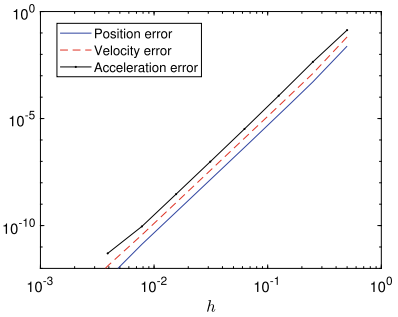
For a stiffly accurate method, $b_j = a_{s,j}$ for all $j$ and $c_s = 1$, so $\boldsymbol{g}(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}) = \boldsymbol{0}$. That is, we can guarantee that the condition $\boldsymbol{g}(t_k, \boldsymbol{x}_k, \boldsymbol{y}_k) = \boldsymbol{0}$ for all $k = 1, 2, \ldots$, at least to the accuracy with which the equations (6.1.72, 6.1.73) are solved. There is certainly no drift occurring here.

The Radau IIA methods are arguably the best Runge–Kutta methods for DAEs: they have near to maximal order. Only Gauss methods have higher order for the same number of stages, but Gauss methods are not stiffly accurate.

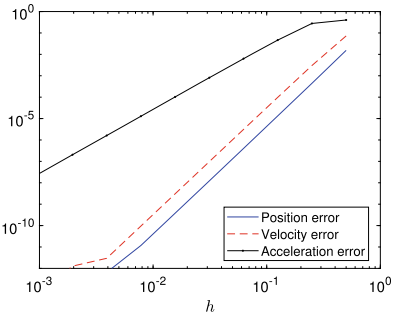
To see how this works in practice, Figure 6.1.11 shows the errors for the pendulum problem formulated as an index 1, index 2, or index 3 DAE for the 3-stage Radau IIA method. Note that different components of the solution have different error behaviors for the higher order DAE formulations. The slopes as estimated and predicted theoretically as shown in Table 6.1.8. The theoretical basis for these orders of accuracy was developed in [118, 135].

**Table 6.1.8** Slopes and theoretically predicted orders of errors in positions, velocities and forces for DAE formulations of different indexes using the 3-stage Radau IIA method

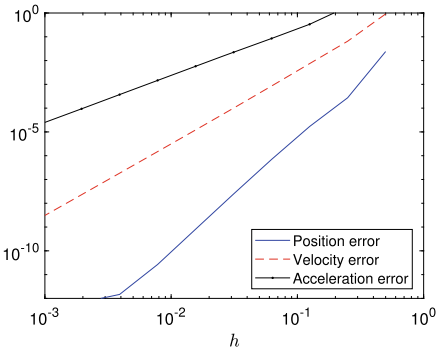
| DAE index: | One | | Two | | Three | |
|---|---|---|---|---|---|---|
| | Slope | Order | Slope | Order | Slope | Order |
| positions | 5.09 | 5 | 4.89 | 5 | 4.58 | 5 |
| velocities | 4.98 | 5 | 4.99 | 5 | 3.07 | 3 |
| forces | 4.95 | 5 | 2.90 | 3 | 2.05 | 2 |




(A) Index 1




(B) Index 2




(C) Index 3

**Fig. 6.1.11** Results for DAE formulations of the pendulum problem

### 6.1.9.2 BDF Methods for DAEs

The first methods developed specifically for solving DAEs were the BDF methods, with software developed including Hindmarsh's LSODI [126, pp. 312-316] and Petzold's DASSL [202]. For the DAE (6.1.60, 6.1.61), the equations to solve for the BDF method (6.1.39) are

$$\boldsymbol{x}_{k+1} = \sum_{j=0}^{m-1} \alpha_j \, \boldsymbol{x}_{k-j} + h \, \beta \, \boldsymbol{f}(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}), \tag{6.1.74}$$

$$\boldsymbol{0} = \boldsymbol{g}(t_{k+1}, \boldsymbol{x}_{k+1}, \boldsymbol{y}_{k+1}). \tag{6.1.75}$$

BDF methods can solve DAEs of index one and two and have the same order of accuracy $m$ as for solving ODEs provided the starting values $\boldsymbol{x}_j$ have errors $\mathcal{O}(h^{m+1})$ and $\boldsymbol{y}_j$ have errors $\mathcal{O}(h^m)$ for $j = 0, 1, \ldots, m-1$ [117, Sec. VI.2 & VII.3].

**Exercises.**

(1) Use Euler, Heun and the standard 4th order Runge–Kutta methods for the differential equation $dy/dt = 1 + y^2$ with $y(0) = 0$ on the interval $[0, 1]$. The exact solution is $y(t) = \tan t$. Do this for step-sizes $h = 2^{-k}$, $k = 1, 2, \ldots, 10$. Compute and plot the maximum error $\max_{k:0\le t_k\le 1} |y(t_k) - y_k|$ against $h$ on a log–log plot. Empirically estimate the order of convergence for the three methods from this data.

(2) Repeat Exercise 1 for the differential equation $dy/dt = y$, $y(0) = 1$ on the interval $[0, 1]$.

(3) The Kepler problem is the problem of determining the motion of a particle of mass $m$ around a fixed mass $M$ at the origin under the inverse square law of Newtonian gravitation. That is, we want to solve the differential equation

$$m\frac{d^2\boldsymbol{x}}{dt^2} = -GMm\frac{1}{\|\boldsymbol{x}\|_2^2}\frac{\boldsymbol{x}}{\|\boldsymbol{x}\|_2}$$

with given initial values for position $\boldsymbol{x}(0) = \boldsymbol{x}_0$ and velocity $d\boldsymbol{x}/dt(0) = \boldsymbol{v}_0$. Note that $G$ is the universal gravitational constant. Since the orbit remains in the plane generated by the origin, $\boldsymbol{x}_0$, and $\boldsymbol{v}_0$, we can assume without loss of generality that $\boldsymbol{x}(t),\ \boldsymbol{v}(t) \in \mathbb{R}^2$. Solve this differential equation for $GM = 1$, $\boldsymbol{x}_0 = [1, 0]^T$, and $\boldsymbol{v}_0 = [0, 1]^T$; also solve for $GM = 1$, $\boldsymbol{x}_0 = [1,\ 0]^T$, and $\boldsymbol{v}_0 = [0,\ 0.2]^T$. Note that this 2nd order equation in $\mathbb{R}^2$ should be first represented as a first order equation in $\mathbb{R}^4$. Do this for step sizes $h = 10^{-k}$, $k = 1, 2, \ldots, 5$, for an interval $t \in [0, 10]$. Plot $\boldsymbol{x}(t)$ against $t$, and also the orbit $\boldsymbol{x}(t)$ as a plot in $\mathbb{R}^2$. Report on how small $h$ needs to be in order to obtain even moderately accurate solutions.

(4) For Exercise 3, show that the energy $E(\boldsymbol{x}(t),\ \boldsymbol{v}(t)) := \frac{1}{2}m\,\|\boldsymbol{v}(t)\|_2^2 - GMm/\,\|\boldsymbol{x}\|_2$ is constant along any exact trajectory. [**Hint:** Show $(d/dt)E(\boldsymbol{x}(t),\ \boldsymbol{v}(t)) = 0$ using the differential equation.] Compute the energy as a function of time for the numerical solutions obtained in Exercise 3. Use this as a check on the error of the solution.

(5) "Chaotic" systems have exponentially diverging solutions until the difference becomes large. Consider, for example, the Lorenz equations [166]

$$\begin{aligned}\frac{dx}{dt} &= \sigma(y - x),\\ \frac{dy}{dt} &= x(\rho - z) - y,\\ \frac{dz}{dt} &= xy - \beta z,\end{aligned}$$

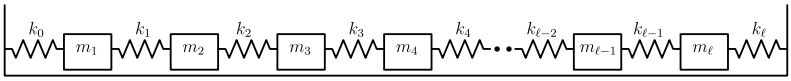
with $\sigma = 10$, $\beta = 8/3$, and $\rho = 28$.

(a) Numerically solve the Lorenz equations using the standard 4th order Runge–Kutta method with $h = 10^{-3}$ with initial conditions $\mathbf{x}_0^T = [x_0, y_0, z_0] = [1, -1, 1]$ and $\mathbf{x}_0^T = [x_0, y_0, z_0] = [1, -1 + 10^{-6}, 1]$.
(b) Plot the difference between the two numerical solutions over the interval $0 \le t \le 50$. Since the difference changes so much in size, use a logarithmic scale on the vertical axis.
(c) Repeat (a) and (b) with $h = \frac{1}{2} \times 10^{-3}$.

(6) ⚠ The Runge–Kutta–Fehlberg (6.1.59) method gives an estimate of the error per step which can be used to adjust the step size $h$ with each step. Implement the adaptive ODE solver Algorithm 63 and apply it to the Kepler problem (Exercise 3). Compare the number of function evaluations needed to achieve a change of energy of no more than $10^{-6}$, especially for the case where $GM = 1$, $\boldsymbol{x}_0 = [1, 0]^T$, and $\boldsymbol{v}_0 = [0, 0.2]^T$.

(7) Implement a general purpose solver for diagonally implicit Runge–Kutta methods. To make it general purpose, one of the inputs to the function is a *solver*$(t, \alpha, \beta, \boldsymbol{f}, \nabla \boldsymbol{f}, \boldsymbol{p}, z_0)$ function for solving $\boldsymbol{z} = \boldsymbol{y} + \alpha \boldsymbol{f}(t, \boldsymbol{y} + \beta \boldsymbol{z}; \boldsymbol{p})$ for $\boldsymbol{z}$. Here the $\boldsymbol{p}$ is a vector or some other structure of parameters for $\boldsymbol{f}$. Include default *solver* functions for the fixed point iteration $\boldsymbol{z}_{m+1} = \boldsymbol{y} + \alpha \boldsymbol{f}(t, \boldsymbol{y} + \beta \boldsymbol{z}_m; \boldsymbol{p})$, and for a default guarded Newton solver. Note that the use of the *solver* function must be flexible enough to allow for functions $\boldsymbol{f}$ where the Jacobian matrix $\nabla \boldsymbol{f}$ is *not* available, or where $\nabla \boldsymbol{f}$ returns a *representation* or *approximation* of the Jacobian matrix as long as an appropriate *solver* function using the output of $\nabla \boldsymbol{f}$ is provided. Test this on the implicit trapezoidal method for the Kepler problem (Exercise 3).

(8) Consider the $n$-body mass-spring system shown below.




The differential equations for the displacements $u_j(t)$ of particle $j$ are

$$m_j \frac{d^2 u_j}{dt^2} = k_{j-1}(u_{j-1} - u_j) + k_j(u_{j+1} - u_j), \qquad j = 1, \ldots, \ell,$$

where we take $u_0 = 0$ and $u_{\ell+1} = 0$. This approximates the wave equation $M\, \partial^2 u/\partial t^2 = K\, \partial^2 u/\partial x^2$ for large $\ell$ if $m_j = M/\ell$ and $k_j = K\, \ell$ for all $j$. Solve this using Heun's method and the implicit trapezoidal rule for $M = K = 1$ and $\ell = 2^r$ with $r = 1, 2, \ldots, 10$. Use this to determine the maximum step size needed for Heun's method for stability empirically for each $\ell$ used, and compare this with the theoretically predicted stability limits. What happens with the solution using the implicit trapezoidal rule with $1/\ell \ll h \ll 1$?
⚠ Read about the Courant–Friedrichs–Levy (CFL) condition for numerically solving the wave equation. Strictly speaking, the CFL condition only applies for a particular explicit method for solving the wave equation, but not for implicit methods. Discuss why the CFL condition might be useful to keep in mind even when using implicit methods.

(9) Solve the three-dimensional double pendulum problem as a system of differential-algebraic equations

$$
\begin{aligned}
m_1 \frac{d^2 \boldsymbol{x}_1}{dt^2} &= -m_1 g \boldsymbol{e}_3 - T_1 \frac{\boldsymbol{x}_1}{\|\boldsymbol{x}_1\|_2} + T_2 \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{\|\boldsymbol{x}_2 - \boldsymbol{x}_1\|_2}, \\
m_2 \frac{d^2 \boldsymbol{x}_2}{dt^2} &= -m_2 g \boldsymbol{e}_3 - T_2 \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{\|\boldsymbol{x}_2 - \boldsymbol{x}_1\|_2}, \\
L_1^2 = \|\boldsymbol{x}_1\|_2^2, \quad &\text{and} \quad L_2^2 = \|\boldsymbol{x}_2 - \boldsymbol{x}_1\|_2^2 .
\end{aligned}
$$

In this system, $g$ is the gravitational acceleration, $m_j$ is the mass of particle $j$, $L_j$ is the length of the rod connecting to particle $j$, and $T_j$ is the tension in the rod of length $L_j$. Here the $m_j$'s and $L_j$'s are constants and the $T_j$'s are algebraic variables. Note that the problem can be reformulated to use modified tensions $\widetilde{T}_1 = T_1 / \|\boldsymbol{x}_1\|_2$ and $\widetilde{T}_2 = T_2 / \|\boldsymbol{x}_2 - \boldsymbol{x}_1\|_2$. Solve this for the specific case with $\boldsymbol{x}_1 = [1,\ 1,\ 0]^T$ and $\boldsymbol{x}_2 = [0,\ 1,\ -1]^T$ and zero initial velocities ($L_1$ and $L_2$ are determined consistent with this data), and $g = m_j = 1$ for all $j$. For the method, use the 3-stage Radau IIA method, solving the problem as an index 3 DAE.

(10) Show that the seventh order BDF method fails to be stable even if $\boldsymbol{f}(t, \boldsymbol{y})$ is identically zero.

## 6.2 Ordinary Differential Equations—Boundary Value Problems

General boundary value problems (BVPs) have the form: given $\boldsymbol{f} : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$, $\boldsymbol{g} : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$, and $a < b$, find the function $\boldsymbol{x} : [a, b] \to \mathbb{R}^n$ satisfying

$$
\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}), \qquad \boldsymbol{g}(\boldsymbol{x}(a), \boldsymbol{x}(b)) = \boldsymbol{0}. \tag{6.2.1}
$$

Particular examples that commonly arise have the form

$$
\frac{d^2 \boldsymbol{y}}{dt^2} = \boldsymbol{f}(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}), \qquad \boldsymbol{y}(a) = \boldsymbol{y}_a, \qquad \boldsymbol{y}(b) = \boldsymbol{y}_b \tag{6.2.2}
$$

for given values $\boldsymbol{y}_a$ and $\boldsymbol{y}_b$. An example might come from looking for steady state solutions of diffusion equations. Consider, for example, the concentration $c(t, x)$ of oxygen in a one-dimensional medium in which it can diffuse, and is absorbed at a rate proportional to its concentration:

$$
\frac{\partial c}{\partial t} = D \frac{\partial^2 c}{\partial x^2} - b\, c
$$

If we assume that oxygen is supplied at $x = L$ and there is an impermeable boundary at $x = 0$, then

$$c(t, L) = c_{\text{end}} > 0 \quad \text{and} \quad \frac{\partial c}{\partial x}(t, 0) = 0.$$

The steady state then satisfies the BVP

$$D\frac{d^2c}{dx^2} = b\,c, \qquad c(L) = c_{\text{end}}, \qquad \frac{dc}{dx}(0) = 0. \tag{6.2.3}$$

Existence and uniqueness of solutions often depend on the specific problem. General existence results usually require some topological assumptions. For example, if $\boldsymbol{x}^T \boldsymbol{f}(t, \boldsymbol{x}) < 0$ for all $t$ and $\boldsymbol{x}$ where $\|\boldsymbol{x}\| = R$, then $\|\boldsymbol{x}(a)\| \le R$ implies $\|\boldsymbol{x}(b)\| \le R$. The map $\boldsymbol{x}(a) \mapsto \boldsymbol{x}(b)$ is continuous. Provided $\|\boldsymbol{h}(\boldsymbol{z})\| \le R$ whenever $\|\boldsymbol{z}\| \le R$, the function $\boldsymbol{x}(a) \mapsto \boldsymbol{h}(\boldsymbol{x}(b))$ has a fixed point by *Brouwer's fixed point theorem*.

Other cases should be considered from the point of view of optimization problems in the *calculus of variations*:

$$\min_{\boldsymbol{y}(\cdot)} \int_a^b F(t,\ \boldsymbol{y}(t),\ \frac{d\boldsymbol{y}}{dt}(t))\,dt \tag{6.2.4}$$

$$\text{subject to } \boldsymbol{y}(a) = \boldsymbol{y}_a, \quad \boldsymbol{y}(b) = \boldsymbol{y}_b. \tag{6.2.5}$$

This is essentially the case with the problem of oxygen concentration shown above. The solution of (6.2.3) minimizes

$$\int_0^L \left( D\left(\frac{dc}{dx}\right)^2 + b\,c^2 \right) dx \quad \text{subject to } c(L) = c_0.$$

Solutions exist for (6.2.4) with or without (6.2.5) provided $F(t, \boldsymbol{y}, \boldsymbol{v})$ is continuous in $(t, \boldsymbol{y}, \boldsymbol{v})$, is convex in $\boldsymbol{v}$, and coercive in $(\boldsymbol{y}, \boldsymbol{v})$: that is, $F(t, \boldsymbol{y}, \boldsymbol{v}) \to \infty$ if $\|\boldsymbol{y}\| + \|\boldsymbol{v}\| \to \infty$. Necessary conditions for a minimizer of (6.2.4, 6.2.5) are the Euler–Lagrange equations provided $F(t, \boldsymbol{y}, \boldsymbol{v})$ is smooth:

$$\frac{d}{dt}\nabla_{\boldsymbol{v}} F(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}) - \nabla_{\boldsymbol{y}} F(t, \boldsymbol{y}, \frac{d\boldsymbol{y}}{dt}) = \boldsymbol{0}. \tag{6.2.6}$$

For numerical methods, we can go in two different directions. One is to treat the problem as a one-dimensional partial differential equation (PDE) and use those techniques. Another is to use the technology of initial value problems to solve the boundary value problem.

**Example 6.10** As a test problem, we will consider the problem of minimizing the area of a surface of revolution about the $x$-axis, as illustrated in Figure 6.2.1. It is the shape of a soap film between two co-axial rings.

The quantity to minimize is

$$A[y] = \int_a^b \sqrt{1 + (\frac{dy}{dx})^2}\, 2\pi\, y\, dx,$$

subject to the condition that $y(a) = y(b) = y_0$. The solution must satisfy the Euler–Lagrange equations for this integrand, which are

$$\frac{d}{dx}\left(\frac{y(dy/dx)}{\sqrt{1+(dy/dx)^2}}\right) - \sqrt{1+(dy/dx)^2}\, y = 0, \qquad \text{or}$$

$$y\,(1+(\frac{dy}{dx})^2)^{-3/2}\frac{d^2y}{dx^2} = \sqrt{1+(\frac{dy}{dx})^2} - \frac{(dy/dx)^2}{\sqrt{1+(dy/dx)^2}} \qquad \text{so}$$

$$\frac{d^2y}{dx^2} = \frac{1}{y}\left[\left(1+(\frac{dy}{dx})^2\right)^2 - \left(1+(\frac{dy}{dx})^2\right)(\frac{dy}{dx})^2\right]$$

$$= \frac{1}{y}\left[1+(\frac{dy}{dx})^2\right].$$

Writing this as a two-dimensional system with $v = dy/dx$, we have

$$\frac{dy}{dx} = v, \tag{6.2.7}$$

$$\frac{dv}{dx} = \frac{1}{y}\left[1+v^2\right]. \tag{6.2.8}$$

### 6.2.1 Shooting Methods

Shooting methods aim to use the solution map $\boldsymbol{S}\colon \boldsymbol{x}(a) \mapsto \boldsymbol{x}(b)$ of the differential equation $d\boldsymbol{x}/dt = \boldsymbol{f}(t, \boldsymbol{x})$ to help solve the problem. The equation $\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}(a),\ \boldsymbol{x}(b))$ can be written as $\boldsymbol{0} = \boldsymbol{g}(\boldsymbol{x}(a),\ \boldsymbol{S}(\boldsymbol{x}(a)))$. For this we can use Newton's method, for example. This requires computing an estimate of the Jacobian matrix

$$\nabla_{\boldsymbol{x}(a)}\left[\boldsymbol{g}(\boldsymbol{x}(a),\ \boldsymbol{S}(\boldsymbol{x}(a)))\right] = \nabla_{\boldsymbol{x}_1}\boldsymbol{g}(\boldsymbol{x}(a),\ \boldsymbol{S}(\boldsymbol{x}(a))) + \nabla_{\boldsymbol{x}_2}\boldsymbol{g}(\boldsymbol{x}(a),\ \boldsymbol{S}(\boldsymbol{x}(a)))\,\nabla\boldsymbol{S}(\boldsymbol{x}(a)).$$

We need to determine $\nabla\boldsymbol{S}(\boldsymbol{x}(a))$. Let $\boldsymbol{x}(t;\boldsymbol{x}_0)$ be the solution of the differential equation $d\boldsymbol{x}/dt = \boldsymbol{f}(t,\boldsymbol{x})$ with $\boldsymbol{x}(a) = \boldsymbol{x}_0$. If $\Phi(t) = \nabla_{\boldsymbol{x}_0}\boldsymbol{x}(t;\boldsymbol{x}_0)$ then

$$\begin{aligned}\frac{d}{dt}\Phi(t) &= \frac{d}{dt}\nabla_{\boldsymbol{x}_0}\boldsymbol{x}(t;\boldsymbol{x}_0) = \nabla_{\boldsymbol{x}_0}\frac{d}{dt}\boldsymbol{x}(t;\boldsymbol{x}_0)\\ &= \nabla_{\boldsymbol{x}_0}\left[\boldsymbol{f}(t,\boldsymbol{x}(t;\boldsymbol{x}_0))\right]\\ &= \nabla_{\boldsymbol{x}}\boldsymbol{f}(t,\ \boldsymbol{x}(t;\boldsymbol{x}_0))\,\nabla_{\boldsymbol{x}_0}\boldsymbol{x}(t;\boldsymbol{x}_0)\\ &= \nabla_{\boldsymbol{x}}\boldsymbol{f}(t,\ \boldsymbol{x}(t;\boldsymbol{x}_0))\,\Phi(t). \qquad (6.2.9)\end{aligned}$$

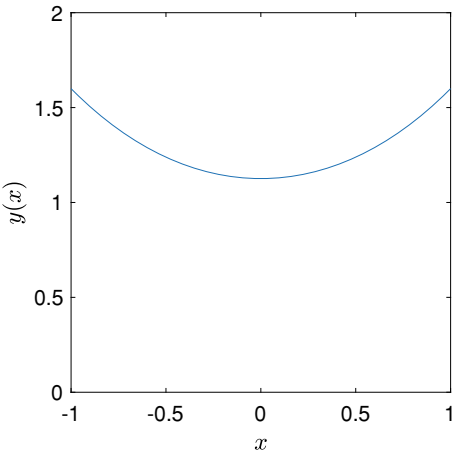



**Fig. 6.2.1** Solution to soap film boundary value problem

This is the *variational equation* for the differential equation $d\boldsymbol{x}dt = \boldsymbol{f}(t, \boldsymbol{x})$. For initial conditions for $\Phi$, we note that $\boldsymbol{x}(a;\ \boldsymbol{x}_0) = \boldsymbol{x}_0$. So $\Phi(a) = \nabla_{\boldsymbol{x}_0}\boldsymbol{x}(a;\ \boldsymbol{x}_0) = \nabla_{\boldsymbol{x}_0}\boldsymbol{x}_0 = I$. Then $\boldsymbol{x}(t)$ and $\Phi(t)$ can be computed together using a standard numerical ODE solver applied to

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{x}\\ \Phi\end{bmatrix} = \begin{bmatrix}\boldsymbol{f}(t,\boldsymbol{x})\\ \nabla_{\boldsymbol{x}}\boldsymbol{f}(t,\ \boldsymbol{x})\,\Phi\end{bmatrix}, \qquad \begin{bmatrix}\boldsymbol{x}(a)\\ \Phi(a)\end{bmatrix} = \begin{bmatrix}\boldsymbol{x}_0\\ I\end{bmatrix}. \tag{6.2.10}$$

This does require computing the Jacobian matrix of $\boldsymbol{f}(t, \boldsymbol{x})$ with respect to $\boldsymbol{x}$. This can be done using symbolic computation, numerical differentiation formulas (see Section 5.5.1.1), or automatic differentiation (see Section 5.5.2).

Once $\Phi(t)$ has been computed, $\nabla \boldsymbol{S}(\boldsymbol{x}_0) = \Phi(b)$, and we can apply Newton's method.

**Example 6.11** For a concrete example, we solve (6.2.7, 6.2.8) with the boundary conditions: $y(a) = y(b) = y_0$ with $a = -1$, $b = +1$, $y_0 = 1.6$. We use the guarded Newton method to compute $v(a) = v_0$ so that $y(b) = y_0$; the variational equations (6.2.10) are used to compute the derivative $\partial\, y(b)/\partial v_0$ needed for the Newton method. Starting from the estimate $v_0 = 0$, the computed value of $v_0 \approx -1.0096$ for which $|y(b) - y_0| < 10^{-7}$ is obtained in five Newton steps. No backtracking is necessary for this example. The resulting function $y(x)$ is then computed using the values $(y_0, v_0)$. The standard fourth order Runge–Kutta method with step size $10^{-2}$ is used throughout for solving the original and variational differential equations. The resulting trajectory is shown in Figure 6.2.1.

There are actually two solutions for this value of $y_0$, and no solutions if $y_0 < y_0^* \approx 1.508879$.

**Example 6.12** Shooting methods can obtain highly accurate solutions to boundary value problems of this type. However, they can also become numerically unstable if

the differential equation is unstable as an initial value problem. Consider for example,

$$\frac{d^2y}{dx^2} = 100\,y, \qquad y(0) = y(10) = 1. \tag{6.2.11}$$

This is an example of (6.2.3) where $b/D = 100$. Using a shooting method means solving

$$\frac{d^2y}{dx^2} = 100\,y, \qquad y(0) = 1, \qquad \frac{dy}{dx}(0) = v_0$$

where we wish to find $v_0$ by solving for $y(10) = 1$. The exact solution of the above equation is

$$y(x) = \frac{1 + v_0/10}{2} e^{10\,x} + \frac{1 - v_0/10}{2} e^{-10\,x}.$$

Putting $x = 10$ we see that

$$\begin{aligned} y(10) &= \frac{1 + v_0/10}{2} e^{100} + \frac{1 - v_0/10}{2} e^{-100} \\ &= \cosh(100) + \frac{v_0}{10} \sinh(100), \end{aligned}$$

so the derivative is $\partial\, y(10)/\partial v_0 = \sinh(100)/10 \approx 1.34 \times 10^{42}$. Any perturbation in $v_0$ is amplified by this amount. Even if the error in $v_0$ is of size of unit roundoff for double precision ($\mathbf{u} \approx 2.2 \times 10^{-16}$), the computed solution will probably be in error by about $1.34 \times 10^{42} \times 2.2 \times 10^{-16} \approx 3 \times 10^{26}$. Yet the boundary value problem is actually very well conditioned *as a boundary value problem*. The exact solution is

$$\begin{aligned} y(x) &= \frac{(y(0) - e^{-100}y(10))e^{-10\,x} + (y(10) - e^{-100}y(0))e^{+10(10-x)}}{1 - e^{-200}} \\ &\approx y(0)\,e^{-10\,x} + y(10)\,e^{-10(10-x)}, \end{aligned}$$

with the approximation being within about $e^{-100}\ \max(|y(0)|\,,\ |y(10)|)$ of the exact solution.

### 6.2.2 *Multiple Shooting*

The most important issue with shooting methods is that the condition number of the Jacobian matrix $\Phi(t)$ in the variational equation (6.2.10) typically grows exponentially as $t \to \infty$. This can result in extremely ill-conditioned equations to solve for the starting point. We can avoid this extreme ill-conditioning by sub-dividing the interval $[a, b]$ into smaller pieces $a = t_0 < t_1 < \cdots < t_m = b$. Then to solve $\boldsymbol{g}(\boldsymbol{x}(a), \boldsymbol{x}(b)) = \mathbf{0}$ where $d\boldsymbol{x}/dt = \boldsymbol{f}(t, \boldsymbol{x})$ we now have additional equations to

satisfy so that $\boldsymbol{x}(t_j^+) = \boldsymbol{x}(t_j^-)$ at every interior break point $t_j$, $j = 1, 2, \ldots, m-1$. We have the functions $\boldsymbol{S}_j(\boldsymbol{x}_j) = \boldsymbol{z}(t_{j+1})$ where $d\boldsymbol{z}/dt = \boldsymbol{f}(t, \boldsymbol{z}(t))$ and $\boldsymbol{z}(t_j) = \boldsymbol{x}_j$. As for the standard shooting algorithm, $\nabla \boldsymbol{S}_j(\boldsymbol{x}_j)$ can be computed by means of the variational equation (6.2.10) except that $\nabla \boldsymbol{S}_j(\boldsymbol{x}_j) = \Phi_j(t_{j+1})$ where $\Phi_j(t_j) = I$. Provided we make $L\left|t_{j+1} - t_j\right|$ modest, the condition number of each $\Phi(t_{j+1})$ should also be modest, and the overall system should not be ill-conditioned. The overall linear system to be solved for each step of Newton's method for solving $\boldsymbol{g}(\boldsymbol{x}(a), \boldsymbol{x}(b)) = \boldsymbol{0}$ is

(6.2.12)

$$\begin{bmatrix} -\nabla \boldsymbol{S}_0(\boldsymbol{x}_0) & I & & & & \\ & -\nabla \boldsymbol{S}_1(\boldsymbol{x}_1) & I & & & \\ & & \ddots & \ddots & & \\ & & & -\nabla \boldsymbol{S}_{m-1}(\boldsymbol{x}_{m-1}) & I & \\ \nabla_{\boldsymbol{x}_0} \boldsymbol{g}(\boldsymbol{x}_0, \boldsymbol{x}_m) & & & & \nabla_{\boldsymbol{x}_m} \boldsymbol{g}(\boldsymbol{x}_0, \boldsymbol{x}_m) \end{bmatrix} \begin{bmatrix} \delta \boldsymbol{x}_0 \\ \delta \boldsymbol{x}_1 \\ \delta \boldsymbol{x}_2 \\ \vdots \\ \delta \boldsymbol{x}_{m-1} \\ \delta \boldsymbol{x}_m \end{bmatrix}$$

$$= - \begin{bmatrix} \boldsymbol{x}_1 - \boldsymbol{x}_0 \\ \boldsymbol{x}_2 - \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_{m-2} - \boldsymbol{x}_{m-1} \\ \boldsymbol{x}_{m-1} - \boldsymbol{x}_m \\ \boldsymbol{g}(\boldsymbol{x}_0, \boldsymbol{x}_m) \end{bmatrix}.$$

Ill-conditioning can still occur, but then it will be inherent in the problem, not an artifact of the shooting method. Also, the multiple shooting matrix is relatively sparse, so block-sparse matrix techniques can be used. For example, we can apply a block LU factorization to the matrix in (6.2.12), utilizing the block sparsity of the matrix. If $\boldsymbol{x}(t) \in \mathbb{R}^n$ then (6.2.12) can be solved in $\mathcal{O}(m\, n^3)$ operations. Of course, LU factorization without pivoting can be numerically unstable. On the other hand, a block QR factorization can be performed in $\mathcal{O}(m\, n^3)$ operations without the risk of numerical instability, and the block sparsity of the matrix is still preserved.

### 6.2.3 *Finite Difference Approximations*

Another approach to boundary value problems for ordinary differential equations, is to directly approximate the derivatives. This is particularly useful for second-order differential equations, such as the equations for diffusion (6.2.3). The basic idea is to approximate $d^2y/dx^2$ with the finite difference approximation $(y(x+h) - 2\,y(x) + y(x-h))/h^2 = d^2y/dx^2(x) + \mathcal{O}(h^2)$. If we use equally spaced points $x_j = a + j\,h$, $n\,h = L$, then (6.2.3)

$$D\frac{d^2c}{dx^2} = b\,c \qquad \text{becomes}$$

$$D\frac{c_{j+1} - 2\,c_j + c_{j-1}}{h^2} = b\,c_j, \qquad j = 1, 2, \ldots, n-1. \tag{6.2.13}$$

We need boundary conditions for the two end points, and these are $c(L) = c_{\text{end}}$ and $dc/dx(0) = 0$. Discretizing these in the obvious way, we get $c_n = c_{\text{end}}$ and $(c_1 - c_0)/h = 0$. Note that the second equation uses the one-sided difference approximation. Using the centered difference approximation is not useful here as that would require $c_{-1}$, which is not available. This gives a linear system

$$\begin{bmatrix} -1/h & +1/h & & & \\ D/h^2 & -2D/h^2 + b & D/h^2 & & \\ & D/h^2 & -2D/h^2 + b & \ddots & \\ & & \ddots & \ddots & D/h^2 \\ & & & D/h^2 & -2D/h^2 + b \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ -D\,c_{\text{end}}/h^2 \end{bmatrix}.$$

Multiplying the first row by $D/h$ gives a symmetric matrix $A_h$. Provided $D, b > 0$, $-A_h$ is also positive definite. To see that $-A_h$ is positive definite, note that

$$-\boldsymbol{c}^T A_h \boldsymbol{c} = (D/h^2)\left[\sum_{j=0}^{n-2}(c_{j+1} - c_j)^2 + c_{n-1}^2\right] + b\sum_{j=1}^{n-1} c_j^2.$$

The condition number of $A_h$ is $\mathcal{O}(h^{-2})$. There is also the bound $\left\|A_h^{-1}\right\|_2 \le 4L^2/(\pi^2 D)$ for all $h$. Solving this linear system can be done using standard sparse matrix techniques.

**Exercises.**

(1) Implement the variational equation solver for (6.2.7, 6.2.8). That is, minimize the discrete approximation

$$\sum_{k=0}^{n-1}\sqrt{1 + \left(\frac{y_{k+1} - y_k}{h}\right)^2}\, 2\pi\frac{y_{k+1} + y_k}{2}\, h$$

of the integral $\int_a^b \sqrt{1 + (dy/dx)^2}\, 2\pi y\, dx$ subject to the boundary conditions that $y_0 = y_n$ is fixed. As usual, $h = (b-a)/n$. Use any available optimization methods and software. Do this for $n = 2^k$, $k = 1, 2, \ldots, 10$, and $y_0 = 1.6$. Plot the maximum error in the solution against $h$; the exact solution has the form $y(x) = (1/\beta)\cosh(\beta x)$ where $(1/\beta)\cosh(\beta) = y_0$.

(2) Consider the problem of finding periodic orbits $\boldsymbol{x}(T;\boldsymbol{a}) = \boldsymbol{a}$ of an autonomous differential equation: $d\boldsymbol{x}/dt = \boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{x}(0;\boldsymbol{a}) = \boldsymbol{a}$. Show that if $\boldsymbol{x}(T_0;\boldsymbol{a}_0) = \boldsymbol{a}_0$ then $\nabla_{\boldsymbol{a}}\boldsymbol{x}(T_0;\boldsymbol{a}_0)\,\boldsymbol{f}(\boldsymbol{a}_0) = \boldsymbol{f}(\boldsymbol{a}_0)$. From this, show that $\nabla_{\boldsymbol{a}}\,[\boldsymbol{x}(T_0;\boldsymbol{a}) - \boldsymbol{a}]$ is not invertible at $\boldsymbol{a} = \boldsymbol{a}_0$.

(3) Continuing the previous Exercise, note that the period $T$ is really an additional variable, and so we should add an additional equation. If $\boldsymbol{n}^T\boldsymbol{f}(\boldsymbol{a}_0) \neq 0$, we can solve the equations

$$\begin{aligned} \boldsymbol{0} &= \boldsymbol{x}(T;\boldsymbol{a}) - \boldsymbol{a} \\ c &= \boldsymbol{n}^T\boldsymbol{a} \end{aligned}$$

for computing both $\boldsymbol{a}$ and $T$, using a convenient value of $c$. Using the variational equation (6.2.10), implement a Newton method for finding $(\boldsymbol{a}, T)$ generating periodic orbits. Apply this to finding periodic orbits for the *van der Pol equation*

$$\frac{d^2v}{dt^2} - \mu(1 - v^2)\frac{dv}{dt} + v = 0 \tag{6.2.14}$$

with $\mu = 2$. Since the periodic orbits of the van der Pol equations are limit cycles, solving the van der Pol equation forward in time can give good starting points for the Newton method.

(4) The Blasius problem is related to the asymptotics of viscous fluid flow over a plate, and is the third-order differential equation $y''' + \frac{1}{2}y\,y'' = 0$ with boundary conditions $y(0) = y'(0) = 0$ and $y'(\infty) = 1$. The boundary condition "at infinity," $y'(\infty) = 1$, can be approximated by $y'(L) = 1$ with $L$ large. Use a shooting method to solve the Blasius problem (the value of $y''(0)$ is the quantity to solve for). Choose different values of $L$ and discuss the convergence of the solution as $L$ becomes large.

(5) ⚠ In this Exercise, we look at a method for finding *geodesics* on a surface; that is, curves of minimal length between two points on the surface. Suppose that a surface is given by a scalar equation $g(\boldsymbol{x}) = 0$. This can be represented by the constrained optimization problem:

$$\begin{aligned} &\text{minimize} \int_0^1 \frac{1}{2}\left\|\frac{d\boldsymbol{x}}{dt}(t)\right\|_2^2 dt \\ &\text{subject to} \quad g(\boldsymbol{x}(t)) = 0 \quad \text{for all } t \\ &\qquad\qquad \boldsymbol{x}(0) = \boldsymbol{x}_0, \quad \boldsymbol{x}(1) = \boldsymbol{x}_1. \end{aligned} \tag{6.2.15}$$

Using a Lagrange multiplier approach with Lagrange multiplier $\lambda(t)$ for the constraint $g(\boldsymbol{x}(t)) = 0$, we obtain

$$\frac{d^2\boldsymbol{x}}{dt^2} + \lambda(t)\nabla g(\boldsymbol{x}(t)) = \boldsymbol{0}.$$

By differentiating $g(\boldsymbol{x}(t)) = 0$ twice, show that

$$0 = \nabla g(\boldsymbol{x}(t))^T \frac{d^2\boldsymbol{x}}{dt^2}(t) + \frac{d\boldsymbol{x}}{dt}(t)^T \operatorname{Hess} g(\boldsymbol{x}(t)) \frac{d\boldsymbol{x}}{dt}(t).$$

Use this to obtain an equation for $\lambda(t)$ in terms of $\nabla g(\boldsymbol{x}(t))$, $\operatorname{Hess} g(\boldsymbol{x}(t))$, and $d\boldsymbol{x}/dt(t)$. Develop a shooting method for finding $\boldsymbol{v}_0 = \boldsymbol{v}(0) = d\boldsymbol{x}/dt(0)$ to hit the target $\boldsymbol{x}(1) = \boldsymbol{x}_1$. The differential equation solver can enforce the constraints $g(\boldsymbol{x}(t)) = 0$ and $\nabla g(\boldsymbol{x}(t))^T (d\boldsymbol{x}/dt)(t) = 0$ at the end of each step to prevent "drift" away from the surface. Apply this to the problem of finding geodesics on an ellipse $(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$. The special case of $a = b = c$ is a sphere, and geodesics on a sphere are "great circles" (circles on the sphere that are centered at the center of the sphere).

## 6.3 Partial Differential Equations—Elliptic Problems

Partial differential equations come in a number of different essential types, which are best exemplified in two spatial dimensions below:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \qquad \text{(Poisson equation)} \tag{6.3.1}$$

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(t, x, y) \qquad \text{(Diffusion equation)} \tag{6.3.2}$$

$$\frac{\partial^2 u}{\partial t^2} = c^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(t, x, y) \qquad \text{(Wave equation)} \tag{6.3.3}$$

The *Poisson* equation is an example of an *elliptic* partial differential equation; the *diffusion* (or *heat*) equation is an example of a *parabolic* partial differential equation; while the *wave* equation is an example of a *hyperbolic* partial differential equation.

To understand the difference between these different types, consider $u(x, y) = \exp(i(k_x x + k_y y))$ for the Poisson equation, and $u(t, x, y) = \exp(i(k_t t + k_x x + k_y y))$ for the diffusion and wave equations. The corresponding $f(x, y)$ and $f(t, x, y)$ that gives these solutions are

$$\begin{aligned}
f(x, y) &= -(k_x^2 + k_y^2)\ \exp(i(k_x x + k_y y)) && \text{for Poisson equation,}\\
f(t, x, y) &= (ik_t + D(k_x^2 + k_y^2))\ \exp(i(k_t t + k_x x + k_y y)) && \text{for diffusion equation, and}\\
f(t, x, y) &= (-k_t^2 + c^2(k_x^2 + k_y^2))\ \exp(i(k_t t + k_x x + k_y y)) && \text{for the wave equation.}
\end{aligned}$$

The wave equation is different from the others as if $k_t^2 = c^2(k_x^2 + k_y^2)$ then $f(t, x, y) = 0$. This means that information can travel in the direction $\boldsymbol{k} = (k_t, k_x, k_y)$ in the solution $u(t, x, y)$ even with $f(t, x, y) = 0$ for all $(t, x, y)$.

For the diffusion equation, we get $k_t$ imaginary for $k_x$ and $k_y$ real: $k_t = iD(k_x^2 + k_y^2)$ so $\exp(i(k_t t + k_x x + k_y y)) = \exp(-D(k_x^2 + k_y^2)\, t + i(k_x x + k_y y))$ which decays exponentially as $t$ increases. This means that high frequency components of $u(t, x, y)$ decay rapidly as $t$ increases. Flipping the sign of $\partial u/\partial t$ changes rapid exponential decay to rapid exponential growth, which is very undesirable. So the sign of $\partial u/\partial t$ is very important for diffusion equations.

For the Poisson equation, if $\boldsymbol{k} = (k_x, k_y) \neq \mathbf{0}$ a component of the solution $u(x, y)$ of the form $\exp(i(k_x x + k_y y))$ must be reflected in $f(x, y)$. Furthermore, the coefficient of $\exp(i(k_x x + k_y y))$ is $-1/(k_x^2 + k_y^2)$ times the coefficient of $\exp(i(k_x x + k_y y))$ in $f(x, y)$. So the coefficient of a high frequency component ($(k_x, k_y)$ large) in the solution is much less than the corresponding coefficient of $f(x, y)$. Thus the solution $u(x, y)$ is generally much smoother than $f(x, y)$.

This classification can be made more sophisticated and more precise through Fourier transforms. More details can be found in [213, 245], for example.

In this section we will focus on equations like the Poisson equation, called elliptic equations. Elliptic equations are linear partial differential equations $A\,u(\boldsymbol{x}) = f(\boldsymbol{x})$ where $u(\boldsymbol{x}) = \exp(i\boldsymbol{k}^T\boldsymbol{x})$ gives $A\,u(\boldsymbol{x}) = s(\boldsymbol{x}, \boldsymbol{k})\,u(\boldsymbol{x})$ and for each $\boldsymbol{x}$, $|s(\boldsymbol{x}, \boldsymbol{k})| \to \infty$ as $\|\boldsymbol{k}\| \to \infty$. The function $s(\boldsymbol{x}, \boldsymbol{k})$ is called the *symbol* of the partial differential equation, and contains valuable information about the equation beyond just its classification.

It can be very helpful to use the operators and theorems of vector calculus to work with partial differential equations. In particular we use the *divergence operator* of a vector function:

$$\operatorname{div} \boldsymbol{v}(\boldsymbol{x}) = \sum_{j=1}^{n} \frac{\partial v_j}{\partial x_j}, \tag{6.3.4}$$

the *gradient operator* of a scalar function

$$\nabla u(\boldsymbol{x}) = [\frac{\partial u}{\partial x_1}(\boldsymbol{x}),\ \frac{\partial u}{\partial x_2}(\boldsymbol{x}), \ldots,\ \frac{\partial u}{\partial x_n}(\boldsymbol{x})]^T,$$

the *Jacobian operator* of a vector function,

$$\nabla \boldsymbol{v}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial v_1}{\partial x_1}(\boldsymbol{x}) & \frac{\partial v_1}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial v_1}{\partial x_n}(\boldsymbol{x}) \\ \frac{\partial v_2}{\partial x_1}(\boldsymbol{x}) & \frac{\partial v_2}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial v_2}{\partial x_n}(\boldsymbol{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial v_m}{\partial x_1}(\boldsymbol{x}) & \frac{\partial v_m}{\partial x_2}(\boldsymbol{x}) & \cdots & \frac{\partial v_m}{\partial x_n}(\boldsymbol{x}) \end{bmatrix},$$

and the *curl* or *rotation operator* for functions $\boldsymbol{v}(\boldsymbol{x}) \in \mathbb{R}^3$ with $\boldsymbol{x} \in \mathbb{R}^3$,

$$\nabla \times \boldsymbol{v}(\boldsymbol{x}) = \left[ \frac{\partial v_3}{\partial x_2}(\boldsymbol{x}) - \frac{\partial v_2}{\partial x_3}(\boldsymbol{x}),\ \frac{\partial v_1}{\partial x_3}(\boldsymbol{x}) - \frac{\partial v_3}{\partial x_1}(\boldsymbol{x}),\ \frac{\partial v_2}{\partial x_1}(\boldsymbol{x}) - \frac{\partial v_1}{\partial x_2}(\boldsymbol{x}) \right]^T.$$

The most important theorem of vector calculus is the *divergence theorem*: for any region $\Omega \subset \mathbb{R}^n$ that is bounded and has a piecewise smooth boundary, and $\boldsymbol{v}$ continuously differentiable,

$$\int_\Omega \operatorname{div} \boldsymbol{v}(\boldsymbol{x})\, d\boldsymbol{x} = \int_{\partial\Omega} \boldsymbol{v}(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})\, dS(\boldsymbol{x}) \tag{6.3.5}$$

where $\partial\Omega$ is the boundary of $\Omega$, $\boldsymbol{n}(\boldsymbol{x})$ is the outward pointing normal vector (perpendicular to the boundary at $\boldsymbol{x}$), and $dS(\boldsymbol{x})$ indicates integration over the surface area of $\partial\Omega$.

The equations given above can be easily represented in terms of these operators:

$$\begin{aligned} \operatorname{div} \nabla u &= f(\boldsymbol{x}), && \text{Poisson equation,} \\ \frac{\partial u}{\partial t} &= D \operatorname{div} \nabla u(\boldsymbol{x}) + f(t, \boldsymbol{x}), && \text{diffusion equation} \\ \frac{\partial^2 u}{\partial t^2} &= c^2 \operatorname{div} \nabla u(\boldsymbol{x}) + f(t, \boldsymbol{x}), && \text{wave equation.} \end{aligned}$$

An important family of differential equations are in divergence form:

$$\operatorname{div} \boldsymbol{F}(\boldsymbol{x}, u(\boldsymbol{x}), \nabla u(\boldsymbol{x})) = f(\boldsymbol{x}). \tag{6.3.6}$$

Equations in this form can be thought of as representing a physical situation where $\boldsymbol{F}(\boldsymbol{x}, u(\boldsymbol{x}), \nabla u(\boldsymbol{x}))$ is a flux, or flow, vector of a conserved quantity $\phi(\boldsymbol{x})$. This insight can be useful for designing numerical methods, especially where the conserved quantity should be conserved by the numerical method.

### 6.3.1 Finite Difference Approximations

We start with the Poisson equation in two dimensions (6.3.1) in a region $\Omega$:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y) \qquad \text{for } (x, y) \in \Omega.$$

For simplicity we suppose that we have Dirichlet boundary conditions: $u(x, y) = g(x, y)$ for any $(x.y) \in \partial\Omega$, the boundary of $\Omega$ where $g(x, y)$ is a given function. Supposing that $\Omega \subseteq [a, b] \times [c, d]$ we set $x_i = a + i\,h$ and $y_j = c + j\,h$ where $h > 0$ is a common fixed spacing. We can use the approximation of the second derivative

$$\frac{\partial^2 u}{\partial x^2}(x, y) = \frac{u(x+h, y) - 2\,u(x, y) - u(x-h, y)}{h^2} + \mathcal{O}(h^2)$$

to give an approximation to the left-hand side:

$$\frac{u(x+h, y) + u(x, y+h) - 4\,u(x, y) + u(x-h, y) - u(x, y-h)}{h^2} = f(x, y) + \mathcal{O}(h^2).$$

Setting $x = x_i$ and $y = y_j$ gives

$$\frac{u(x_{i+1}, y_j) + u(x_i, y_{j+1}) - 4\,u(x_i, y_j) + u(x_{i-1}, y_j) - u(x_i, y_{j-1})}{h^2} = f(x_i, y_j) + \mathcal{O}(h^2).$$

Using the computed quantities $u_{ij} \approx u(x_i, y_j)$ we have

(6.3.7)
$$\frac{u_{i+1,j} + u_{i,j+1} - 4\,u_{ij} + u_{i-1,j} - u_{i,j-1}}{h^2} = f(x_i, y_j) \qquad \text{for } (x_i, y_j) \in \Omega.$$

If $(x_i, y_j) \notin \Omega$ we set $u_{ij} = g(x_i, y_j)$.

If we set $\Omega_h = \{\, (i, j) \mid (x_i, y_j) \in \Omega \,\}$ to be the discrete domain, then we can consider our unknowns $u_{i,j}$ for $(i, j) \in \Omega_h$ as a vector $\boldsymbol{u}_h \in \mathbb{R}^{\Omega_h}$ with indexes $(i, j) \in \Omega_h$. Then the system of equations (6.3.7) is represented in matrix–vector form as $A_h \boldsymbol{u}_h = \boldsymbol{f}_h - \boldsymbol{k}_h$ where $(\boldsymbol{f}_h)_{(i,j)} = f(x_i, y_j)$, and

$$(\boldsymbol{k}_h)_{(i,j)} = \frac{\widehat{g}_{i+1,j} + \widehat{g}_{i,j+1} - 4\,\widehat{g}_{i,j} + \widehat{g}_{i-1,j} + \widehat{g}_{i,j-1}}{h^2}, \quad \text{where}$$
$$\widehat{g}_{i,j} = \begin{cases} g(x_i, y_j), & \text{if } (i, j) \notin \Omega_h, \\ 0, & \text{if } (i, j) \in \Omega_h. \end{cases}$$

This allows us to incorporate the boundary conditions into the linear system.

The matrix $A_h$ is given by the formula below for $(i, j),\ (k, \ell) \in \Omega_h$,

$$(A_h)_{(i,j),(k,\ell)} = \begin{cases} -4, & \text{if } (i, j) = (k, \ell), \\ +1, & \text{if } (i, j) = (k, \ell \pm 1) \text{ or } (i, j) = (k \pm 1, \ell), \\ 0, & \text{otherwise.} \end{cases}$$

It is easy to check that $A_h$ is symmetric (that is, $(A_h)_{(i,j),(k,\ell)} = (A_h)_{(k,\ell),(i,j)}$ for all $(i, j),\ (k, \ell) \in \Omega_h$). Also, $-A_h$ is positive definite as

$$-z_h^T A_h z_h = \frac{1}{2} \sum_{(i,j) \in \Omega_h} \Big[ \big(z_{i,j} - z_{i+1,j}\big)^2 + \big(z_{i,j} - z_{i-1,j}\big)^2 + \big(z_{i,j} - z_{i,j+1}\big)^2 + \big(z_{i,j} - z_{i,j-1}\big)^2 \Big],$$

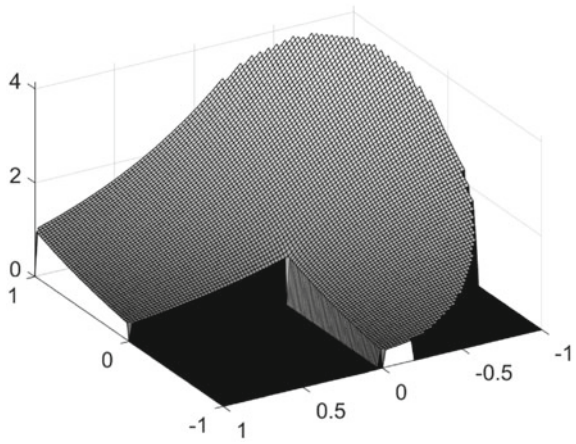



**Fig. 6.3.1** Solution to Poisson equation

taking the value of $z_{ij} = 0$ for $(i, j) \notin \Omega_h$. The value of $-\boldsymbol{z}_h^T A_h \boldsymbol{z}_h \geq 0$ as it is a sum of squares. Also $\boldsymbol{z}_h^T A_h \boldsymbol{z}_h = 0$ implies that for all $(i, j) \in \Omega_h$ we have $z_{i,j} = z_{i\pm1,j} = z_{i,j\pm1}$. Because $\Omega_h$ is finite, we must therefore have $z_{i,j} = 0$ for all $(i, j)$ in this case. Thus $-A_h$ is symmetric positive definite.

Results for the region $\Omega = \{ (x, y) \mid (x^2 + y^2 < 1 \text{ and } (x > 0 \text{ or } y > 0)) \text{ or } 0 < x, y < 1 \}$ with $u(x, y) = \exp(x - y)$ on $\partial\Omega$, $h = 1/50$, and $f(x, y) = 1$ are shown as a three-dimensional plot in Figure 6.3.1.

### 6.3.1.1 Convergence Proof for Poisson Equation

Convergence can be proven for these finite difference methods as $h \downarrow 0$. Note that if $\boldsymbol{u}$ is the vector in $\mathbb{R}^{\Omega_h}$ given by $(\boldsymbol{u})_{i,j} = u(x_i, y_j)$ for the exact solution, then

$$\begin{aligned} A_h \boldsymbol{u}_h &= \boldsymbol{f}_h - \boldsymbol{k}_h, \\ A_h \boldsymbol{u} &= \boldsymbol{f}_h - \boldsymbol{k}_h + \boldsymbol{\eta}_h \qquad \text{where} \\ (\boldsymbol{\eta})_{i,j} &= (A_h \boldsymbol{u})_{i,j} - \left( \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) (x_i, y_j) = \mathcal{O}(h^2). \end{aligned}$$

The hidden constant in $\mathcal{O}(h^2)$ depends on the fourth derivatives of $u$. Subtracting the equations for $\boldsymbol{u}$ and $\boldsymbol{u}_h$ gives

$$A_h(\boldsymbol{u} - \boldsymbol{u}_h) = \boldsymbol{\eta}_h.$$

We can see from the above calculations that $\|\boldsymbol{\eta}_h\|_\infty = \mathcal{O}(h^2)$ provided $u$ has continuous fourth derivatives. Our task then is to show that $\|A_h^{-1}\|_\infty$ is bounded, independently of the grid spacing $h$ as $h \downarrow 0$.

First, we show that the entries of $-A_h^{-1}$ are non-negative. We can write $A_h = h^{-2}(-4\,I + E_h)$ where

$$(E_h)_{(i,j),(k,\ell)} = \begin{cases} 1, & \text{if } (i,j) = (k \pm 1, \ell) \text{ or } (i,j) = (k, \ell \pm 1) \\ 0, & \text{otherwise,} \end{cases}$$

for $(i, j)$, $(k, \ell) \in \Omega_h$. If we consider solving the equation $-A_h \boldsymbol{w} = \boldsymbol{b}$ with $\boldsymbol{b} \geq \boldsymbol{0}$ in the sense that $b_{(i,j)} \geq 0$ for all $(i, j) \in \Omega_h$, we can express this in the form $4\boldsymbol{w} = h^2\,\boldsymbol{b} + E_h \boldsymbol{w}$. This can be solved by the iterative method $\boldsymbol{w}^{(k+1)} = (h/2)^2 \boldsymbol{b} + \frac{1}{4} E_h \boldsymbol{w}^{(k)}$. This is a convergent iteration provided $\rho(E_h) < 4$. Since every entry of $E_h$ is zero or one, and there are at most four non-zero entries per row, $\rho(E_h) \leq \|E_h\|_\infty \leq 4$ by Theorem 2.16. We now show that $\rho(E_h) \neq 4$. Since $E_h$ is a real symmetric matrix, it has real eigenvalues. So we just need to show that neither $+4$ nor $-4$ are eigenvalues of $E_h$. Suppose $E_h \boldsymbol{\xi} = \pm 4 \boldsymbol{\xi}$, $\boldsymbol{\xi} \neq \boldsymbol{0}$. Choose $(i^*, j^*) \in \Omega_h$ where $\left|\xi_{(i,j)}\right| = \max_{(i,j) \in \Omega_h} \left|\xi_{(i,j)}\right|$. Then, taking $\xi_{(k,\ell)} = 0$ if $(k, \ell) \notin \Omega_h$, we have

$$\begin{aligned} \pm 4\,\xi_{(i^*,j^*)} &= \xi_{(i^*+1,j^*)} + \xi_{(i^*-1,j^*)} + \xi_{(i^*,j^*+1)} + \xi_{(i^*,j^*-1)}, \qquad \text{so} \\ 4\,\left|\xi_{(i^*,j^*)}\right| &\leq \left|\xi_{(i^*+1,j^*)}\right| + \left|\xi_{(i^*-1,j^*)}\right| + \left|\xi_{(i^*,j^*+1)}\right| + \left|\xi_{(i^*,j^*-1)}\right| \\ &\leq 4\,\left|\xi_{(i^*,j^*)}\right|. \end{aligned}$$

This can only occur if $\xi_{(i^*,j^*)} = \xi_{(i^*+1,j^*)} = \xi_{(i^*-1,j^*)} = \xi_{(i^*,j^*+1)} = \xi_{(i^*,j^*-1)}$. Applying the above argument to $(i^* + 1, j^*)$, $(i^* - 1, j^*)$, $(i^*, j^* + 1)$, and $(i^*, j^* - 1)$, we can see that as long as $\Omega_h$ is connected we have $\xi_{(k,\ell)} = \xi_{(i^*,j^*)}$ for all $(k, \ell) \in \Omega_h$. Eventually we will have $(i^*, j^*) \in \Omega_h$ but there is some $(i^* \pm 1, j^*)$ or $(i^*, j^* \pm 1)$ that is not in $\Omega_h$, and we will have $\xi_{(i^*,j^*)} = 0$. This means that $\boldsymbol{\xi} = \boldsymbol{0}$ and $\boldsymbol{\xi}$ would not then be an eigenvector of $E_h$. Thus $\rho(E_h) < 4$. Then by Theorem 2.15, the iteration $\boldsymbol{w}^{(k+1)} = (h/2)^2 \boldsymbol{b} + \frac{1}{4} E_h \boldsymbol{w}^{(k)}$ converges to a solution of $-A_h \boldsymbol{w} = \boldsymbol{b}$:

$$\boldsymbol{w} = \left[ I + \frac{1}{4} E_h + \left(\frac{1}{4} E_h\right)^2 + \left(\frac{1}{4} E_h\right)^3 + \cdots \right] \frac{h^2}{2} \boldsymbol{b} = -A_h^{-1} \boldsymbol{b}.$$

Since each term of the infinite sum for $A_h^{-1}$ is a matrix with non-negative entries, $-A_h^{-1}$ is a matrix with non-negative entries.

We want to obtain a bound on $\left\|A_h^{-1}\right\|_\infty$ based on the diameter of $\Omega$ and the grid spacing $h$. To do this, we compare $\left\|A_h^{-1}\right\|_\infty$ for $\Omega$ with $\left\|(A_h')^{-1}\right\|_\infty$ for a larger region $\Omega'$ where this can be computed explicitly.

Consider expanding the discrete domain $\Omega_h \subset \Omega_h'$: If $A_h'$ is the matrix for $\Omega_h'$ and $A_h$ the matrix for $\Omega_h$, then

$$\begin{aligned} A_h' &= \frac{1}{h^2}(-4\,I + E_h') \qquad \text{and} \\ A_h &= \frac{1}{h^2}(-4\,I + E_h) \qquad \text{where} \end{aligned}$$

$$E'_h = \begin{bmatrix} E_h & (E'_h)_{12} \\ (E'_h)_{12}^T & (E'_h)_{22} \end{bmatrix}$$

if we order the rows and columns for $(i, j) \in \Omega_h$ before the rows and columns for $(i, j) \in \Omega'_h \backslash \Omega_h = \left\{ (i, j) \mid (i, j) \in \Omega'_h \text{ but } (i, j) \notin \Omega_h \right\}$. Since all these $E$ matrices have non-negative entries,

$$E'_h \geq \begin{bmatrix} E_h & 0 \\ 0 & 0 \end{bmatrix} \quad \text{entry-wise.}$$

This means that

$$\begin{aligned} -\left(A'_h\right)^{-1} &= \left[ I + \frac{1}{4}E'_h + \left(\frac{1}{4}E'_h\right)^2 + \left(\frac{1}{4}E'_h\right)^3 + \cdots \right] \frac{h^2}{2} \\ &\geq \begin{bmatrix} \left[ I + \frac{1}{4}E_h + \left(\frac{1}{4}E_h\right)^2 + \left(\frac{1}{4}E_h\right)^3 + \cdots \right] \frac{h^2}{2} & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} -A_h^{-1} & 0 \\ 0 & 0 \end{bmatrix} \quad \text{entry-wise.} \end{aligned}$$

So $\left\| (A'_h)^{-1} \right\|_\infty \geq \left\| A_h^{-1} \right\|_\infty$. We can now look for a suitable expansion $\Omega'_h$ of $\Omega_h$ for which we can compute a bound of $\left\| (A'_h)^{-1} \right\|_\infty$ that is independent of $h > 0$.

We can take $\Omega'_h$ to be essentially a circle: $\Omega'_h = \left\{ (i, j) \mid i^2 + j^2 \leq N^2 \right\}$ and set $R = h(N + 1)$. Since $-A'_h$ has only non-negative entries, $\left\| (A'_h)^{-1} \right\|_\infty = \left\| (A'_h)^{-1} \boldsymbol{e} \right\|_\infty$ where $\boldsymbol{e}$ is the vector of ones of the appropriate size. The vector $(A_h)^{-1}\boldsymbol{e}$ is the solution of the discrete Poisson equations (6.3.7) for the square $\Omega'$ with $f(x, y) = 1$ for all $(x, y)$. In fact, if $\boldsymbol{b} \geq \boldsymbol{e}$ entry-wise, then $\left\| (A'_h)^{-1} \right\|_\infty \leq \left\| (A'_h)^{-1} \boldsymbol{b} \right\|_\infty$. The finite difference approximation is exact for quadratic functions. Let $w(x, y) = R^2 - x^2 - y^2$ and $w_{(i,j)} = w(x_i, y_j)$. Then provided $(i, j), (i \pm 1, j)$, and $(i, j \pm 1)$ all belong to $\Omega'_h$,

$$-\frac{w_{(i+1,j)} + w_{(i,j+1)} - 4w_{(i,j)} + w_{(i-1,j)} + w_{(i,j-1)}}{h^2} = -\frac{\partial^2 w}{\partial x^2}(x_i, y_j) - \frac{\partial^2 w}{\partial y^2}(x_i, y_j) = 4.$$

If $(i, j) \in \Omega'_h$ but some neighbor $(i \pm 1, j)$ or $(i, j \pm 1)$ is not, then because $i^2 + j^2 \leq N^2$, $(i \pm 1)^2 + j^2 \leq N^2 + 2|i| + 1 \leq (N + 1)^2$, it follows that $w_{(i\pm1,j)} \geq 0$. Similarly we can show that $w_{(i,j\pm1)} \geq 0$. In such a case $(-A_h \boldsymbol{w})_{(i,j)} \geq 4$. So $-A_h \boldsymbol{w} \geq 4\boldsymbol{e}$ entry-wise. Setting $\boldsymbol{b} = (-A_h \boldsymbol{w})/4$, we see that $\boldsymbol{b} \geq \boldsymbol{e}$ entry-wise. Then

$$\left\| (A'_h)^{-1} \right\|_\infty \leq \left\| (A'_h)^{-1} \boldsymbol{b} \right\|_\infty = \| -\boldsymbol{w}/4 \|_\infty \leq R^2/4.$$

This bound is clearly independent of $h$, and so

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_\infty \le \left\|(A_h')^{-1}\right\|_\infty \|\boldsymbol{\eta}_h\|_\infty \le \frac{1}{4}R^2 \left( \left\| \frac{\partial^4 u}{\partial x^4} \right\|_\infty + \left\| \frac{\partial^4 u}{\partial y^4} \right\|_\infty \right) h^2.$$

Not only have we shown convergence, but we have proved the rate of convergence to match the consistency order of the method, at least provided the exact solution is smooth. This is not always the case for interior corners that "cut into" the region, such as the origin in Figure 6.3.1.

There are many variants of the Poisson equation, and the methods of showing convergence can be modified to deal with these situations. The hardest part of these proofs is showing that the numerical method is stable in a suitable sense. For the Poisson equation, it comes down to showing that $\left\|A_h^{-1}\right\|_\infty$ is bounded independent of $h > 0$. The method of proof relies on the fact that $-A_h^{-1}$ has only non-negative entries, which can be traced back to the corresponding property of the Poisson equation: if $-(\partial^2 u/\partial x^2 + \partial^2 u/\partial y^2) \ge 0$ in $\Omega$ with $u(x, y) = 0$ on $\partial\Omega$, then $u(x, y) \ge 0$ for all $(x, y) \in \Omega$.

### 6.3.1.2 Conservation Principles

Many partial differential equations come from physical problems or can be understood as modeling some physical process. Typically, some quantities are conserved, like energy, mass, and momentum. Conservation of a quantity does not necessarily mean that the total amount of that quantity remains fixed. But, it does mean that we can identify net production (or consumption) of that quantity as a function of time and space. For the Poisson equation in two dimensions, and its variants, consider the diffusion equation

$$\frac{\partial u}{\partial t} = D\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + f(x, y) \qquad \text{in } \Omega, \tag{6.3.8}$$

$$u(x, y) = g(x, y) \qquad \text{for } (x, y) \in \partial\Omega. \tag{6.3.9}$$

Here $u(t, x, y)$ represents the concentration of a certain chemical species, for example. Here $D$ is the diffusion constant for the chemical species in whatever medium it is diffusing in. The function $f(x, y)$ represents the net rate of production of this chemical species per unit area per unit time.

Re-writing the diffusion equation (6.3.8) for constant $D$ as

$$\frac{\partial u}{\partial t} = \operatorname{div} D\,\nabla u + f(\boldsymbol{x}), \tag{6.3.10}$$

we can identify $D\,\nabla u$ as the rate of flow, or *flux vector*, for $u$. Note that (6.3.10) is in divergence form. The rate of flow is $\boldsymbol{n}^T D\,\nabla u$ per unit time per unit length or area of the boundary, where $\boldsymbol{n}$ is the unit vector perpendicular to the boundary. If we divide the region $\Omega$ into cells as shown in Figure 6.3.2, then the net rate of outflow from

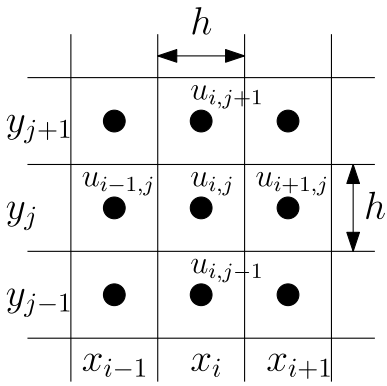



**Fig. 6.3.2** Finite volume cells

the central cell is

$$D\frac{u_{i,j}-u_{i-1,j}}{h}\,h + D\frac{u_{i,j}-u_{i+1,j}}{h} + D\frac{u_{i,j}-u_{i,j-1}}{h} + D\frac{u_{i,j}-u_{i,j+1}}{h}.$$

This can be equated to $-(d/dt)(h^2\,u_{i,j}) + h^2\,f(x_i, y_j)$ as $h^2 u_{i,j}$. This gives the equation

$$\frac{d}{dt}(h^2\,u_{i,j}) = D\left[u_{i+1,j} + u_{i,j+1} - 4\,u_{i,j} + u_{i-1,j} + u_{i,j-1}\right] + h^2\,f(x_i, y_j).$$

For the steady state, we set $(d/dt)(h^2\,u_{i,j}) = 0$. Then

$$D\frac{u_{i+1,j} + u_{i,j+1} - 4\,u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} = -f(x_i, y_j),$$

which is the discretization obtained by the second-order difference formula. The advantage of this formulation is in dealing with non-uniform grids, and varying diffusion coefficient $D(x, y)$. We still need the partial differential equation to be in divergence form.

As an example, suppose we keep uniform spacing of the grid points, but we have $D(x, y)$. Set $D_{i\pm1/2,\,j} = D(\frac{1}{2}(x_i + x_{i\pm1}), y_j)$ and $D_{i,\,j\pm1/2} = D(x_i, \frac{1}{2}(y_j + y_{j\pm1}))$. Then the rate of net outflow from the central cell in Figure 6.3.2 is best approximated by

$$D_{i+1/2,\,j}\frac{u_{i,j}-u_{i+1,j}}{h}h + D_{i-1/2,\,j}\frac{u_{i,j}-u_{i-1,j}}{h}h + D_{i,\,j+1/2}\frac{u_{i,j}-u_{i,j+1}}{h}h + D_{i,\,j-1/2}\frac{u_{i,j}-u_{i,j-1}}{h}h.$$

Using this approximation we obtain the discretized equations

$$\frac{1}{h^2}\left[\left(D_{i+1/2,\,j} + D_{i-1/2,\,j} + D_{i,\,j+1/2} + D_{i,\,j-1/2}\right)u_{i,j}\right.$$

$$- D_{i+1/2,j}u_{i+1,j} - D_{i,j+1/2}u_{i,j+1} - D_{i-1/2,j}u_{i-1,j} - D_{i,j-1/2}u_{i,j-1}\big]$$
$$= f(x_i, y_j).$$

If $D(x, y) > 0$ and continuous, this system has a unique solution $u_{i,j}$ if we set $u_{i,j} = g(x_i, y_j)$ for $(x_i, y_j) \notin \Omega$. This can be proven using the techniques of the previous section.

#### 6.3.1.3 Computational Issues

The system of equations for the Poisson equation is a linear system. It is a large, sparse system of equations, with no more than five non-zeros per row in two spatial dimensions. The system can be solved by either direct or iterative methods.

Since $-A_h$ is symmetric positive definite, sparse Cholesky factorization can be used to solve the system. In this case, ordering the rows and columns to reduce the amount of fill-in can be important to reduce both the amount of memory needed, and the time taken to form the factorization and to solve the system. One of the best approaches is to use nested dissection (see Section 2.3.3.2). Using nested dissection requires $\mathcal{O}(N^3)$ floating point operations and $\mathcal{O}(N^2 \log N)$ memory for an $N \times N$ grid.

Alternatively, conjugate gradients (see Section 2.4.2) provide an iterative method that avoids the cost of the memory needed to store the fill-in. The condition number of $-A_h$ is $\kappa_2(-A_h) = \mathcal{O}(h^{-2}) = \mathcal{O}(N^2)$, so the number of iterations needed to obtain an error of $\epsilon$ requires $\mathcal{O}(\sqrt{N^2}\log(1/\epsilon))$ iterations. To achieve an error in the equation solver of $\mathcal{O}(h^2) = \mathcal{O}(N^{-2})$ (to be comparable to the error for the exact solution of the *discretized* equations) thus requires $\mathcal{O}(N \log N)$ iterations. Each iteration takes $\mathcal{O}(N^2)$ floating point operations as there are $N^2$ rows in $A_h$ and each row has no more than five entries. Thus conjugate gradients *without preconditioning* takes $\mathcal{O}(N^3 \log N)$ floating operations. Preconditioning can greatly reduce the number of iterations needed for conjugate gradients. In this regard, *multigrid methods* [33, 248] arguably give near optimal performance with $\kappa_2(B_h A_n) = \mathcal{O}(1)$ where $B_h$ is a multigrid preconditioner, bringing the total cost down to $\mathcal{O}(N^2 \log N)$ floating point operations, although the hidden constant is also fairly large.

Three-dimensional problems give even more advantage to iterative methods over direct methods, even without preconditioning.

### *6.3.2 Galerkin Method*

There is another approach to solving partial differential equations that is much more flexible, based on triangulations instead of rectangular grids. The mathematical foundation is the *Galerkin method*, but this method is also known as the *finite element method*. The method is named after Boris Galerkin for his work on rods and plates

in 1915, although the mathematical underpinning was already developed by Walther Ritz in 1902. The history of the development of these methods is outlined in [99]. The method is based on reformulating the partial differential equation in its *weak form*. We take the Poisson equation as our example:

$$\begin{aligned} \operatorname{div} \nabla u(\boldsymbol{x}) &= f(\boldsymbol{x}) \qquad \text{for } \boldsymbol{x} \in \Omega, \\ u(\boldsymbol{x}) &= g(\boldsymbol{x}) \qquad \text{for } \boldsymbol{x} \in \partial\Omega. \end{aligned}$$

Then for any smooth function $v(\boldsymbol{x})$ where $v(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \partial\Omega$,

$$\int_\Omega (f(\boldsymbol{x}) - \operatorname{div} \nabla u(\boldsymbol{x}))\, v(\boldsymbol{x})\, d\boldsymbol{x} = 0.$$

Now

$$\begin{aligned} &\int_\Omega (\operatorname{div} \nabla u(\boldsymbol{x}))\, v(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad = \int_\Omega \left\{ \operatorname{div} (v(\boldsymbol{x})\, \nabla u(\boldsymbol{x})) - \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x}) \right\} d\boldsymbol{x} \\ &\qquad \text{(using } \operatorname{div}(\phi\boldsymbol{\psi}) = \nabla\phi^T\boldsymbol{\psi} + \phi \operatorname{div}\boldsymbol{\psi} \text{ for smooth } \phi \text{ and } \boldsymbol{\psi}) \\ &\quad = \int_{\partial\Omega} v(\boldsymbol{x})\, \nabla u(\boldsymbol{x}) \cdot \boldsymbol{n}(\boldsymbol{x})\, dS(\boldsymbol{x}) - \int_\Omega \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad = \int_{\partial\Omega} v(\boldsymbol{x})\, \frac{\partial u}{\partial n}(\boldsymbol{x})\, dS(\boldsymbol{x}) - \int_\Omega \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x})\, d\boldsymbol{x} \end{aligned}$$

where $\boldsymbol{n}(\boldsymbol{x})$ is the unit outward pointing normal vector at $\boldsymbol{x}$ to the boundary $\partial\Omega$. By "outward pointing" we mean that $\boldsymbol{x} + \epsilon\, \boldsymbol{n}(\boldsymbol{x}) \notin \Omega$ for any sufficiently small $\epsilon > 0$. The "normal derivative" $\partial u/\partial n(\boldsymbol{x})$ is $\boldsymbol{n}(\boldsymbol{x})^T \nabla u(\boldsymbol{x})$, the derivative of $u(\boldsymbol{x})$ in the direction $\boldsymbol{n}(\boldsymbol{x})$.

Since $v(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \partial\Omega$, $\int_{\partial\Omega} v(\boldsymbol{x})\, (\partial u/\partial n)(\boldsymbol{x})\, dS(\boldsymbol{x}) = 0$. Then

$$\int_\Omega (\operatorname{div} \nabla u(\boldsymbol{x}))\, v(\boldsymbol{x})\, d\boldsymbol{x} = -\int_\Omega \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x})\, d\boldsymbol{x}, \qquad \text{so}$$

$$\int_\Omega (f(\boldsymbol{x}) - \operatorname{div} \nabla u(\boldsymbol{x}))\, v(\boldsymbol{x})\, d\boldsymbol{x} = \int_\Omega \left[ f(\boldsymbol{x})\, v(\boldsymbol{x}) + \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x}) \right] d\boldsymbol{x} = 0.$$

The formulation

$$\int_\Omega f(\boldsymbol{x})\, v(\boldsymbol{x}) d\boldsymbol{x} = -\int_\Omega \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x})\, d\boldsymbol{x} \qquad \text{for all } v(\cdot) \tag{6.3.11}$$

where $v(\boldsymbol{x}) = 0$ on $\partial\Omega$, is called the weak form of the Poisson equation.

The Galerkin method uses finite-dimensional spaces of functions $V_h$ depending on some parameter $h > 0$ where the integrals $\int_\Omega f(\boldsymbol{x})\, v(\boldsymbol{x}) d\boldsymbol{x}$ and $\int_\Omega \nabla v(\boldsymbol{x})^T \nabla u(\boldsymbol{x})\, d\boldsymbol{x}$

are defined for every $u, v \in V_h$. These spaces $V_h$ must be subspaces of the Sobolev space $H^1(\Omega)$:

$$u \in H^1(\Omega) \quad \text{if and only if} \quad \int_\Omega \left[|u|^2 + \nabla u^T \nabla u\right] \quad \text{is finite.} \tag{6.3.12}$$

The Galerkin method is to find $u_h \in V_h$ where $u_h(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$ and

$$\int_\Omega f(\boldsymbol{x})\, v_h(\boldsymbol{x})\, d\boldsymbol{x} = -\int_\Omega \nabla v_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x})\, d\boldsymbol{x} \quad \text{for all } v_h \in V_h \tag{6.3.13}$$

$$\text{where } v_h(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in \partial\Omega, \text{ and} \tag{6.3.14}$$

$$u_h(\boldsymbol{x}) = g(\boldsymbol{x}) \quad \text{for } \boldsymbol{x} \in \partial\Omega. \tag{6.3.15}$$

Since $V_h$ is finite dimensional, there is a basis $\{\phi_1, \phi_2, \ldots, \phi_N\}$ for $V_h$. As $u_h \in V_h$, we can write $u_h = \sum_{j=1}^N u_j \phi_j$. We assume that for $i = 1, 2, \ldots, M$ we have $\phi_i(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \partial\Omega$. If $g$ is equal to a function $g_h \in V_h$ on $\partial\Omega$, then we can write $g_h = \sum_{j=M+1}^N g_j \phi_j$. So $u_h = \sum_{j=1}^M u_j \phi_j + \sum_{j=M+1}^N g_j \phi_j$, with $g_j$ given. The Galerkin equations (6.3.13), taking $v_h = \phi_i$ for $i = 1, 2, \ldots, M$ are

$$\begin{aligned}\int_\Omega f(\boldsymbol{x})\, \phi_i(\boldsymbol{x})\, d\boldsymbol{x} &= -\int_\Omega \left[\sum_{j=1}^M \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, u_j + \sum_{j=M+1}^N \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, g_j\right] d\boldsymbol{x} \\ &= -\sum_{j=1}^M u_j \int_\Omega \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, d\boldsymbol{x} - \sum_{j=M+1}^N g_j \int_\Omega \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, d\boldsymbol{x}.\end{aligned}$$

If we write $a_{ij} = -\int_\Omega \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x})\, d\boldsymbol{x}$, $b_i = \int_\Omega f(\boldsymbol{x})\, \phi_i(\boldsymbol{x})\, d\boldsymbol{x}$ and $A_h = \left[a_{ij} \mid i, j = 1, 2, \ldots, M\right]$, $\widetilde{A}_h = \left[a_{ij} \mid i = 1, 2, \ldots, M,\ j = M+1, \ldots, N\right]$ then

$$\boldsymbol{b}_h = A_h \boldsymbol{u}_h + \widetilde{A}_h \boldsymbol{g}_h$$

where $A_h$ is an $M \times M$ matrix. We will see soon that under standard conditions, $-A_h$ is positive definite, and so is invertible. We can then solve the Galerkin equations for the unknown coefficients $u_j$, $j = 1, 2, \ldots, M$:

$$\boldsymbol{u}_h = A_h^{-1}\left[\boldsymbol{b}_h - \widetilde{A}_h \boldsymbol{g}_h\right].$$

We can create these subspaces $V_h$ in different ways, although the most common is to use a triangulation $\mathcal{T}_h$, and $V_h$ is the space of piecewise polynomial functions over $\mathcal{T}_h$ of a specified degree that are continuous across the triangulation. See Section 4.3.2 for more detailed discussion of triangulations.

The boundary condition $u_h(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$ often cannot be enforced as specified if $g$ on the boundary $\partial\Omega$ is not equal to any function in $V_h$ on $\partial\Omega$. However, we can use $g_h(\boldsymbol{x})$ being a polynomial interpolant or some other approximation of $g(\boldsymbol{x})$ over the boundary $\partial\Omega_h$ of the triangulated region $\Omega_h := \bigcup_{T\in\mathcal{T}_h} T$.

With a triangulation $\mathcal{T}_h$ we can use various interpolation methods that are consistent across the triangles of the triangulation, such as piecewise linear interpolation. See Sections 4.3.1 and 4.3.2 for more examples and error estimates for interpolation over triangles and triangulations.

We need the integrals $\int_\Omega \nabla u_h^T \nabla v_h \, d\boldsymbol{x}$ to be well-defined for each $u_h,\ v_h \in V_h$. From the theory of Lebesgue integration [220], $\int_\Omega h(\boldsymbol{x}) \, d\boldsymbol{x}$ is defined if and only if $\int_\Omega |h(\boldsymbol{x})| \, d\boldsymbol{x}$ is finite. Now $\left|\boldsymbol{a}^T\boldsymbol{b}\right| \le \|\boldsymbol{a}\|_2 \, \|\boldsymbol{b}\|_2$ by the Cauchy–Schwarz inequality (A.1.4). Since $r\,s \le \frac{1}{2}(r^2 + s^2)$ for any real $r$ and $s$ (coming from $0 \le (r-s)^2$), we have $\left|\boldsymbol{a}^T\boldsymbol{b}\right| \le \frac{1}{2}\left(\|\boldsymbol{a}\|_2^2 + \|\boldsymbol{b}\|_2^2\right)$. Thus, as long as $\int_\Omega \|\nabla u_h\|_2^2 \, d\boldsymbol{x}$ and $\int_\Omega \|\nabla v_h\|_2^2 \, d\boldsymbol{x}$ are both finite, $\int_\Omega \nabla u_h^T \nabla v_h \, d\boldsymbol{x}$ is well-defined. So we can apply this approach as long as $u_h$, $v_h$ belong to the Sobolev space $H^1(\Omega)$ defined in (6.3.12) with norm given by

$$\|u\|_{H^1(\Omega)} = \left( \int_\Omega \left[ u(\boldsymbol{x})^2 + \|\nabla u(\boldsymbol{x})\|_2^2 \right] d\boldsymbol{x} \right)^{1/2}. \tag{6.3.16}$$

There is also an inner product that generates the norm:

$$(u, v)_{H^1(\Omega)} = \int_\Omega \left[ u(\boldsymbol{x})\, v(\boldsymbol{x}) + \nabla(\boldsymbol{x})^T \nabla v(\boldsymbol{x}) \right] d\boldsymbol{x}, \qquad \text{so} \tag{6.3.17}$$
$$\|u\|_{H^1(\Omega)} = \sqrt{(u, u)_{H^1(\Omega)}}.$$

Because the $H^1(\Omega)$ norm is generated by an inner product, $H^1(\Omega)$ is a *Hilbert space*.

An issue we seem to have is that $\nabla u_h$ is not usually defined everywhere. At every boundary between triangles, if we use piecewise linear functions, for example, the functions match on the boundary but the gradients typically do not. We can, however, consider smooth functions that approximate the piecewise linear functions $u_h$. If we think about the situation in one variable we can smooth out the transition as illustrated in Figure 6.3.3. Note that the smoothed function $u_{h,\epsilon}$ does not have gradients larger than the maximum gradient of $u_h$, and the region on which $u_h$ and $u_{h,\epsilon}$ differ has a total area that goes to zero as $\epsilon \to 0$. Then by the dominated convergence theorem [220, p. 26],

$$\int_\Omega \left\| \nabla u_{h,\epsilon}(\boldsymbol{x}) \right\|_2^2 d\boldsymbol{x} \to \int_\Omega \|\nabla u_h(\boldsymbol{x})\|_2^2 \, d\boldsymbol{x} \qquad \text{as } \epsilon \to 0.$$

Note that this argument would fail if we were dealing with *second* derivatives. Differentiating a piecewise linear function results in jumps in the first derivative. Integrating a function with a jump is perfectly fine. But if we differentiate a function with jumps *in the sense of distributions*, we obtain Dirac $\delta$-functions, whose squares are not integrable. While functions with jumps can be smoothed out, if we try to use smooth approximations $u_\epsilon(x)$ to $u(x)$ with a jump we find that $\int_a^b u_\epsilon''(x)^2 dx \to \infty$ as $\epsilon \downarrow 0$.

The matrix $A_h$ for the linear system for $u_h(\boldsymbol{x}) = \sum_{j=1}^M u_j \, \phi_j(\boldsymbol{x})$ is given by $a_{ij} = \int_\Omega \nabla \phi_i^T \nabla \phi_j \, d\boldsymbol{x}$. This matrix is clearly symmetric as $a_{ij} = a_{ji}$ by symmetry of the

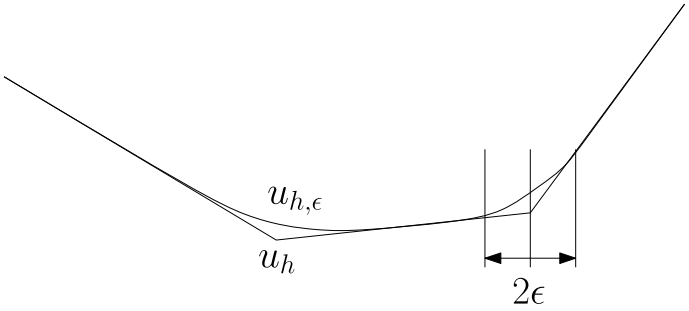



**Fig. 6.3.3** Smoothing of piecewise linear function

inner product. What is more, the matrix is positive definite. To see this, we compute

$$\begin{aligned} \boldsymbol{z}^T A_h \boldsymbol{z} &= \sum_{i,j=1}^{M} z_i z_j \int_\Omega \nabla \phi_i^T \nabla \phi_j \, d\boldsymbol{x} \\ &= \int_\Omega \left( \sum_{i=1}^{M} z_i \, \nabla \phi_i \right) \left( \sum_{j=1}^{M} z_j \, \nabla \phi_j \right) d\boldsymbol{x}. \end{aligned}$$

Setting $z_h(\boldsymbol{x}) = \sum_{j=1}^{M} z_j \, \phi_j(\boldsymbol{x})$, we see that $\nabla z_h(\boldsymbol{x}) = \sum_{j=1}^{M} z_j \, \nabla \phi_j(\boldsymbol{x})$, and so

$$\boldsymbol{z}^T A_h \boldsymbol{z} = \int_\Omega \nabla z_h(\boldsymbol{x})^T \nabla z_h(\boldsymbol{x}) \, d\boldsymbol{x} = \int_\Omega \|\nabla z_h(\boldsymbol{x})\|_2^2 \, d\boldsymbol{x} \geq 0.$$

Furthermore, $\boldsymbol{z}^T A_h \boldsymbol{z} = 0$ can only happen if $\nabla z_h(\boldsymbol{x}) = \boldsymbol{0}$ for all $\boldsymbol{x}$ except those in a set of zero volume (or area). Since $z_h$ is piecewise linear and continuous, this implies that $z_h$ is constant. As $z_h(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \partial\Omega$, that constant must be zero. That is, $\boldsymbol{z} = \boldsymbol{0}$, and thus, $A_h$ is positive definite. The system of equations is therefore solvable. However, to show that this solution method is reliable, several more issues must be dealt with.

We need to understand under what conditions as $h \to 0$ the computed solution converges $u_h \to u$ to the exact solution, and in what sense. The linear system is solvable in exact arithmetic, but we want need to know how perturbations due to roundoff error, integration error, and other sources affect the numerical result. This will involve looking at the condition number of $A_h$. The third issue is how to generate and use triangulations in order to efficiently compute the desired coefficients $a_{ij}$ and $b_j$.

### 6.3.2.1 Existence and Uniqueness of Solutions

The framework we use for developing the convergence theory uses a space of functions $V$ from which we select a finite-dimensional subspace $V_h \subset V$ to which we apply the Galerkin method. For the Poisson equation, we take $V$ to be $H^1(\Omega)$ as defined in (6.3.12). We need a bilinear form $a: V \times V \to \mathbb{R}$ that is continuous using the appropriate norm $\|\cdot\|_V$ on $V$. For the Poisson equation,

$$a(u, v) = \int_\Omega \nabla u^T \nabla v \, d\boldsymbol{x}. \tag{6.3.18}$$

We also need the linear form $b(v) = \int_\Omega f\, v \, d\boldsymbol{x}$. The solution $u \in V$ then satisfies

$$a(u, v) = b(v) \quad \text{for all } v \in V \text{ where} \tag{6.3.19}$$

$$v = 0 \quad \text{on } \partial\Omega, \text{ and} \tag{6.3.20}$$

$$u = g \quad \text{on } \partial\Omega. \tag{6.3.21}$$

The operator taking $u \in H^1(\Omega)$ to its restriction to the boundary $\partial\Omega$ is known as the *trace operator*, and is often denoted $u \mapsto \gamma u$. In fact, $\gamma: H^1(\Omega) \to H^{1/2}(\partial\Omega)$ where $H^{1/2}(\partial\Omega)$ is a *fractional order Sobolev space*. Details of how exactly fractional order Sobolev spaces are defined, and why the trace operator has values in $H^{1/2}(\partial\Omega)$ can be found in [11, 31, 245], for example.

We assume that the boundary function $g$ can be extended to a function $\widetilde{g} \in H^1(\Omega)$: $\gamma\,\widetilde{g} = g$. In fact, the trace operator $\gamma: H^1(\Omega) \to H^{1/2}(\partial\Omega)$ is onto, and so there is an extension $\widetilde{g} \in H^1(\Omega)$ where $\gamma\widetilde{g} = g$ whenever $g \in H^{1/2}(\partial\Omega)$. But with this extension of $g$, $\widetilde{u} := u - \widetilde{g} \in H^1(\Omega)$ is zero on the boundary $\partial\Omega$. Then the solution $u$ satisfies

$$a(u - \widetilde{g}, v) = a(u, v) - a(\widetilde{g}, v) = b(v) - a(\widetilde{g}, v), \qquad \text{and}$$
$$u - \widetilde{g} = 0 \quad \text{on } \partial\Omega.$$

Let $V_0 = \{\, v \in V \mid v = 0 \text{ on } \partial\Omega \,\}$. Then $\widetilde{u} \in V_0$ and

$$a(\widetilde{u}, v) = b(v) - a(\widetilde{g}, v) \quad \text{for all } v \in V_0. \tag{6.3.22}$$

To solve an equation in the weak form $\widetilde{u} \in V_0$ and

$$a(\widetilde{u}, v) = \widetilde{b}(v) \quad \text{for all } v \in V_0,$$

we need some conditions on the bilinear form $a(\cdot, \cdot)$. The main condition we use is the condition that $a(\cdot, \cdot)$ is *elliptic*: there is a constant $\alpha > 0$ where

$$a(\widetilde{u}, \widetilde{u}) \geq \alpha \, \|\widetilde{u}\|^2_{V_0} \quad \text{for all } \widetilde{u} \in V_0. \tag{6.3.23}$$

In the case of the Poisson equation, this means showing that for some $\alpha > 0$,

$$a(\widetilde{u}, \widetilde{u}) = \int_\Omega \|\nabla \widetilde{u}(\boldsymbol{x})\|_2^2 \, d\boldsymbol{x} \geq \alpha \int_\Omega \left[ \|\nabla \widetilde{u}(\boldsymbol{x})\|_2^2 + \widetilde{u}(\boldsymbol{x})^2 \right] d\boldsymbol{x} \tag{6.3.24}$$

for any smooth function $\widetilde{u}$ on $\Omega$ where $\widetilde{u}(\boldsymbol{x}) = 0$ for all $\boldsymbol{x} \in \partial\Omega$. The arguments needed to prove this involve functional analysis, specifically of Sobolev spaces [213, 245], and are also summarized in [12].

Once it is established that the bilinear form $a(u, v)$ is elliptic, we want to use this to show existence and uniqueness of solutions to (6.3.22). This can be done via the *Lax–Milgram Lemma*:

**Lemma 6.13** *(Lax–Milgram Lemma) If $a\colon V \times V \to \mathbb{R}$ is a continuous elliptic bilinear form where $V$ is a Hilbert space, and $b\colon V \to \mathbb{R}$ is a continuous linear function, then there is one and only one $u \in V$ where*

$$a(u, v) = b(v) \quad \textit{for all } V.$$

The proof of this requires some knowledge of functional analysis, which can be found in textbooks such as [11, 158, 245]. We assume the Lax–Milgram Lemma holds in what follows.

Note that for a bilinear form $a\colon V \times V \to \mathbb{R}$, $a$ is continuous if and only if there is a constant $M$ where $|a(u, v)| \leq M \, \|u\|_V \, \|v\|_V$ for all $u, v \in V$. Continuity of a linear functional $b\colon V \to \mathbb{R}$ is equivalent to there being a constant $B$ where $|b(v)| \leq B \, \|v\|_V$ for all $v \in V$.

For the Poisson equation, we can show the existence of a solution of the weak form (6.3.18) by applying the Lax–Milgram Lemma (Lemma 6.13) to the restricted weak form (6.3.22) where $a(\widetilde{u}, \widetilde{v}) = \int_\Omega \nabla \widetilde{u}^T \nabla \widetilde{v} \, d\boldsymbol{x}$, $b(\widetilde{v}) = \int_\Omega \widetilde{v} \, f \, d\boldsymbol{x}$ and $V_0 = \left\{ \widetilde{u} \in H^1(\Omega) \mid \gamma \widetilde{u} = 0 \right\}$ where $\gamma \widetilde{u}$ is the restriction of $\widetilde{u}$ to the boundary $\partial\Omega$. Also, the linear form $b(\widetilde{v})$ is continuous provided $f^2$ is integrable, as $\widetilde{v} \in H^1(\Omega)$ implies

$$\left| \int_\Omega \widetilde{v} \, f \right| \leq \left[ \int_\Omega \widetilde{v}^2 d\boldsymbol{x} \right]^{1/2} \left[ \int_\Omega f^2 d\boldsymbol{x} \right]^{1/2} \leq \|\widetilde{v}\|_{H^1(\Omega)} \left[ \int_\Omega f^2 d\boldsymbol{x} \right]^{1/2}.$$

The solution of the original problem is then $u = \widetilde{u} + \widetilde{g}$, so that $u(\boldsymbol{x}) = \widetilde{g}(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$.

### 6.3.2.2 Convergence Theory

The abstract Galerkin method for finding $v \in V$ where

$$a(u, v) = b(v) \quad \text{for all } V,$$

is to pick a finite-dimensional space of approximations $V_h \subset V$ and find $v_h \in V_h$ where

$$a(u_h, v_h) = b(v_h) \qquad \text{for all } v_h \in V_h. \tag{6.3.25}$$

The foundation of the convergence theory of finite-element methods is Céa's inequality:

**Theorem 6.14** *(Céa's inequality). If $a(u, v)$ is a continuous elliptic bilinear form on a Hilbert space $V$, then there is a constant $C$, depending only on $M$ and $\alpha$ for the elliptic form $a(\cdot, \cdot)$, where the true solution $u$ and the solution $u_h$ of (6.3.25) satisfy*

$$\|u - u_h\|_V \le C \min_{v \in V_h} \|u - v\|_V .$$

That is, the error in the solution of the Galerkin method in the $V$ norm is within a constant factor of the approximation error in the $V$ norm by functions in $V_h$.

***Proof*** Suppose that $a(u, u) \ge \alpha \|u\|_V^2$ for all $u \in V$ with $\alpha > 0$ since $a(\cdot, \cdot)$ is elliptic. Suppose also that $|a(u, v)| \le M \|u\|_V \|v\|_V$ since $a(\cdot, \cdot)$ is a continuous bilinear form. Then the Galerkin method (6.3.25) implies that $u_h \in V_h$ where

$$\begin{aligned} a(u_h, v_h) &= b(v_h) \qquad \text{for all } v \in V_h, \qquad \text{while} \\ a(u\ ,\ v\ ) &= b(\ v\ ) \qquad \text{for all } v \in V. \end{aligned}$$

Then

$$\begin{aligned} \alpha \|u_h - u\|_V^2 &\le a(u - u_h,\ u - u_h) \\ &= a(u - u_h,\ u - v_h) \qquad \text{for any } v_h \in V_h \text{ since} \\ a(u - u_h,\ u_h - v_h) &= a(u,\ u_h - v_h) - a(u_h,\ u_h - v_h) \\ &= b(u_h - v_h) - b(u_h - v_h) = 0 \qquad \text{as } u_h - v_h \in V_h \subset V. \end{aligned}$$

Thus, for any $v_h \in V_h$,

$$\begin{aligned} \alpha \|u_h - u\|_V^2 &\le a(u - u_h,\ u - v_h) \\ &\le M \|u - u_h\|_V \|v - v_h\|_V . \end{aligned}$$

Dividing by $\|u - u_h\|_V$ (assuming that this is positive) we get

$$\alpha \|u_h - u\|_V \le M \|v - v_h\|_V .$$

Setting $C = M/\alpha$, we obtain

$$\|u_h - u\|_V \le C \|v - v_h\|_V$$

for all $v_h \in V_h$. Since $V_h$ is finite-dimensional, the minimum over $v_h \in V_h$ exists, and we obtain Céa's inequality. □

Usually, in the finite element method, the spaces $V_h$ are spaces of piecewise polynomials that are interpolants over triangulations (see Section 4.3.2). So we use the approximation properties of $V_h$ in the $V$ norm. For the Poisson equation, $V = H^1(\Omega)$ or a suitable subspace of this. Because this $V = H^1(\Omega)$ norm involves first derivatives, it is necessary for the triangulations to be "well-shaped" in that the ratio of the diameter of the triangles to the narrowest width must be bounded. That is, the triangles must have a bounded "aspect ratio". See Section 4.3.3 for more details.

#### 6.3.2.3 Conditioning of the Linear Systems

The error bounds arising from Céa's inequality (Theorem 6.14) give the impression that the only issue is the ability to approximate a solution $u \in V$ by functions $u_h \in V_h$. From this point of view, the more functions in $V_h$ the better. However, the linear systems can become extremely ill-conditioned if the basis for $V_h$ is close to being linearly dependent. For example, using the basis $\phi_j(x) = x^{j-1}$ for $j = 1, 2, \ldots, n$ on $\Omega = (0, 1)$ will result in extremely ill-conditioned linear systems, with the condition number growing at least exponentially in $n$. Small errors in the formulation which can include floating point roundoff, or in the numerical solver, can result in potentially large errors in the solution.

Fortunately, well-shaped triangulations can avoid this exponential growth in the condition number under some very mild conditions. To see why, we start with the *mass matrix*, which we can use to identify near linear dependence of basis functions. For the Poisson problem using a basis $\{\phi_1, \phi_2, \ldots, \phi_N\}$, the mass matrix $M_h$ is given by

$$m_{jk} = \int_\Omega \phi_j(\boldsymbol{x}) \cdot \phi_k(\boldsymbol{x})\, d\boldsymbol{x}, \tag{6.3.26}$$

while the linear system to solve for $u_h = \sum_{j=1}^N u_j \phi_j$ has the matrix $A_h$ given by

$$a_{jk} = \int_\Omega \nabla\phi_j(\boldsymbol{x})^T \nabla\phi_k(\boldsymbol{x})\, d\boldsymbol{x}, \tag{6.3.27}$$

which is often called the *stiffness matrix*.

For a basis coming from an interpolation scheme on triangles, we look to the basis $\left\{\widehat{\phi}_1, \ldots, \widehat{\phi}_{\widehat{N}}\right\}$ for the reference triangle $\widehat{K}$. As these are linearly independent, the mass matrix $\widehat{M}$ for $\widehat{K}$ by itself,

$$\widehat{m}_{rs} = \int_{\widehat{K}} \widehat{\phi}_r(\widehat{\boldsymbol{x}}) \cdot \widehat{\phi}_s(\widehat{\boldsymbol{x}})\, d\widehat{\boldsymbol{x}},$$

must be positive definite. The condition number $\kappa_2(\widehat{M})$ can be used as a measure of the quality of the basis on $\widehat{K}$ (smaller is better), but with a fixed basis on $\widehat{K}$, this is a known and finite quantity. For example, if $\widehat{K}$ is the triangle with vertices $(0,0)$, $(1,0)$, and $(0,1)$, and the basis is linear $\widehat{\phi}_1(x,y) = x$, $\widehat{\phi}_2(x,y) = y$, and $\widehat{\phi}_3(x,y) = 1 - x - y$ then

$$\widehat{M} = \frac{1}{24}\begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}; \qquad \kappa_2(\widehat{M}) = 4.$$

We assume that the basis functions we use on a triangle $K$ in a triangulation are $\phi_j(\boldsymbol{x}) = \widehat{\phi}_r(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$ with $\boldsymbol{T}_K$ an affine transformation $\widehat{K} \to K$. We write $\boldsymbol{T}_K(\widehat{\boldsymbol{x}}) = A_K\widehat{\boldsymbol{x}} + \boldsymbol{b}_K$. Note that $\|A_K\|_2 = \mathcal{O}(h_K)$ where $h_K = \operatorname{diam}(K)$. We assume that the mesh is well-shaped so that $\kappa_2(A_K) \le \kappa_{\max}$ no matter the triangle $K$ in the triangulation. The mapping from the index for the basis function on $\widehat{K}$ to the index of the basis function on $K$ in the triangulation is $r \mapsto j = j(K, r)$.

To determine the condition number of the matrix $M_h$ we look to the quadratic form

$$\begin{aligned} \boldsymbol{u}^T M_h \boldsymbol{u} &= \sum_{j,k=1}^{N} m_{jk} u_j u_k \\ &= \int_\Omega \left(\sum_{j=1}^{N} u_j \phi_j(\boldsymbol{x})\right)\left(\sum_{k=1}^{N} u_k \phi_k(\boldsymbol{x})\right) d\boldsymbol{x} \\ &= \sum_{K\in\mathcal{T}} \int_K \left(\sum_{j=1}^{N} u_j \phi_j(\boldsymbol{x})\right)\left(\sum_{k=1}^{N} u_k \phi_k(\boldsymbol{x})\right) d\boldsymbol{x} \end{aligned}$$

where $\mathcal{T}$ is the triangulation of $\Omega$. Writing this in terms of the basis functions $\widehat{\phi}_r$ we have

$$\begin{aligned} \boldsymbol{u}^T M_h \boldsymbol{u} &= \sum_{K\in\mathcal{T}} \int_K \left(\sum_{j=1}^{N} u_j \phi_j(\boldsymbol{x})\right)\left(\sum_{k=1}^{N} u_k \phi_k(\boldsymbol{x})\right) d\boldsymbol{x} \\ &= \sum_{K\in\mathcal{T}} \int_K \left(\sum_{r=1}^{\widehat{N}} u_{j(K,r)} \widehat{\phi}_r(\widehat{\boldsymbol{x}})\right)\left(\sum_{s=1}^{\widehat{N}} u_{j(K,s)} \widehat{\phi}_s(\widehat{\boldsymbol{x}})\right) d\boldsymbol{x} \\ &\qquad \text{where } \boldsymbol{x} = T_K(\widehat{\boldsymbol{x}}), \\ &= \sum_{K\in\mathcal{T}} \sum_{r,s=1}^{\widehat{N}} u_{j(K,r)} u_{j(K,s)} \, |\det A_K| \int_{\widehat{K}} \widehat{\phi}_r(\widehat{\boldsymbol{x}}) \widehat{\phi}_s(\widehat{\boldsymbol{x}}) \, d\widehat{\boldsymbol{x}} \end{aligned}$$

$$= \sum_{K \in \mathcal{T}} |\det A_K| \sum_{r,s=1}^{\widehat{N}} u_{j(K,r)} u_{j(K,s)} \widehat{m}_{rs}.$$

If you write $\boldsymbol{u}_K$ for the vector of entries $u_{j(K,r)}$ for $r = 1, 2, \ldots, \widehat{N}$, then

$$\boldsymbol{u}^T M_h \boldsymbol{u} = \sum_{K \in \mathcal{T}} |\det A_K| \, \boldsymbol{u}_K^T \widehat{M} \boldsymbol{u}_K.$$

We can bound $\boldsymbol{u}_K^T \widehat{M} \boldsymbol{u}_K$ above and below by $\lambda_{\min}(\widehat{M}) \ \|\boldsymbol{u}_K\|_2^2$ and $\lambda_{\max}(\widehat{M}) \ \|\boldsymbol{u}_K\|_2^2 = \lambda_{\min}(\widehat{K}) \, \kappa_2(\widehat{M}) \ \|\boldsymbol{u}_K\|_2^2$ respectively, since $\widehat{M}$ is a symmetric positive definite matrix. Also note that $|\det A_K| = \text{area}(K)/\text{area}(\widehat{K})$, so

$$\boldsymbol{u}^T M_h \boldsymbol{u} = \sum_{K \in \mathcal{T}} \frac{\text{area}(K)}{\text{area}(\widehat{K})} \boldsymbol{u}_K^T \widehat{M} \boldsymbol{u}_K.$$

If $d > 2$ we should use $\text{vol}_d(K)$, the $d$-dimensional volume, instead of $\text{area}(K)$. Other than changing from areas to $d$-dimensional volumes, the remainder of the argument here holds for $\Omega \subset \mathbb{R}^d$. The value of

$$\begin{aligned}
\boldsymbol{u}_K^T \widehat{M} \boldsymbol{u}_K &= c_K \lambda_{\min}(\widehat{M}) \, \|\boldsymbol{u}_K\|_2^2 \qquad \text{where } 1 \le c_K \le \kappa_2(\widehat{M}); \text{ so} \\
\boldsymbol{u}^T M_h \boldsymbol{u} &= \sum_{K \in \mathcal{T}} \frac{\text{area}(K)}{\text{area}(\widehat{K})} c_K \lambda_{\min}(\widehat{M}) \, \|\boldsymbol{u}_K\|_2^2 \\
&= \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{K \in \mathcal{T}} \text{area}(K) c_K \, \|\boldsymbol{u}_K\|_2^2 \\
&= \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{K \in \mathcal{T}} \text{area}(K) c_K \sum_{r=1}^{\widehat{N}} u_{j(K,r)}^2.
\end{aligned}$$

We can turn this into a sum over $j$ by reversing the order of integration:

$$\boldsymbol{u}^T M_h \boldsymbol{u} = \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{j=1}^{N} \left[ \sum_{K \in \mathcal{K}(j)} \text{area}(K) c_K \right] u_j^2.$$

Here $\mathcal{K}(j)$ is the set of all triangles $K \in \mathcal{T}_h$ where $j = j(K, r)$ for some $r = 1, 2, \ldots, \widehat{N}$. This inner sum $\sum_{K \in \mathcal{K}(j)} \text{area}(K) c_K$ is between $\sum_{K \in \mathcal{K}(j)} \text{area}(K)$ and $\sum_{K \in \mathcal{K}(j)} \text{area}(K) \, \kappa_2(\widehat{M})$. The lower bound is the sum of the areas of the triangles $K$ on which $\phi_j$ is not identically zero. We will call this area $\alpha_j = \sum_{K \in \mathcal{K}(j)} \text{area}(K) = \text{area}(\text{supp}\, \phi_j)$ where $\text{supp}\, g = \overline{\{\boldsymbol{x} \mid g(\boldsymbol{x}) \neq 0\}}$. Then we have

$$\boldsymbol{u}^T M_h \boldsymbol{u} = \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{j=1}^{N} \left[ \frac{\sum_{K \in \mathcal{K}(j)} \text{area}(K)\, c_K}{\sum_{K \in \mathcal{K}(j)} \text{area}(K)} \right] \alpha_j u_j^2.$$

The quantity in square brackets is a weighted average of numbers between one and $\kappa_2(\widehat{M})$, so we can write

$$\boldsymbol{u}^T M_h \boldsymbol{u} = \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{j=1}^{N} \widetilde{c}_j \alpha_j u_j^2,$$

where $1 \le \widetilde{c}_j \le \kappa_2(\widehat{M})$. Then

$$\kappa_2(M_h) \le \kappa_2(\widehat{M}) \frac{\max_j \alpha_j}{\min_j \alpha_j} = \kappa_2(\widehat{M}) \frac{\max_j \text{area}(\text{supp}\, \phi_j)}{\min_j \text{area}(\text{supp}\, \phi_j)}. \tag{6.3.28}$$

This gives modest bounds on $\kappa_2(M_h)$ provided the areas of the supports $\text{supp}\, \phi_j$ do not vary widely in size.

There are cases where we do want the supports of $\phi_j$ to vary widely in size: there might be regions where the solution is less smooth, or changes rapidly. In these regions we want the triangles to be small, while in regions where the solution is more smooth, or changes slowly, we want to use larger triangles. If we take $D$ to be a diagonal matrix with diagonal entries $D_{jj} = \alpha_j^{-1/2}$, we can pre-condition $M_h$ with $D$:

$$\boldsymbol{w}^T D M_h D \boldsymbol{w} = \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{j=1}^{N} \widetilde{c}_j \alpha_j (\alpha_j^{-1/2} w_j)^2 = \frac{\lambda_{\min}(\widehat{M})}{\text{area}(\widehat{K})} \sum_{j=1}^{N} \widetilde{c}_j w_j^2$$

so $DM_hD$ has condition number bounded by $\kappa_2(\widehat{M})$. So even in this case, $M_h$ has a diagonal preconditioner that gives a condition number that is bounded independently of the triangulation.

We still want to get a bound on $\kappa_2(A_h)$, and for the Poisson problem,

$$a_{jk} = \int_\Omega \nabla \phi_j(\boldsymbol{x})^T \nabla \phi_k(\boldsymbol{x})\, d\boldsymbol{x}.$$

For conditioning of $A_h$, having a well-shaped triangulation is more important than for $M_h$. However, we will use the condition number of the mass matrix $M_h$ to help give us a condition number of $A_h$. More specifically, we use

$$\frac{\boldsymbol{u}^T A_h \boldsymbol{u}}{\boldsymbol{u}^T \boldsymbol{u}} = \frac{\boldsymbol{u}^T A_h \boldsymbol{u}}{\boldsymbol{u}^T M_h \boldsymbol{u}} \frac{\boldsymbol{u}^T M_h \boldsymbol{u}}{\boldsymbol{u}^T \boldsymbol{u}}.$$

Letting $u_h = \sum_{j=1}^{N} u_j \phi_j$ we can write $\boldsymbol{u}^T A_h \boldsymbol{u} = \int_\Omega \nabla u_h^T \nabla u_h \, d\boldsymbol{x}$ while $\boldsymbol{u}^T M_h \boldsymbol{u} = \int_\Omega u_h^2 \, d\boldsymbol{x}$. Note that for the Poisson problem with boundary values specified, we have $u_h(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \partial\Omega$. The ratio

$$\frac{\boldsymbol{u}^T M_h \boldsymbol{u}}{\boldsymbol{u}^T \boldsymbol{u}}$$

lies between $\lambda_{\min}(M_h)$ and $\lambda_{\max}(M_h) = \lambda_{\min}(M_h)\, \kappa_2(M_h)$, and so cannot vary greatly. The ratio

$$\frac{\boldsymbol{u}^T A_h \boldsymbol{u}}{\boldsymbol{u}^T M_h \boldsymbol{u}} = \frac{\int_\Omega \nabla u_h^T \nabla u_h \, d\boldsymbol{x}}{\int_\Omega u_h^2 \, d\boldsymbol{x}}. \tag{6.3.29}$$

If we allow $u_h$ to range over *all* functions $u$ in $H^1(\Omega)$ with zero boundary conditions, there is still a *lower bound* to this ratio $\int_\Omega \nabla u^T \nabla u \, d\boldsymbol{x} / \int_\Omega u^2 \, d\boldsymbol{x}$ that is positive. The minimum is, in fact, the smallest eigenvalue of the negative Laplacian operator:

$$\begin{aligned} -\nabla^2 u &= \lambda u \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega. \end{aligned}$$

However, there is no upper bound to the ratio $\int_\Omega \nabla u^T \nabla u \, d\boldsymbol{x} / \int_\Omega u^2 \, d\boldsymbol{x}$. Instead, we need to use *inverse estimates*, which come from the elements.

As before, we have the basis for $\widehat{K}$ given by $\{\widehat{\phi}_1, \widehat{\phi}_2, \ldots, \widehat{\phi}_{\widehat{N}}\}$, and $\phi_j(\boldsymbol{x}) = \widehat{\phi}_r(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$. Now

$$\frac{\int_{\widehat{K}} \nabla \widehat{u}^T \nabla \widehat{u} \, d\boldsymbol{x}}{\int_{\widehat{K}} \widehat{u}^2 \, d\boldsymbol{x}} \le B \qquad \text{for all } \widehat{u} = \sum_{r=1}^{\widehat{N}} c_r \widehat{\phi}_r.$$

The bound exists because $\widehat{u}$ belongs to a finite dimensional space. For example, for linear functions over the standard unit triangle $\widehat{K}$, the maximum ratio is $3/2$. Now for $\phi_j(\boldsymbol{x}) = \widehat{\phi}_r(\widehat{\boldsymbol{x}})$ and $\phi_k(\boldsymbol{x}) = \widehat{\phi}_s(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}})$, we have $\nabla\phi_j(\boldsymbol{x}) = A_K^{-T} \nabla\widehat{\phi}_r(\widehat{\boldsymbol{x}})$ and $\nabla\phi_k(\boldsymbol{x}) = A_K^{-T} \nabla\widehat{\phi}_s(\widehat{\boldsymbol{x}})$. Let $u_h(\boldsymbol{x}) = \sum_{j=1}^{N} u_j \phi_j(\boldsymbol{x})$. Then for $\boldsymbol{x} \in K$, $\nabla u_h(\boldsymbol{x}) = A_K^{-T} \sum_{r=1}^{\widehat{N}} u_{j(K,r)} \nabla\widehat{\phi}_r(\widehat{\boldsymbol{x}})$. Integrating over $K$ gives

$$\begin{aligned} \int_K \nabla u_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x}) \, d\boldsymbol{x} &= \sum_{r,s=1}^{\widehat{N}} u_{j(K,r)} u_{j(K,s)} \int_K \nabla\phi_{j(K,r)}(\boldsymbol{x})^T \nabla\phi_{j(K,s)}(\boldsymbol{x}) \, d\boldsymbol{x} \\ &= \sum_{r,s=1}^{\widehat{N}} u_{j(K,r)} u_{j(K,s)} \, |\det A_K| \int_{\widehat{K}} \nabla\widehat{\phi}_r(\widehat{\boldsymbol{x}})^T A_K^{-1} A_K^{-T} \nabla\widehat{\phi}_s(\widehat{\boldsymbol{x}}) \, d\widehat{\boldsymbol{x}} \\ &\le \left\| A_K^{-1} \right\|_2^2 \sum_{r,s=1}^{\widehat{N}} u_{j(K,r)} u_{j(K,s)} \, |\det A_K| \int_{\widehat{K}} \nabla\widehat{\phi}_r(\widehat{\boldsymbol{x}})^T \nabla\widehat{\phi}_s(\widehat{\boldsymbol{x}}) \, d\widehat{\boldsymbol{x}} \\ &= \left\| A_K^{-1} \right\|_2^2 |\det A_K| \int_{\widehat{K}} \nabla \left( \sum_{r=1}^{\widehat{N}} u_{j(K,r)} \widehat{\phi}_r \right) (\widehat{\boldsymbol{x}})^T \nabla \left( \sum_{s=1}^{\widehat{N}} u_{j(K,s)} \widehat{\phi}_s \right) (\widehat{\boldsymbol{x}}) \, d\widehat{\boldsymbol{x}}. \end{aligned}$$

Now we can use the bound for $\widehat{K}$:

$$\begin{aligned}\int_K \nabla u_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x})\,d\boldsymbol{x} &\le \left\|A_K^{-1}\right\|_2^2 |\det A_K| \int_{\widehat{K}} \nabla\left(\sum_{r=1}^{\widehat{N}} u_{j(K,r)}\widehat{\phi}_r\right)(\widehat{\boldsymbol{x}})^T \nabla\left(\sum_{s=1}^{\widehat{N}} u_{j(K,s)}\widehat{\phi}_s\right)(\widehat{\boldsymbol{x}})\,d\widehat{\boldsymbol{x}} \\ &\le B\left\|A_K^{-1}\right\|_2^2 |\det A_K| \int_{\widehat{K}} \left(\sum_{r=1}^{\widehat{N}} u_{j(K,r)}\widehat{\phi}_r\right)(\widehat{\boldsymbol{x}})^2\,d\widehat{\boldsymbol{x}} \\ &= B\left\|A_K^{-1}\right\|_2^2 \int_K u_h(\boldsymbol{x})^2\,d\boldsymbol{x}.\end{aligned}$$

Noting that $\kappa_2(A_K) = \|A_K\|_2 \left\|A_K^{-1}\right\|_2$, we see that $\left\|A_K^{-1}\right\|_2 = \kappa_2(A_K)/\|A_K\|_2 = \mathcal{O}(\kappa_{\max}/h_K)$.

Summing over all triangles, we get

$$\begin{aligned}\sum_{K\in\mathcal{T}} \int_K \nabla u_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x})\,d\boldsymbol{x} &\le B\sum_{K\in\mathcal{T}} \left\|A_K^{-1}\right\|_2^2 \int_K u_h(\boldsymbol{x})^2\,d\boldsymbol{x} \\ &\le B\max_{K\in\mathcal{T}} \left\|A_K^{-1}\right\|_2^2 \sum_{K\in\mathcal{T}} \int_K u_h(\boldsymbol{x})^2\,d\boldsymbol{x}.\end{aligned}$$

That is,

$$\int_\Omega \nabla u_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x})\,d\boldsymbol{x} = \mathcal{O}((\min_K h_K)^{-2}) \int_\Omega u_h(\boldsymbol{x})^2\,d\boldsymbol{x}, \text{ so}$$

$$\frac{\boldsymbol{u}^T A_h \boldsymbol{u}}{\boldsymbol{u}^T M_h \boldsymbol{u}} = \mathcal{O}(h_{\min}^{-2}) \qquad \text{where } h_{\min} = \min_{K\in\mathcal{T}} h_K. \tag{6.3.30}$$

Combined with the lower bound on $\boldsymbol{u}^T A_h \boldsymbol{u}/\boldsymbol{u}^T M_h \boldsymbol{u}$, this shows that $h_{\min}^2\,\kappa_2(A_h)$ is bounded provided we have a bound on $h_{\max}/h_{\min}$. As $h_{\min}$ decreases, the condition number grows, but only quadratically.

If we have a situation where $h_{\max}/h_{\min}$ is large, but neighboring triangles have similar sizes ($h_K/h_L \le 2$ for any neighboring triangles $K$ and $L$, for example), then it is still possible to have a diagonal preconditioner with $\kappa_2(DA_hD) = \mathcal{O}(h_{\max}^{-2})$.

### 6.3.3 Handling Boundary Conditions

So far we have considered *essential* (or *Dirichlet*) *boundary conditions*, where $u(\boldsymbol{x}) = g(\boldsymbol{x})$ for all $\boldsymbol{x} \in \partial\Omega$ with $g$ a given function. There are many other kinds of linear boundary conditions, most particularly *natural* (or *Neumann*) *boundary conditions* which in this case have the form $\partial u/\partial n(\boldsymbol{x}) = h(\boldsymbol{x})$ on $\partial\Omega$ where $\partial/\partial n$ is the outward normal derivative, and *mixed* (or *Robin*) *boundary conditions* which combine the previous two types. Consider the elliptic partial differential equation

$$-\operatorname{div}(a(\boldsymbol{x})\,\nabla u) + b(\boldsymbol{x})\,u = f(\boldsymbol{x}) \qquad \text{in } \Omega. \tag{6.3.31}$$

We can create a weak form through multiplying by a smooth function $v(\boldsymbol{x})$ and integrating over $\Omega$. Then

$$\begin{aligned}
&\int_\Omega v\,[-\operatorname{div}(a(\boldsymbol{x})\,\nabla u) + b(\boldsymbol{x})\,u - f]\,d\boldsymbol{x} \\
&\quad = \int_\Omega \left\{-\operatorname{div}(v\,a(\boldsymbol{x})\,\nabla u) + a(\boldsymbol{x})\nabla v^T\nabla u + v\,[b(\boldsymbol{x})\,u - f]\right\} d\boldsymbol{x} \\
&\quad = -\int_{\partial\Omega} v\,a(\boldsymbol{x})\,\nabla u^T \boldsymbol{n}(\boldsymbol{x})\,dS(\boldsymbol{x}) \\
&\qquad + \int_\Omega \left[a(\boldsymbol{x})\nabla v^T\nabla u + b(\boldsymbol{x})\,v\,u - f\,v\right] d\boldsymbol{x}.
\end{aligned}$$

If we have essential boundary conditions $u(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \Gamma_D$ with $\Gamma_D$ a subset of $\partial\Omega$, then we need to impose the condition that $v(\boldsymbol{x}) = 0$ for $\boldsymbol{x} \in \Gamma_D$. On the other hand, if we have natural boundary conditions $\partial u/\partial n(\boldsymbol{x}) = \boldsymbol{n}(\boldsymbol{x})^T\nabla u(\boldsymbol{x}) = h(\boldsymbol{x})$ for $\boldsymbol{x} \in \Gamma_N$, then we have to set

$$\begin{aligned}
0 = &-\int_{\partial\Omega} v(\boldsymbol{x})\,a(\boldsymbol{x})\,\nabla u(\boldsymbol{x})^T\boldsymbol{n}(\boldsymbol{x})\,dS(\boldsymbol{x}) \\
&+ \int_\Omega \left[a(\boldsymbol{x})\nabla v(\boldsymbol{x})^T\nabla u(\boldsymbol{x}) + b(\boldsymbol{x})\,v(\boldsymbol{x})\,u(\boldsymbol{x}) - f(\boldsymbol{x})\,v(\boldsymbol{x})\right] d\boldsymbol{x}.
\end{aligned}$$

Provided the part of the boundary on which the natural conditions hold, $\Gamma_N$ is complementary to the boundary where the essential Dirichlet conditions hold $\Gamma_N = \partial\Omega\backslash\Gamma_D$, we can write

$$\begin{aligned}
\int_{\partial\Omega} v(\boldsymbol{x})\,a(\boldsymbol{x})\,\nabla u(\boldsymbol{x})^T\boldsymbol{n}(\boldsymbol{x})\,dS(\boldsymbol{x}) &= \int_{\Gamma_N} v(\boldsymbol{x})\,a(\boldsymbol{x})\,\frac{\partial u}{\partial n}(\boldsymbol{x})\,dS(\boldsymbol{x}) \\
&= \int_{\Gamma_N} v(\boldsymbol{x})\,a(\boldsymbol{x})\,h(\boldsymbol{x})\,dS(\boldsymbol{x}).
\end{aligned}$$

So the weak form for these boundary conditions is

$$\begin{aligned}
&\int_\Omega \left[a(\boldsymbol{x})\nabla v(\boldsymbol{x})^T\nabla u(\boldsymbol{x}) + b(\boldsymbol{x})\,v(\boldsymbol{x})\,u(\boldsymbol{x})\right] d\boldsymbol{x} \\
&\quad = \int_\Omega f(\boldsymbol{x})\,v(\boldsymbol{x})\,d\boldsymbol{x} + \int_{\Gamma_N} v(\boldsymbol{x})\,a(\boldsymbol{x})\,h(\boldsymbol{x})\,dS(\boldsymbol{x}) \\
&\qquad \text{for all smooth } v \text{ where } v(\boldsymbol{x}) = 0 \text{ on } \Gamma_D.
\end{aligned} \tag{6.3.32}$$

The Galerkin method for this problem is then: find $u_h \in V_h$ where $u_h(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x} \in \Gamma_D$, and

$$\int_\Omega \left[a(\boldsymbol{x})\nabla v_h(\boldsymbol{x})^T \nabla u_h(\boldsymbol{x}) + b(\boldsymbol{x})\, v_h(\boldsymbol{x})\, u_h(\boldsymbol{x})\right] d\boldsymbol{x}$$
$$= \int_\Omega f(\boldsymbol{x})\, v_h(\boldsymbol{x})\, d\boldsymbol{x} + \int_{\Gamma_N} v_h(\boldsymbol{x})\, a(\boldsymbol{x})\, h(\boldsymbol{x})\, dS(\boldsymbol{x})$$
$$\text{for all } v_h \in V_h \text{ where } v_h(\boldsymbol{x}) = 0 \text{ on } \Gamma_D. \tag{6.3.33}$$

This gives a symmetric linear system of equations for the coefficients $u_j$ in $u_h = \sum_{j=1}^N u_j \phi_j$. The matrix entries are

$$a_{jk} = \int_\Omega \left[a(\boldsymbol{x})\nabla\phi_j(\boldsymbol{x})^T \nabla\phi_k(\boldsymbol{x}) + b(\boldsymbol{x})\, \phi_j(\boldsymbol{x})\, \phi_k(\boldsymbol{x})\right] d\boldsymbol{x}.$$

Provided $a(\boldsymbol{x}) > 0$ and $b(\boldsymbol{x}) \geq 0$ for all $\boldsymbol{x} \in \Omega$, and $\Gamma_D$ has positive length (for $d = 2$) or area (for $d = 3$), then the matrix $A_h = \left[a_{jk} \mid j, k = 1, 2, \ldots, N\right]$ is also positive definite. The right-hand side is the vector with components

$$f_j = \int_\Omega f(\boldsymbol{x})\, \phi_j(\boldsymbol{x})\, d\boldsymbol{x} + \int_{\Gamma_N} \phi_j(\boldsymbol{x})\, a(\boldsymbol{x})\, h(\boldsymbol{x})\, dS(\boldsymbol{x}).$$

*Robin boundary conditions* are a mix of Dirichlet and natural boundary conditions and have the form

$$\frac{\partial u}{\partial n}(\boldsymbol{x}) + c(\boldsymbol{x})\, u(\boldsymbol{x}) = h(\boldsymbol{x}) \qquad \text{for } \boldsymbol{x} \in \Gamma_R \subseteq \partial\Omega. \tag{6.3.34}$$

The weak form of (6.3.31) with $u = g$ on $\Gamma_D$ and $\partial u/\partial n + c\, u = h$ on $\Gamma_R$ where $\Gamma_D$ and $\Gamma_N$ partition the boundary $\partial\Omega$ is

$$\int_\Omega \left[a(\boldsymbol{x})\nabla v^T \nabla u + b(\boldsymbol{x})\, u\, v\right] d\boldsymbol{x} + \int_{\Gamma_R} a(\boldsymbol{x})c(\boldsymbol{x})\, u\, v\, dS(\boldsymbol{x})$$
$$= \int_\Omega f(\boldsymbol{x})\, v\, d\boldsymbol{x} + \int_{\Gamma_R} a(\boldsymbol{x})h(\boldsymbol{x})\, v\, dS(\boldsymbol{x})$$

for all smooth $v$ with $v = 0$ on $\Gamma_D$. Usually we require that $c(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \Gamma_N$. The stiffness matrix is then given by

$$a_{ij} = \int_\Omega \left[a(\boldsymbol{x})\nabla\phi_i^T \nabla\phi_j + b(\boldsymbol{x})\, \phi_i\, \phi_j\right] d\boldsymbol{x} + \int_{\Gamma_R} a(\boldsymbol{x})c(\boldsymbol{x})\, \phi_i\, \phi_j\, dS(\boldsymbol{x}),$$

and is again positive definite under the inequalities assumed above on $a$, $b$, and $c$ provided $\text{area}(\Gamma_D) > 0$ or $b(\boldsymbol{x}) > 0$ for all $\boldsymbol{x} \in \Omega$.

#### 6.3.3.1 Numerical Integration

In general, these entries will need to be computed numerically using a suitable numerical integration method, such as are described in Section 5.3.4. This will perturb the matrix entries and the right-hand side of the linear system to be solved. Using a numerical approximation of these integrals

$$a_{jk} \approx \sum_{\ell=1}^{M} w_\ell \left[ a(z_\ell) \nabla \phi_j(z_\ell)^T \nabla \phi_k(z_\ell) + b(z_\ell)\, \phi_j(z_\ell)\, \phi_k(z_\ell) \right]$$

we still obtain symmetric, and provided there are sufficiently many integration points $z_\ell$ in each triangle, positive definite linear systems of equations.

Given an integration method on a reference triangle $\widehat{K}$ and using an affine transformation $\boldsymbol{T}_K : \widehat{K} \to K$ we have a corresponding integration method on $K$:

$$\int_{\widehat{K}} \widehat{\psi}(\widehat{\boldsymbol{x}})\, d\widehat{\boldsymbol{x}} \approx \sum_{\ell=1}^{\widehat{M}} \widehat{w}_\ell\, \widehat{\psi}(\widehat{z}_\ell).$$

If $\psi(\boldsymbol{x}) = \widehat{\psi}(\widehat{\boldsymbol{x}})$ where $\boldsymbol{x} = \boldsymbol{T}_K(\widehat{\boldsymbol{x}}) = A_K \widehat{\boldsymbol{x}} + \boldsymbol{b}_K$, we have the approximation

$$\begin{aligned} \int_K \psi(\boldsymbol{x}) &= |\det A_K| \int_{\widehat{K}} \widehat{\psi}(\widehat{\boldsymbol{x}})\, d\widehat{\boldsymbol{x}} \\ &\approx |\det A_K| \sum_{\ell=1}^{\widehat{M}} \widehat{w}_\ell\, \widehat{\psi}(\widehat{z}_\ell) \\ &= |\det A_K| \sum_{\ell=1}^{\widehat{M}} \widehat{w}_\ell\, \psi(\boldsymbol{T}_K(\widehat{z}_\ell)). \end{aligned}$$

For $\phi_j(\boldsymbol{x}) = \widehat{\phi}_r(\widehat{\boldsymbol{x}})$ when $\boldsymbol{x} \in K$, note that $\nabla \phi_j(\boldsymbol{x}) = A_K^{-T} \nabla \widehat{\phi}_r(\widehat{\boldsymbol{x}})$, and so

$$\nabla \phi_j(\boldsymbol{x})^T \nabla \phi_k(\boldsymbol{x}) = \nabla \widehat{\phi}_r(\widehat{\boldsymbol{x}})^T A_K^{-1} A_K^{-T} \nabla \widehat{\phi}_s(\widehat{\boldsymbol{x}}),$$

where $\phi_k(\boldsymbol{x}) = \widehat{\phi}_s(\widehat{\boldsymbol{x}})$ for $\boldsymbol{x} \in K$.

If we can write $\Omega = \bigcup_{K \in \mathcal{T}_h} K$ where $\mathcal{T}_h$ is a triangulation of $\Omega$, then the integral

$$\begin{aligned} &\int_\Omega a(\boldsymbol{x}) \nabla \phi_j(\boldsymbol{x})^T \nabla \phi_k(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad = \sum_{K \in \mathcal{T}_h} \int_K a(\boldsymbol{x}) \nabla \phi_j(\boldsymbol{x})^T \nabla \phi_k(\boldsymbol{x})\, d\boldsymbol{x} \\ &\quad = \sum_{K \in \mathcal{T}_h} |\det A_K| \int_{\widehat{K}} a(\boldsymbol{T}_K(\widehat{\boldsymbol{x}})) \nabla \widehat{\phi}_r(\widehat{\boldsymbol{x}})^T A_K^{-1} A_K^{-T} \nabla \widehat{\phi}_k(\widehat{\boldsymbol{x}})\, d\widehat{\boldsymbol{x}} \end{aligned}$$

$$\text{where } \phi_j(\boldsymbol{x}) = \widehat{\phi}_r(\widehat{\boldsymbol{x}}),\ \ \phi_k(\boldsymbol{x}) = \widehat{\phi}_s(\widehat{\boldsymbol{x}}) \text{ for } \boldsymbol{x} \in K$$

$$\approx \sum_{K \in \mathcal{T}_h} |\det A_K| \sum_{\ell=1}^{\widehat{M}} \widehat{w}_\ell a(\boldsymbol{T}_K(\widehat{\boldsymbol{z}}_\ell)) \nabla \widehat{\phi}_r(\widehat{\boldsymbol{z}}_\ell)^T A_K^{-1} A_K^{-T} \nabla \widehat{\phi}_k(\widehat{\boldsymbol{z}}_\ell).$$

Similarly,

$$\int_\Omega b(\boldsymbol{x}) \phi_j(\boldsymbol{x}) \phi_k(\boldsymbol{x})\, d\boldsymbol{x}$$

$$\approx \sum_{K \in \mathcal{T}_h} |\det A_K| \sum_{\ell=1}^{\widehat{M}} \widehat{w}_\ell b(\boldsymbol{T}_K(\widehat{\boldsymbol{z}}_\ell)) \widehat{\phi}_r(\widehat{\boldsymbol{z}}_\ell) \widehat{\phi}_s(\widehat{\boldsymbol{z}}_\ell).$$

If the basis functions $\widehat{\phi}_r$ on $\widehat{K}$ are polynomials of degree $\le m$, we need the integration methods to be exact for polynomials of degree $\le 2(m-1)$ in order to get convergence, and preferably exact for polynomials of degree $\le 2m$. To see why, consider a triangle $K$ of diameter $h_K$ that is much smaller than the diameter of $\Omega$. Even though we might assume $a(\boldsymbol{x})$ and $b(\boldsymbol{x})$ to change slowly over $K$, the same cannot be said of the basis functions $\phi_j(\boldsymbol{x})$. Thus we need to integrate $\int_K \nabla \phi_j^T \nabla \phi_k\, d\boldsymbol{x}$ (and $\int_K \phi_j \phi_k\, d\boldsymbol{x}$ if $b \not\equiv 0$) exactly. For example, if we are dealing with cubic Hermite elements we should use an integration method that is exact for degree 4 (and if $b \not\equiv 0$, degree 6) polynomials.

### 6.3.4 *Convection—Going with the Flow*

If $u(t, \boldsymbol{x})$ represents the concentration of an unreacting chemical species pulled along by a current in water with velocity $\boldsymbol{v}(\boldsymbol{x})$, for example, the equation for $u(t, \boldsymbol{x})$ is

$$\frac{\partial u}{\partial t} + (\boldsymbol{v}(\boldsymbol{x}) \cdot \nabla) u = \operatorname{div}(D \nabla u) + f(\boldsymbol{x}) \qquad \text{in } \Omega \tag{6.3.35}$$

with various boundary conditions. The boundary conditions can describe prescribed concentrations (perhaps at the inflow to a region: $u = g$ on $\Gamma_D$), and zero flux conditions (that apply at a wall, for example, where $\boldsymbol{v} \cdot \boldsymbol{n}\, u + D\, \partial u / \partial n = 0$ on $\Gamma_{ZF}$), and outflow conditions ($\partial u / \partial n = 0$ on $\Gamma_O$). The velocity field $\boldsymbol{v}(\boldsymbol{x})$ represents the velocity of the current at $\boldsymbol{x}$.

If we look for steady-state solutions, we set $\partial u / \partial t = 0$ and so

$$(\boldsymbol{v}(\boldsymbol{x}) \cdot \nabla)\, u - \operatorname{div}(D \nabla u) = f(\boldsymbol{x}) \qquad \text{in } \Omega.$$

If we use $w(\boldsymbol{x})$ as a smooth function for creating the weak form, then the weak form is

$$\int_\Omega \left[ w\, \boldsymbol{v} \cdot \nabla u + D\, \nabla w^T \nabla u \right] d\boldsymbol{x} - \int_{\partial\Omega} w\, D\, \frac{\partial u}{\partial n}\, dS = \int_\Omega w\, f(\boldsymbol{x})\, d\boldsymbol{x}.$$

The main difference with the equation without convection is the term $\int_\Omega w\, \boldsymbol{v} \cdot \nabla u\, d\boldsymbol{x}$. Note that

$$\operatorname{div}(w\, u\, \boldsymbol{v}) = (u\boldsymbol{v}) \cdot \nabla w + (w\boldsymbol{v}) \cdot \nabla u + w\, u \operatorname{div} \boldsymbol{v}.$$

If $\operatorname{div} \boldsymbol{v} = 0$, which is the case for an incompressible flow field, then

$$\begin{aligned} \int_\Omega w\boldsymbol{v} \cdot \nabla u\, d\boldsymbol{x} &= \int_\Omega [\operatorname{div}(w\, u\, \boldsymbol{v}) - u\boldsymbol{v} \cdot \nabla w]\, d\boldsymbol{x} \\ &= \int_{\partial\Omega} w\, u\, \boldsymbol{v} \cdot \boldsymbol{n}\, dS - \int_\Omega u\boldsymbol{v} \cdot \nabla w\, d\boldsymbol{x}. \end{aligned}$$

If $w = 0$ on $\partial\Omega$ then we get

$$\int_\Omega w\boldsymbol{v} \cdot \nabla u\, d\boldsymbol{x} = -\int_\Omega u\boldsymbol{v} \cdot \nabla w\, d\boldsymbol{x}.$$

In terms of the matrices, if $b_{jk} = \int_\Omega \phi_j\, \boldsymbol{v} \cdot \nabla \phi_k\, d\boldsymbol{x}$ and *either* $\phi_j$ or $\phi_k$ is zero on $\partial\Omega$, then $b_{kj} = -b_{jk}$. That is, apart from terms related to the boundary, the matrix $B_h = \left[ b_{jk} \mid j, k = 1, 2, \ldots, N \right]$ is anti-symmetric. If $a_{jk} = \int_\Omega D\, \nabla \phi_j^T \nabla \phi_k\, d\boldsymbol{x}$, then $A_h$ is positive definite provided $\Gamma_D$ has length (or area) that is positive. In that case and ignoring the boundary terms, $A_h + B_h$ would also be positive definite. The resulting system of equations is then invertible. The condition number of $A_h + B_h$ would also not be much larger than that of $A_h$.

There can be problems where $D > 0$ but is small, and $\boldsymbol{v}(\boldsymbol{x})$ is large. This is the convection dominated regime. Although, apart from boundary terms, $B_h$ is anti-symmetric, if $B_h$ has large entries this can cause problems with solving $(A_h + B_h)\boldsymbol{u}_h = \boldsymbol{f}_h$ for $\boldsymbol{u}_h$.

### 6.3.5 Higher Order Problems

Fourth order partial differential equations arise in a number of settings, such as elastic plate problems. A typical example is the biharmonic equation that can be written as

$$\Delta\Delta u = f(\boldsymbol{x}) \qquad \text{in } \Omega \tag{6.3.36}$$

where $\Delta$ is the Laplacian operation ($\Delta u = \partial^2 u/\partial x^2 + \partial^2 u/\partial y^2$ in two dimensions) and appropriate boundary conditions, such as Dirichlet conditions $u(\boldsymbol{x}) = g(\boldsymbol{x})$, $\partial u/\partial n(\boldsymbol{x}) = k(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$. The weak form of the equation with Dirichlet boundary conditions is that

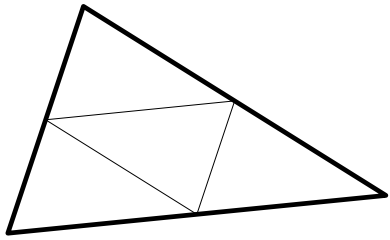


**Fig. 6.3.4** Triangle decomposition

$$\int_\Omega [(\Delta u)(\Delta v) - f\, v]\, d\boldsymbol{x} \qquad \text{for all smooth } v, \tag{6.3.37}$$

where $v = \partial v/\partial n = 0$ on $\partial\Omega$. Standard conforming finite element methods have to use basis functions $\phi_i$ where $\int_\Omega (\Delta\phi_i)^2\, d\boldsymbol{x}$ is finite. This means that if $\phi_i$ is piecewise smooth, then there cannot be any jumps in $\nabla\phi_i$. The basis functions should therefore be $C^1$ (continuous first derivatives), which are harder to create. Section 4.3.2.1 shows some examples: the Argyris element (Figure 4.3.7), and the HCT macro element (Figure 4.3.8). The order of convergence of these methods is essentially given by the order of the polynomials that can be represented by the elements used. These $C^1$ finite elements are complicated to construct, so there has been a great deal of interest in other methods of solving equations like the biharmonic equation. The equation $\Delta\Delta u = f$ in $\Omega$ with Dirichlet boundary conditions is an elliptic partial differential equation on $H^2(\Omega)$. Most of the theory of this section can be extended to problems of this type, although the condition number of the system of equations $\kappa_2(A_h) = \mathcal{O}(h_{\min}^{-4})$ rather than $\mathcal{O}(h_{\min}^{-2})$ for the second order elliptic equations.

**Exercises.**

(1) Implement the finite difference method for the problem $\nabla\cdot(a(\boldsymbol{x})\,\nabla u) = f(\boldsymbol{x})$ for $\boldsymbol{x}\in\Omega\subset\mathbb{R}^2$ and $u(\boldsymbol{x}) = g(\boldsymbol{x})$ for $\boldsymbol{x}\in\partial\Omega$ using the approximation

$$\frac{\partial}{\partial x}\left(a(x,y)\frac{\partial u}{\partial x}(x,y)\right) \approx \frac{1}{h}\left[a(x+\tfrac{1}{2}h)\frac{u(x+h,y)-u(x,y)}{h} + a(x-\tfrac{1}{2}h)\frac{u(x-h,y)-u(x,y)}{h}\right].$$

Show empirically that the asymptotic error of this formula is $\mathcal{O}(h^m)$ as $h\to 0$ for some $m$ assuming that both $a$ and $u$ are smooth. What is the value of $m$ in general?

(2) Consider solving the partial differential equations $\operatorname{div}\boldsymbol{q} = f(\boldsymbol{x})$ and $\boldsymbol{q} = \nabla u$ in $\Omega\subset\mathbb{R}^2$. From the divergence theorem, note that for a region $R$, $\int_R \operatorname{div}\boldsymbol{q}\, d\boldsymbol{x} = \int_{\partial R}\boldsymbol{q}\cdot\boldsymbol{n}\, dS$. In particular, for a point $(x_j, y_k)\in\Omega$ where $x_j = x_0 + j\,h$ and $y_k = y_0 + k\,h$, let $R$ be the square $(x_j - \frac{1}{2}h,\ x_j + \frac{1}{2}h)\times(y_k - \frac{1}{2}h,\ y_k + \frac{1}{2}h)$. We set the values of

$$q_1(x_j, y_k \pm \frac{1}{2}h)\,\operatorname{sign}(y_k \pm \frac{1}{2}h) = h^{-1}\left[u(x_j, y_k \pm h) - u(x_j, y_k)\right],$$

$$q_2(x_j \pm \frac{1}{2}h, y_k)\,\mathrm{sign}(x_j \pm \frac{1}{2}h) = h^{-1}\left[u(x_j \pm h, y_k) - u(x_j, y_k)\right].$$

Develop equations for the differential equations for values of $u$ and $q_1, q_2$ at the above points using $\int_{\partial R} \boldsymbol{q} \cdot \boldsymbol{n}\, dS = \int_R \mathrm{div}\, \boldsymbol{q}\, d\boldsymbol{x} \approx f(x_j, y_k)\, h^2$. This approach, using fluxes like $\boldsymbol{q}$ and approximations of integrals, is known as the *finite volume method*.

(3) For $\Omega = \left\{ (x, y) \mid x^2 + y^2 < 1 \right\}$ and a regular grid $(x_j, y_k)$ with $x_j = j\,h$ and $y_k = k\,h$ for grid spacing $h > 0$, let $A_h$ be the matrix for the differential equation $-\nabla^2 u = f(\boldsymbol{x})$ in $\Omega$ with boundary conditions $u(\boldsymbol{x}) = 0$ on $\partial\Omega$ discretized using the finite difference method on these grid points. Plot the condition number $\kappa_2(A_h)$ against $h$ for $h = 2^{-k}$, $k = 1, 2, \ldots, 7$. Note that if $A_h$ is $n_h \times n_h$, then $n_h$ is the number of grid points in $\Omega$, which is $\sim \pi/h^2$ as $h \to 0$. Because $n_h^2$ becomes large rapidly as $h$ decreases, it is important that if $A_h$ is created explicitly, then it should be represented as a sparse matrix.

(4) Suppose we apply the finite element method to $\nabla \cdot (a\,\nabla u) = f(\boldsymbol{x})$ in $\Omega$ with a triangulation $\mathcal{T}_h$ with each element $T \in \mathcal{T}_h$ having diameter $\le C_1\, h$. Suppose also that $a(\boldsymbol{x})$ is smooth. Suppose that we use Lagrange basis functions of degree $d$ (as illustrated in Figure 4.3.1). Show that using an integration method that is exact for polynomials of degree $\le m$ will give errors in the estimate of $\int_K a(\boldsymbol{x})\, \nabla\phi_j(\boldsymbol{x})^T \nabla\phi_k(\boldsymbol{x})\, d\boldsymbol{x}$ of size $\mathcal{O}(h^{m-2d+3}) \int_K \left\|\nabla\phi_j(\boldsymbol{x})\right\| \left\|\nabla\phi_k(\boldsymbol{x})\right\| d\boldsymbol{x}$ as $h \to 0$ for basis functions $\phi_j$, $\phi_k$, provided $m \ge 2d - 2$. [**Hint:** Let $\widetilde{a}(\boldsymbol{x})$ be an interpolant of $K$ of degree $m - 2d + 2$; use the estimate $\max_{\boldsymbol{x}\in K} |a(\boldsymbol{x}) - \widetilde{a}(\boldsymbol{x})| = \mathcal{O}(h^{m-2d+3})$.]

(5) Consider the PDE boundary value problem

$$\begin{aligned} \mathrm{div}(a(\boldsymbol{x})\, u) &= f(\boldsymbol{x}) \qquad \text{for } \boldsymbol{x} \in \Omega, \\ \beta(\boldsymbol{x})u(\boldsymbol{x}) + \frac{\partial u}{\partial n}(\boldsymbol{x}) &= g(\boldsymbol{x}) \qquad \text{for } \boldsymbol{x} \in \partial\Omega. \end{aligned}$$

Write this PDE in weak form. Show that the bilinear form in the weak form is coercive with respect to $H^1(\Omega)$ provided $\inf_{\boldsymbol{x}\in\Omega} a(\boldsymbol{x}) > 0$ and $\inf_{\boldsymbol{x}\in\partial\Omega} \beta(\boldsymbol{x}) > 0$. Assume that $\int_{\partial\Omega} u^2\, dS(\boldsymbol{x}) / \int_\Omega \left[\|\nabla u\|^2 + u^2\right] d\boldsymbol{x}$ has a positive lower bound. These kinds of boundary conditions are called *Robin conditions* after Victor Robin (French mathematician, 1855-1897).

(6) Use some implementation of the finite element method (perhaps this chapter's Project), to solve the equation $-\nabla^2 u = f(\boldsymbol{x})$ in $\Omega := \left\{ (x, y) \mid x^2 + y^2 < 1 \;\&\; \max(x, y) > 0 \right\}$ with $u(x, y) = g(x, y)$ for $(x, y) \in \partial\Omega$ using the finite element method with piecewise linear basis functions on a triangulation you generate. The function $g(x, y)$ is not piecewise linear, but we can create a piecewise linear approximation by minimizing $\int_{\partial\Omega} (u_h(\boldsymbol{x}) - g(\boldsymbol{x}))^2 dS(\boldsymbol{x})$, which can be done by computing the mass matrix on the boundary: solve $\widetilde{M}\widetilde{\boldsymbol{u}} = \widetilde{\boldsymbol{b}}$ where $\widetilde{m}_{k\ell} = \int_{\partial\Omega} \phi_k(\boldsymbol{x})\, \phi_\ell(\boldsymbol{x})\, dS(\boldsymbol{x})$, $\widetilde{b}_\ell = \int_{\partial\Omega} g(\boldsymbol{x})\, \phi_\ell(\boldsymbol{x})\, dS(\boldsymbol{x})$, and we set $u_h(\boldsymbol{p}_k) = \widetilde{u}_k$ where $\boldsymbol{p}_k \in \partial\Omega$. We use $g(x, y) = x^2 - e^y xy + \cos(x + y)$

and $f(x, y) = 2\cos(y + x) + x\,(y + 2)\,e^y - 2$. Note that the exact solution is, in fact, $u(x, y) = g(x, y)$. Use this fact to compute the error in the finite element solution with an element size of approximately 0.1.

(7) Any triangle can be subdivided into four congruent subtriangles (*congruent*, meaning same size and shape, but possibly different positions and orientations): join the midpoints of each edge. This is illustrated in Figure 6.3.4. Write code to take a given triangulation represented by $p$ (position) and $t$ (triangle) arrays as described in Exercise 2, and produces a new triangulation where each triangle in the original triangulation is replaced by its subdivision into four subtriangles.

(8) Using the subdivision method of the previous Exercise and the method used in Exercise 2 to estimate the convergence rate of the finite element method. Do this by plotting $\max_i \left|(\boldsymbol{u}_h)_i - u(\boldsymbol{p}_i)\right|$ against the number of subdivisions made to the original triangulation. Do this for $j$ levels of triangle decomposition for $j = 0, 1, 2, 3, 4$. Plot the results with a logarithmic scale for the maximum error, but linear in $j$. Use this to obtain an estimate for the maximum error of the form $C\,h^\alpha$ where $h$ is the maximum diameter of the triangles in the triangulation. Note that each level of triangle decomposition halves the value of $h$.

(9) Let $\widehat{\phi}_j$, $j = 1, 2, 3$, be the standard linear nodal basis functions on the standard unit triangle $\widehat{K}$ with vertices $\widehat{\boldsymbol{v}}_1 = (0, 0)$, $\widehat{\boldsymbol{v}}_2 = (1, 0)$, and $\widehat{\boldsymbol{v}}_3 = (0, 1)$, and $\widehat{\psi}_k$, $k = 1, 2, \ldots, 6$, be quadratic nodal basis functions on $\widehat{K}$ (see Figure 4.3.1(a) for interpolation nodes). Find the matrix $B$ $(3 \times 6)$ so that $\widehat{\phi}_j = \sum_{k=1}^{6} b_{jk}\widehat{\psi}_k$. Find a pseudo-inverse $C$ $(6 \times 3)$ so that $BC = I$ $(3 \times 3)$. The matrix $C$ should have proper symmetries so that if $T : \widehat{K} \to \widehat{K}$ is an affine symmetry of $\widehat{K}$ where $\widehat{\phi}_\ell = \widehat{\phi}_j \circ T$ and $\widehat{\psi}_m = \widehat{\psi}_k \circ T$, then $c_{m\ell} = c_{kj}$. [**Note:** The pseudo-inverse can be the Moore–Penrose pseudo-inverse (2.5.10).]

## 6.4 Partial Differential Equations—Diffusion and Waves

Introducing time into partial differential equations gives a new range of phenomena as well as numerical methods. Most methods for these problems are time-stepping methods, that combine methods for initial value problems discussed in Section 6.1 with the methods for partial differential equations in Section 6.3. Standard examples of these kinds of equations include the *diffusion* or *heat equation*

$$\frac{\partial u}{\partial t} = \nabla \cdot (D(\boldsymbol{x})\nabla u) + f(t, \boldsymbol{x}) \qquad \text{in } \Omega \tag{6.4.1}$$

and the *wave equation*

$$\frac{\partial^2 u}{\partial t^2} = \nabla \cdot \left(c^2 \nabla u\right) + f(t, \boldsymbol{x}) \qquad \text{in } \Omega. \tag{6.4.2}$$

The boundary conditions can be Dirichlet type conditions ($u(t, \boldsymbol{x}) = g(t, \boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$) or Neumann type conditions ($\partial u/\partial n(t, \boldsymbol{x}) = g(t, \boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$), or mixed Robin type conditions ($\alpha\, u(t, \boldsymbol{x}) + \beta\, \partial u/\partial n(t, \boldsymbol{x}) = g(t, \boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$). It should be noted that the behavior of solutions to diffusion and wave equations are quite different. Solutions of diffusion equations rapidly smooth out as time increases, while solutions of wave equations generally maintain their roughness over time. Nonlinear wave equations are particularly challenging.

Typically, finite element or finite difference methods are used for the spatial variables, while standard ODE methods are used in the time variable. More recently, there have been considerable efforts to develop so-called space–time and moving mesh methods that can have advantages in dealing with, for example, simulations of isolated waves. Space-time discretizations are outside the scope of this book.

### 6.4.1 Method of Lines

The method of lines first uses a discretization of the spatial variables, leaving the time derivatives as time derivatives. For example, consider the standard diffusion equation

$$\begin{aligned} \frac{\partial u}{\partial t} &= \nabla^2 u + f(t, \boldsymbol{x}) \qquad \text{in } \Omega, \\ u(t, \boldsymbol{x}) &= g(\boldsymbol{x}) \qquad \text{on } \partial\Omega, \end{aligned}$$

we can apply the finite element method in $\boldsymbol{x}$: triangulate $\Omega$, and create a basis of functions $\phi_i$, $i = 1, 2, \ldots, N$, based on a particular element for this triangulation. We write $u(t, \boldsymbol{x}) = \sum_{i=1}^{N} u_i(t)\, \phi_i(\boldsymbol{x})$ in terms of the basis functions $\phi_i$ with time-varying coefficients $u_i(t)$. Applying the weak form with $v(\boldsymbol{x}) = \phi_j(\boldsymbol{x})$ with $v = 0$ on the boundary $\partial\Omega$,

$$\begin{aligned} \int_\Omega \sum_{i=1}^{N} \frac{du_i}{dt}(t)\phi_i(\boldsymbol{x})\, \phi_j(\boldsymbol{x})\, d\boldsymbol{x} = \int_\Omega \left[ -\sum_{i=1}^{N} u_i(t) \nabla\phi_i(\boldsymbol{x})^T \nabla\phi_j(\boldsymbol{x}) + f(t, \boldsymbol{x}) \sum_{i=1}^{N} u_i(t)\phi_i(\boldsymbol{x}) \right] d\boldsymbol{x} \\ + \int_{\partial\Omega} \sum_{i=1}^{N} u_i(t)\, \frac{\partial\phi_i}{\partial n}(\boldsymbol{x})\, \phi_j(\boldsymbol{x})\, dS(\boldsymbol{x}). \end{aligned}$$

The final integral is zero as $\phi_j = 0$ on $\partial\Omega$. We set $\sum_{i=1}^{N} u_i(t)\phi_i(\boldsymbol{x}) \approx g(\boldsymbol{x})$ for $\boldsymbol{x} \in \partial\Omega$. If we are using a nodal basis, we can set $u_i(t) = g(\boldsymbol{x}_i)$ where $\boldsymbol{x}_i \in \partial\Omega$ is the "node" for basis function $\phi_i$.

Let $\boldsymbol{u}_h(t)$ be the vector of coefficients $u_i(t)$ where $\phi_i(\boldsymbol{x})$ is zero for $\boldsymbol{x} \in \partial\Omega$. The values of $u_i(t)$ are assumed to be fixed by the boundary conditions if $\phi_i(\boldsymbol{x}) \neq 0$ for some $\boldsymbol{x} \in \partial\Omega$. If we set $\mathcal{N} = \{\, i \mid \phi_i(\boldsymbol{x}) = 0 \text{ for all } \boldsymbol{x} \in \partial\Omega \,\}$, then we obtain the ordinary differential equations

$$\sum_{i\in\mathcal{N}}\left(\int_\Omega \phi_i\,\phi_j\,d\boldsymbol{x}\right)\frac{du_i}{dt}(t) = -\sum_{i\in\mathcal{N}}\left(\int_\Omega \nabla\phi_i^T\nabla\phi_j\,d\boldsymbol{x}\right)u_i(t) - \sum_{i\notin\mathcal{N}}\left(\int_\Omega \nabla\phi_i^T\nabla\phi_j\,d\boldsymbol{x}\right)g_i$$
$$+\int_\Omega f(t,\boldsymbol{x})\phi_j(\boldsymbol{x})\,d\boldsymbol{x} \qquad \text{for } j\in\mathcal{N}. \tag{6.4.3}$$

The matrix with entries $m_{ij} = \int_\Omega \phi_i\,\phi_j\,d\boldsymbol{x},\ i,j\in\mathcal{N}$ is the *mass matrix* $M_h$ (6.3.26), while the matrix $\left[a_{ij} = \int_\Omega \nabla\phi_i^T\nabla\phi_j\,d\boldsymbol{x} \mid i,j\in\mathcal{N}\right]$ is the *stiffness matrix* $A_h$ (6.3.27). Both are symmetric positive semi-definite matrices; the mass matrix is positive definite provided the basis functions $\phi_i, i\in\mathcal{N}$ are linearly independent over $\Omega$.

The resulting ordinary differential equation for $\boldsymbol{u}_h(t)$ is

$$M_h\frac{d\boldsymbol{u}_h}{dt} = -A_h\,\boldsymbol{u} + \boldsymbol{f}_h(t). \tag{6.4.4}$$

For the wave equation, the resulting differential equation is

$$M_h\frac{d^2\boldsymbol{u}_h}{dt^2} = -A_h\,\boldsymbol{u} + \boldsymbol{f}_h(t). \tag{6.4.5}$$

A variant of this approach is the *lumped mass approximation*, which approximates $M_h$ with a diagonal matrix: $M = \text{diag}(m_i \mid i\in\mathcal{N})$ with $m_i \approx \sum_{j=1}^N m_{ij}$, for example.

The eigenvalues of the stiffness matrix $A_h$ approach the eigenvalues of the negative Laplacian $-\nabla^2$; the eigenvalues $\lambda_j$ of $-\nabla^2$ are $\geq 0$ and $\lambda_j\to+\infty$ as $j\to\infty$. The large eigenvalues of $-\nabla^2$ have highly oscillatory eigenfunctions and correspond to high frequency components of the solution. The eigenfunctions associated with large eigenvalues $-\nabla^2$ are hard to approximate, and so the large eigenvalues $\widehat{\lambda}_j$ of $A_h$ will not necessarily be good approximations of $\lambda_j$. However, even in this case, the eigenvectors of $A_h$ with large eigenvalues will also represent highly oscillatory functions.

The theory of diffusion and heat equations can be understood in terms of the eigenvalues $\lambda_j$ and eigenfunctions $\psi_j$ of $-\nabla^2$ on $\Omega$ with homogeneous Dirichlet boundary conditions:

$$-\nabla^2\psi_j = \lambda_j\psi_j \qquad \text{in } \Omega,$$
$$\psi_j(\boldsymbol{x}) = 0 \qquad \text{for } \boldsymbol{x}\in\partial\Omega,$$

for Dirichlet boundary conditions. We order the eigenvalues $0\leq\lambda_1\leq\lambda_2\leq\cdots$. If we write the solution $u(t,\boldsymbol{x}) = \sum_{j=1}^\infty c_j(t)\,\psi_j(\boldsymbol{x})$ then the coefficients $c_j(t)$ are solutions of the differential equations $dc_j/dt = -\lambda_j c_j$; that is, $c_j(t) = c_j(0)\exp(-\lambda_j t)$. Thus, the coefficients $c_j(t)\to 0$ as $t\to+\infty$. Furthermore, the exponential decay is much more rapid for large $j$, and high frequency components are rapidly damped out.

The wave equation, on the other hand, has $u(t,\boldsymbol{x}) = \sum_{j=1}^\infty c_j(t)\,\psi_j(\boldsymbol{x})$ where $d^2c_j/dt^2 = -\lambda_j c_j$ and so $c_j(t) = a_j\cos(\lambda_j^{1/2}t) + b_j\sin(\lambda_j^{1/2}t)$ for suitable con-

stants $a_j$ and $b_j$. Thus the solutions do not become smoother with time in general. In one spatial dimension, the wave equation becomes $\partial^2 u/\partial t^2 = \partial^2 u/\partial x^2$, which has solutions of the form $u_+(x-t) + u_-(x+t)$ for any functions $u_+$ and $u_-$. Apart from momentary cancellation, any roughness in either $u_-$ or $u_+$ persists for all time. In more than one dimension it is possible to have focusing solutions formed by, for example, a radially symmetric wave that arrives at a single point at an instant. For the wave equation $\partial^2 u/\partial t^2 = \nabla^2 u$, the energy $\int_\Omega \frac{1}{2}\left[(\partial u/\partial t)^2 + \|\nabla u\|_2^2\right] d\boldsymbol{x}$ is constant provided either Dirichlet or Neumann boundary conditions hold.

#### 6.4.1.1 Stability Issues

The numerical discretization in time should be appropriate for the type of equation. The fact that some eigenvalues of $A_h$ are large, means the differential equations we obtain are stiff, and that either

**(a):** the step size in time should be limited according to the maximum eigenvalue of $A_h$, **or**

**(b):** implicit methods for the time stepping should be used.

In case (a), the step size is limited due to stability considerations. To see how this works, consider the one-spatial-dimension diffusion $\partial u/\partial t = \partial^2 u/\partial x^2$ and wave equation $\partial^2 u/\partial t^2 = \partial^2 u/\partial x^2$ with Dirichlet boundary conditions $u(0) = u(1) = 0$, discretized using equally spaced piecewise linear elements. In this case,

$$A_h = \frac{1}{h}\begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & -1 & 2 & \ddots & \\ & & \ddots & \ddots & -1 \\ & & & -1 & 2 \end{bmatrix}, \quad M_h = \frac{h}{6}\begin{bmatrix} 4 & 1 & & & \\ 1 & 4 & 1 & & \\ & 1 & 4 & \ddots & \\ & & \ddots & \ddots & 1 \\ & & & 1 & 4 \end{bmatrix},$$

both $(N-1)\times(N-1)$, with $h = 1/N$. The matrices $A_h$ and $M_h$ have common eigenvectors $\boldsymbol{v}_j$ with $(\boldsymbol{v}_j)_\ell = \sin(\pi j \ell/N)$ and eigenvalues $\lambda_j = N(2 - 2\cos(\pi j/N))$ for $A_h$ and $\mu_j = (4 + 2\cos(\pi j/N))/N$ for $M_h$, with $j = 1, 2, \ldots, N-1$. The eigenvalues of $M_h^{-1}A_h$ are then $\lambda_j/\mu_j = N^2(1 - \cos(\pi j/N))/(2 + \cos(\pi j/N))$.

If we use the explicit Euler method (6.1.8) for the diffusion equation, then

$$\boldsymbol{u}_{h,k+1} = (I - (\Delta t)M_h^{-1}A_h)\boldsymbol{u}_{h,k} + (\Delta t)M_h^{-1}\boldsymbol{f}(t_k).$$

Stability of this method then depends on the eigenvalues of the iteration matrix $I - (\Delta t)M_h^{-1}A_h$ which are

$$1 - (\Delta t)N^2\frac{1-\cos(\pi j/N)}{2+\cos(\pi j/N)}, \qquad j = 1, 2, \ldots, N-1.$$

The worst case is at $j = N - 1$ where the eigenvalue of the iteration matrix is

$$1 - (\Delta t)N^2 \frac{1 - \cos(\pi(N-1)/N)}{2 + \cos(\pi(N-1)/N)} \approx 1 - 2(\Delta t)N^2.$$

To keep the magnitude of this eigenvalue less than one, we need $\Delta t < 1/N^2$, which can be a strong limit on the time step.

If we use a different method for the diffusion equation, we need $-2(\Delta t)N^2$ to be inside or very close to the stability region for the method. If $N$ is large, as we need for accurate spatial approximation, then for any explicit method, we need $\Delta t = \mathcal{O}(N^{-2})$. In general, if we use an explicit time-stepping method and a finite element method based on a triangulation $\mathcal{T}_h$, we need $\Delta t = \mathcal{O}(h_{\min}^2)$ where $h_{\min} = \min_{K \in \mathcal{T}_h} h_K$ is the diameter of the smallest element in the triangulation.

While the high frequency components of the solution are forcing a small time step for the sake of stability, our interest is usually in the lower frequency components of the solution which are accurately resolved spatially. The rate at which the lowest frequency component is damped in our one-spatial-dimension problem is $\approx \exp(-\pi^2 t)$ and does not depend significantly on $N$.

Discretizing the wave equation gives

$$M_h \frac{d^2 \boldsymbol{u}_h}{dt^2} = -A_h \boldsymbol{u}_h + \boldsymbol{f}_h(t_k).$$

Writing $\boldsymbol{v}_h = d\boldsymbol{u}_h/dt$ we have the first order system

$$\begin{aligned} \frac{d\boldsymbol{u}_h}{dt} &= \boldsymbol{v}_h, \\ M_h \frac{d\boldsymbol{v}_h}{dt} &= -A_h \boldsymbol{u}_h + \boldsymbol{f}_h(t_k). \end{aligned}$$

Applying the explicit Euler method gives

$$\begin{aligned} \boldsymbol{u}_{h,k+1} &= \boldsymbol{u}_{h,k} + (\Delta t)\boldsymbol{v}_{h,k}, \\ \boldsymbol{v}_{h,k+1} &= \boldsymbol{v}_{h,k} - (\Delta t)M_h^{-1} A_h \boldsymbol{u}_{h,k} + (\Delta t)M_h^{-1} \boldsymbol{f}_h(t_k). \end{aligned}$$

The eigenvalues of the iteration matrix $\begin{bmatrix} I & (\Delta t)I \\ -(\Delta t)M_h^{-1}A_h^{-1} & I \end{bmatrix}$ are the complex numbers $1 \pm i\,(\Delta t)\sqrt{\lambda_j/\mu_j}$; thus the explicit Euler method is always unstable for the wave equation just as it is for simple harmonic motion $d^2u/dt^2 = -\omega^2 u$. We can, however, note that for $j = N - 1$, perturbations in the direction of the corresponding eigenvector are amplified by a factor of $\approx \sqrt{1 + (\Delta t)^2 N^2}$ with each iteration. Even if the method chosen is an explicit method whose stability region includes some of the imaginary axis, such as the original 4th order method of Runge and Kutta (6.1.19), we find that stability requires that $(\Delta t)N$ is bounded. This property is related to,

although not identical with, the *Courant–Friedrichs–Levy* (CFL) *condition* [60] for applying explicit numerical methods to the equation $\partial u/\partial t + a\,\partial u/\partial x = f(t, x)$.

One conclusion of this analysis, for numerical methods applied to wave equations, is that implicit methods should be used. Furthermore, the stability region of the method and its boundary should include the imaginary axis. For example, the implicit Euler method, the implicit trapezoidal rule, the Gauss methods, and the Radau IIA methods are all acceptable for solving the discreteized wave equation. However, the implicit Euler method and the Radau IIA methods are all dissipative and smooth out the waves, while the implicit trapezoidal method and the Gauss Runge–Kutta methods preserve the energy $\int_\Omega \frac{1}{2}\left[v_h^2 + \|\nabla u_h\|_2^2\right] d\boldsymbol{x}$.

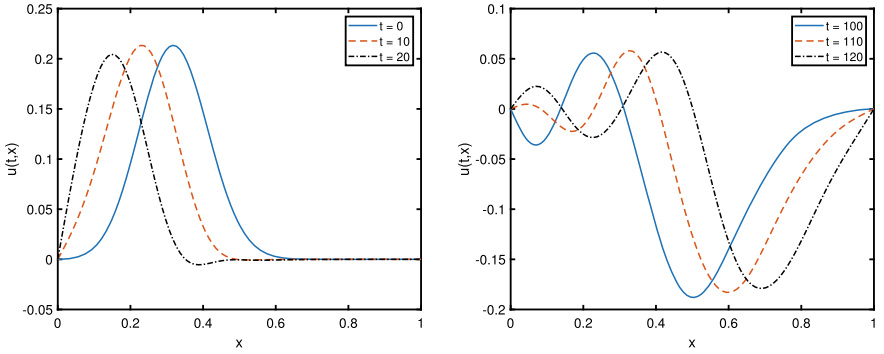
#### 6.4.1.2 Numerical Dispersion

Even if a stable numerical method is used for the wave equation, there are a number of errors that arise in numerical solutions that users should be aware of. One of these is *dispersion*, where different frequency components have different wave speeds. Exact solutions of the wave equation have no dispersion. Dispersion occurs physically for light passing through water or glass, where different colors are refracted into different angles because of the slightly different wave speeds of light of different colors. Other systems where dispersion occurs naturally include cables. This was very important for long-distance telegraphy in the nineteenth century. The *telegraph equations* developed by Oliver Heaviside [121, vol. 2, p. 52] model the voltage and current along a transmission line:

$$\frac{\partial V}{\partial x}(t, x) + L\frac{\partial I}{\partial t}(t, x) = -R\,I(t, x),$$
$$\frac{\partial I}{\partial x}(t, x) + C\frac{\partial V}{\partial t}(t, x) = -G\,V(t, x).$$

These equations show dispersion as well as damping if $LG \neq RC$; in this case, sinusoidal waves of different frequencies traveled along the transmission line with different speeds. The consequence is that the shape of the signal changes as it travels along the transmission line. As a result, what was originally sent as a clear on/off by a telegraph operator at one end of the cable becomes a smeared out oscillation at the receiving end.

Even though the wave equation itself has no dispersion, numerical methods for the wave equation do have dispersion. As an example, Figure 6.4.1 shows snapshots of the results of the implicit trapezoidal method using the finite element method spatially for the wave equation $\partial^2 u/\partial t^2 = \partial^2 u/\partial x^2$ with Dirichlet boundary conditions $u(0) = u(1) = 0$. Piecewise linear elements with $N + 1$ equally spaced nodes $x_j = j\,h$, $j = 0, 1, 2, \ldots, N$, are used for the finite element method. Specifically, we use $N = 10^2$ and time step $\Delta t = 10^{-3}$. The initial conditions are $u(0, x) = x(1 - x)\exp(-a(x - x_0)^2)$ with $a = 50$, and $v(0, x) = -(d/dx)u(0, x)$. The exact solution preserves its




(A) Profiles for $t = 0, 10, 20$ (B) Profiles $t = 100, 110, 120$

**Fig. 6.4.1** Numerical dispersion illustrated for the wave equation solved by the implicit trapezoidal method

shape away from the reflecting ends of the interval $[0, 1]$; each reflection inverts the wave. Because the length of the interval is one, and the wave speed is one, two complete reflections will return the wave to its original shape. In fact, the exact solution is periodic with period two.

Figure 6.4.1(a) shows that the wave speed that arises from the numerical method is not exactly one, the wave speed for the exact wave equation $\partial^2 u/\partial t^2 = \partial^2 u/\partial x^2$. Figure 6.4.1(b) shows the higher frequency components are slightly faster than the lower frequency components of the numerically computed wave.

Higher order methods in both time and space can reduce the amount of dispersion. However, numerical methods of the type we have been discussing, which lead to finite difference schemes that are independent of position and time, cannot eliminate dispersion. In two or higher dimensions, irregular spatial discretizations can lead to other numerical artifacts, such as numerical scattering where a wave passing over a region results in multiple waves moving radially from the irregularity. Dispersion and scattering cannot be eliminated by conventional numerical methods, but users should be aware of the phenomenon, especially for long integration periods.

**Exercises.**

(1) Implement the piecewise linear finite element method for $\partial^2/\partial x^2$ on $\Omega = (0, 1)$ with the boundary conditions $u(t, 0) = u(t, 1) = 0$. Use this to solve $\partial u/\partial t = \partial^2 u/\partial x^2 + f(t, x)$ with these boundary conditions. Do this with spatial grid spacing $h = 1/N$ for $N = 2^j$ with $j = 2, 3, 4, \ldots, 10$. Apply this to the problem where $f(t, x) = 2e^t(1 - 2x)\sin x + 2e^t(1 + x - x^2)\cos x + e^x[(\sin t - \cos t)x^2 + (3\sin t + \cos t)x]$. The exact solution is $u(t, x) = x(1 - x)[e^t \cos x + \sin t\, e^x]$. Use the implicit Euler method to solve the discretized differential equations with $\Delta t = 2^{-\ell}$, $\ell = 1, 2, \ldots, 10$. Plot the logarithm of the

maximum error against $(\log(\Delta t),\ \log N)$ to form a three-dimensional surface plot.

(2) Show that the matrix exponential $e^B = I + B + (1/2!)B^2 + (1/3!)B^3 + \cdots$ converges for any matrix $B$. However, operators like $\nabla^2$ are unbounded; its eigenvalues $\lambda_j \to -\infty$ as $j \to \infty$. Good approximations can be found even for unbounded operators like this using Padé approximations (see Exercise 10) of the exponential function: $e^z \approx r(z) = n(z)/d(z)$ where $n$ and $d$ are polynomials and $r^{(k)}(0) = 1$ for $k = 0, 1, 2, \ldots, m$. In order to handle unbounded operators, we want $r(z) = n(z)/d(z)$ to be bounded as $z \to -\infty$, so that $\deg n \geq \deg d$. Then we can approximate $e^{hB} \approx r(hB)$. Show that $e^{hB} - r(hB) = \mathcal{O}(h^{m+1})$ as $h \to 0$.

(3) If $r(z) = n(z)/d(z)$ is a Padé approximation to $e^z$ then for $B$ symmetric, show that the condition number $\kappa_2(d(hB)) = \max_\lambda |d(h\lambda)| \,/\, \min_\lambda |d(h\lambda)|$ where $\lambda$ ranges over the eigenvalues of $B$. If $\|hB\|_2$ is large and $\deg d$ is large (say, over four), then this condition number can be very large. Instead, it can be very useful to factor $d(z)$ into linear or quadratic factors to avoid the problems of solving a system of equations with large condition number. The Padé approximation of the exponential function with $(\deg n,\ \deg d) = (2,\ 4)$ is

$$e^z \approx \frac{1 + \frac{1}{3}z + \frac{1}{30}z^2}{1 - \frac{2}{3}z + \frac{1}{5}z^2 - \frac{1}{30}z^3 + \frac{1}{360}z^4} = \frac{n(z)}{d(z)}.$$

Factor $d(z) = d_1(z)\, d_2(z)$ into quadratics. Assuming $-B$ is positive semi-definite (so that $\lambda \leq 0$ for any eigenvalue $\lambda$ of $B$), bound $\kappa_2(d_1(hB))$ and $\kappa_2(d_2(hB))$ in terms of $h\,\|B\|_2$.

(4) Consider the system of differential equations

$$M\frac{d\boldsymbol{u}}{dt} = -A\boldsymbol{u} + \boldsymbol{f}(t)$$

with $M$ and $A$ symmetric and positive definite. Show that

$$\frac{d}{dt}\left(\frac{1}{2}\boldsymbol{u}^T M\boldsymbol{u}\right) = -\boldsymbol{u}^T A\boldsymbol{u} + \boldsymbol{u}^T\boldsymbol{f}(t) \leq \boldsymbol{u}^T\boldsymbol{f}(t).$$

If $M$ is the mass matrix and $A$ the stiffness matrix for a partial differential equation, interpret this to show that $\int_\Omega u_h(t,\boldsymbol{x})^2 d\boldsymbol{x} \leq \int_\Omega u_h(0,\boldsymbol{x})^2 d\boldsymbol{x} + \int_0^t \int_\Omega u_h(t,\boldsymbol{x})\ f(t,\boldsymbol{x})\, d\boldsymbol{x}$.

(5) The *sine–Gordon equation* is the differential equation

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} - \sin u.$$

Solve this numerically using the finite element method spatially with piecewise linear elements and the implicit trapezoidal rule in time. Use periodic boundary

conditions $u(t, L) = u(t, -L)$ and $\partial u/\partial x(t, L) = \partial u/\partial x(t, -L)$ with $L = 5$, and spatial grid spacing $h = 1/N$ with $N = 100$. The sine–Gordon equation is notable in that it has wave-like solutions called *solitons* of the form $u_{\text{soliton}}(t, x) = 4\tan^{-1}(\exp(\gamma(x - x_0 - vt)))$ where $\gamma^2(1 - v^2) = 1$. Apply your numerical method to the initial conditions

$$u(0, x) = 4\left[\tan^{-1}(\exp(\gamma(x - x_0))) - \tan^{-1}(\exp(-\gamma(x + x_0)))\right],$$
$$\frac{\partial u}{\partial t}(0, x) = 4v\left[\frac{\exp(\gamma(x - x_0))}{1 + \exp(\gamma(x - x_0))^2} + \frac{\exp(-\gamma(x + x_0))}{1 + \exp(-\gamma(x + x_0))^2}\right],$$

with $x_0 = 2$, $v = \frac{1}{2}$, and $\gamma = 2/\sqrt{3}$. Compare your numerical results to the exact soliton solution. [**Note:** To solve the implicit trapezoidal equations for the updated velocity $\boldsymbol{v}_{k+1}$ in terms of the previous pair $(\boldsymbol{u}_k, \boldsymbol{v}_k)$ can be reduced to $(M_h + \frac{1}{4}(\Delta t)^2 A_h)\boldsymbol{v}_{k+1} = (M_h - \frac{1}{4}(\Delta t)^2 A_h)\boldsymbol{v}_k - (\Delta t)A_h\boldsymbol{u}_k - (\Delta t)\varphi(\boldsymbol{u}_k + \frac{1}{4}\Delta t(\boldsymbol{v}_k + \boldsymbol{v}_{k+1}))$ where $\varphi(\boldsymbol{z})_i = \sin(z_i)$. The fixed point iteration $(M_h + \frac{1}{4}(\Delta t)^2 A_h)\boldsymbol{v}_{k+1}^{(\ell+1)} = (M_h - \frac{1}{4}(\Delta t)^2 A_h)\boldsymbol{v}_k - (\Delta t)A_h\boldsymbol{u}_k - (\Delta t)\varphi(\boldsymbol{u}_k + \frac{1}{4}\Delta t(\boldsymbol{v}_k + \boldsymbol{v}_{k+1}^{(\ell)}))$, $\ell = 0, 1, 2, \ldots$ converges rapidly for small $\Delta t$, and only requires $M_h + \frac{1}{4}(\Delta t)^2 A_h$ to be factored once.]

(6) The *shallow water equations* in two dimensions are an approximation to the Navier–Stokes equations for fluid mechanics where a viscousless fluid moves over a two-dimensional surface with varying depth $H(\boldsymbol{x})$ creating waves of height $h(t, \boldsymbol{x})$ with $|h(t, \boldsymbol{x})| \ll H(\boldsymbol{x})$ and the depth is much smaller than the horizontal scale. These equations, ignoring Coriolis effects due to the rotation of the Earth, can be written as

$$\frac{\partial^2 h}{\partial t^2} = \operatorname{div}(g\, H(\boldsymbol{x})\, \nabla h) - b\frac{\partial h}{\partial t} \qquad \text{over } \Omega \subset \mathbb{R}^2,$$
$$\frac{\partial h}{\partial n} = 0 \qquad \text{on } \partial\Omega.$$

Here $g$ is the gravitational acceleration, $b$ is a viscous damping parameter, and $\partial h/\partial n$ is the normal derivative of $h$ on the boundary of $\Omega$. Implement this using the finite element method with piecewise linear elements for the spatial discretization and the implicit trapezoidal rule in time. Test your method on $\Omega = \left\{ (x, y) \mid y \geq x^4 \right\}$ with $h(0, \boldsymbol{x}) = \exp(-\alpha \|\boldsymbol{x} - \boldsymbol{e}_2\|_2^2)$ and $\partial h/\partial t\,(0, \boldsymbol{x}) = 0$ with $\alpha = 100$.

(7) The time-dependent *Schrödinger equation* of quantum mechanics has the form

$$-i\hbar\frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m}\nabla^2\psi + V(\boldsymbol{x})\psi$$

where $\hbar = h/(2\pi)$ is the reduced Planck constant, $m$ the mass of the particle, and $V(\boldsymbol{x})$ the potential energy function. If we create a spatial discretization

$\psi(t, \boldsymbol{x}) = \sum_{k=1}^{N} \psi_k(t)\, \phi_k(\boldsymbol{x})$ and we apply the Galerkin approximation, show that the resulting system of ordinary differential equations is

$$-i\hbar M \frac{d\boldsymbol{\psi}}{dt} = \frac{\hbar^2}{2m} A\boldsymbol{\psi} + \widetilde{V}\boldsymbol{\psi}$$

where $m_{k\ell} = \int_{\mathbb{R}^3} \phi_k \phi_\ell \, d\boldsymbol{x}$, $a_{k\ell} = \int_{\mathbb{R}^3} \nabla\phi_k^T \nabla\phi_\ell \, d\boldsymbol{x}$, and $\widetilde{v}_{k\ell} = \int_{\mathbb{R}^3} V(\boldsymbol{x})\phi_k(\boldsymbol{x})\phi_\ell(\boldsymbol{x})\, d\boldsymbol{x}$. Show that, provided that for each basis function $\phi_k$ we have $\int_{\mathbb{R}^3} \phi_k^2 \, d\boldsymbol{x}$ finite, each of $M$, $A$, and $\widetilde{V}$ are symmetric while $M$ and $A$ are also positive definite. What are the possible exponents $\alpha$ where $\psi(t, \boldsymbol{x}) = \exp(\alpha t)\, \widehat{\phi}(\boldsymbol{x})$ is a solution of the Schrödinger equation for some $\widehat{\phi}$? Given that the eigenvalues of $M^{-1}A$ are typically large, what regions should be contained in the stability region of the ODE method used?

## Projects

(1) Write code to find geodesics on a surface given by $S = \{\, \boldsymbol{x} \mid g(\boldsymbol{x}) = 0 \,\} \subset \mathbb{R}^n$ with $\nabla g(\boldsymbol{x}) \neq \boldsymbol{0}$ for every $\boldsymbol{x} \in S$. Do this by solving the geodesic equation

$$\frac{d^2\boldsymbol{x}}{dt^2} = -\lambda\, \nabla g(\boldsymbol{x}),$$
$$\lambda = \frac{(d\boldsymbol{x}/dt)^T \operatorname{Hess} g(\boldsymbol{x})(d\boldsymbol{x}/dt)}{\|\nabla g(\boldsymbol{x})\|_2^2},$$

as a two-point boundary value problem with boundary conditions $\boldsymbol{x}(a) = \boldsymbol{x}_0$ and $\boldsymbol{x}(b) = \boldsymbol{x}_1$. We can solve this using a shooting method to find $\boldsymbol{v}_0 := d\boldsymbol{x}/dt(a)$. The constraint that $g(\boldsymbol{x}(t)) = 0$ for all $t$ means that $\nabla g(\boldsymbol{x}_0)^T \boldsymbol{v}_0 = 0$. Use the variational equations for the geodesic equations (which involves 3rd derivatives of $g$!) to create the Newton equations. Since the Newton equations with the condition $\nabla g(\boldsymbol{x}_0)^T \boldsymbol{v}_0 = 0$ is over-determined, use the least squares solution (that is, use the pseudo-inverse (2.5.10) instead of the inverse). Apply this to finding geodesics on ellipsoids $(x/a)^2 + (y/b)^2 + (z/c)^2 = 1$ with $a \neq b \neq c$.

(2) Implement a basic finite element system based on piecewise linear elements over a triangulation in two dimensions. The triangulation $\mathcal{T}$ is represented by a pair of arrays: the vertices of the triangulation are given by an array $p[i, j] = (\boldsymbol{p}_j)_i$ (the $i$th coordinate of the $j$th point) and $j = t[r, s]$ being the index in $p$ of the $s$th vertex of the $r$th triangle. Thus, $p$ is a $2 \times n$ array of real numbers and $t$ is an $m \times 3$ array of integers where $n$ is the number of points and $m$ is the number of triangles. Assume that the region $\Omega$ is the union of the triangles in the triangulation. Write code to compute

$$a_{k\ell} = \int_\Omega \left[\alpha(\boldsymbol{x})\nabla\phi_k(\boldsymbol{x})^T\nabla\phi_\ell(\boldsymbol{x}) + \boldsymbol{\beta}(\boldsymbol{x})^T\nabla\phi_k(\boldsymbol{x})\phi_\ell(\boldsymbol{x}) + \gamma(\boldsymbol{x})\phi_k(\boldsymbol{x})\phi_\ell(\boldsymbol{x})\right] d\boldsymbol{x},$$
$$f_\ell = \int_\Omega f(\boldsymbol{x})\,\phi_\ell(\boldsymbol{x})\,d\boldsymbol{x}$$

for $k, \ell = 1, 2, \ldots, n$, given functions $\alpha$, $\boldsymbol{\beta}$, $\gamma$, and $f$. Here $\phi_k$ is the piecewise linear function with $\phi_k(\boldsymbol{p}_j) = 1$ if $k = j$ and zero if $k \neq j$. Note that $\int_\Omega = \sum_K \int_K$ where $K$ ranges over triangles in the triangulation. For triangle $K$ with vertices $\boldsymbol{p}_j$, $\boldsymbol{p}_k$, $\boldsymbol{p}_\ell$, $\phi_r$ is zero on $K$ provided $r \neq j, k$ or $\ell$. Check that $a_{k\ell} = 0$ if $\boldsymbol{p}_k$ and $\boldsymbol{p}_\ell$ are *not* both vertices of the same triangle $K$. Thus $A$ is a sparse matrix. It can be constructed by summing over triangles:

$A \leftarrow 0$; $\boldsymbol{f} \leftarrow \boldsymbol{0}$
`for` $K \in \mathcal{T}$
  $\mathcal{J} \leftarrow$ `vertices(`$K$`) (set of vertex indices)`
  `for` $k, \ell \in \mathcal{J}$
    $a_{k\ell} \leftarrow a_{k\ell} + \int_K \left[\alpha(\boldsymbol{x})\nabla\phi_k(\boldsymbol{x})^T\nabla\phi_\ell(\boldsymbol{x}) + \cdots\right] d\boldsymbol{x}$
    $f_\ell \leftarrow f_\ell + \int_K f(\boldsymbol{x})\,\phi_\ell(\boldsymbol{x})\,d\boldsymbol{x}$
  `end for`
`end for`

The integrals can be computed by a numerical integration method of your choice, preferably one that is exact for quadratic functions at least.

Boundary vertices and boundary edges can be determined as follows: each boundary edge is an edge of exactly one triangle; a boundary vertex is an endpoint of at least one boundary edge. Normally, every boundary vertex is the endpoint of exactly two boundary edges. We set $\boldsymbol{u}_{\partial\Omega} = [u_\ell \mid \boldsymbol{x}_\ell \in \partial\Omega]$, and $\boldsymbol{u}_{\text{int}\Omega} = [u_\ell \mid \boldsymbol{x}_\ell \notin \partial\Omega]$. Similarly we let $A_{\text{int}\Omega,\text{int}\Omega} = [a_{k\ell} \mid \boldsymbol{x}_k, \boldsymbol{x}_\ell \notin \partial\Omega]$, $A_{\text{int}\Omega,\partial\Omega} = [a_{k\ell} \mid \boldsymbol{x}_k \notin \Omega,\ \boldsymbol{x}_\ell \notin \partial\Omega]$, etc.

For setting Dirichlet boundary conditions, such as $u(x, y) = g(x, y)$ for $(x, y) \in \partial\Omega$ where $g$ is not piecewise linear we can set up a least squares approximation as follows: create a "boundary mass matrix" $\widetilde{m}_{k\ell} = \int_{\partial\Omega} \phi_k(\boldsymbol{x})\phi_\ell(\boldsymbol{x})\,dS(\boldsymbol{x})$ and the vector $\widetilde{b}_\ell = \int_{\partial\Omega} \phi_\ell(\boldsymbol{x})\,g(\boldsymbol{x})\,d\boldsymbol{x}$. Then the boundary values $u_\ell$, where $\boldsymbol{x}_\ell$ is a boundary node, can be computed by solving $\widetilde{M}\boldsymbol{u}_{\partial\Omega} = \widetilde{\boldsymbol{b}}$. The partial differential equation with Dirichlet boundary conditions can then be solved approximately by solving $A_{\text{int}\Omega,\text{int}\Omega}\boldsymbol{u}_{\text{int}\Omega} = \boldsymbol{f}_{\text{int}\Omega} - A_{\text{int}\Omega,\partial\Omega}\boldsymbol{u}_{\partial\Omega}$ where $\boldsymbol{u}_{\partial\Omega}$ has already been computed.

If you are feeling ambitious, extend your code to also handle quadratic Lagrange elements (see Figure 4.3.6(a)).

# Chapter 7
# Randomness


## 7.1 Probabilities and Expectations

Probabilities have been used since the ninth century by Arab scholars studying cryptography [34]. Games of chance were the motivation for Europeans to start studying probabilities with the work of Gerolamo Cardano [48], Pierre de Fermat, and Blaise Pascal in the seventeenth century [169, 218]. The Dutch physicist Christiaan Huygens was also in this tradition of studying games of chance, writing a textbook about it [130] (1657). Jakob Bernoulli (1654–1705) and Abraham de Moivre (1667–1754) also contributed to the development of the theory. In the nineteenth century, Pierre de Laplace (1749–1827) contributed greatly to the theory with his work *Théorie Analytique des Probabilités* (1812) [154]. From the late nineteenth century to the twentieth century contributors include Chebyshev, Markov, and Kolmogorov. It is Kolmogorov that created the modern theory of probability with his 1933 monograph *Foundations of Probability Theory* [147]. We start with Kolmogorov's formalism.

### *7.1.1 Random Events and Random Variables*

Kolmogorov's insight was that we need to start with a space of "all things that can possibly happen" called an *event space* $\Omega$. If we were dealing with Monopoly games, then this space would include all possible Monopoly games with details down to the level of the individual dice rolls. In general, this event space may be discrete or continuous, or some combination of the two.

With an event space $\Omega$, we do not necessarily assign a probability to each individual $\omega \in \Omega$; for continuous quantities, the probability of a single specific $\omega \in \Omega$ would be zero. (Example: What is the probability that a number, chosen at random between zero and one, is exactly $1/2$? If we start writing out a decimal expansion of a random number, the probability of getting "5" followed by infinitely many "0"'s is zero.) Instead Kolmogorov leaned on the theory of Lebesgue integration and



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_7

measure theory: there must be a collection $\mathfrak{B}$ of "measurable" subsets of $\Omega$ that forms a $\sigma$-algebra of sets. That is,

- $\emptyset \in \mathfrak{B}$,
- $B_1, B_2, \ldots \in \mathfrak{B}$ implies $\bigcap_{i=1}^{\infty} B_i \in \mathfrak{B}$ and $\bigcup_{i=1}^{\infty} B_i \in \mathfrak{B}$, and
- $B \in \mathfrak{B}$ implies $\Omega \backslash B \in \mathfrak{B}$.

Every $B \in \mathfrak{B}$ is a subset of $\Omega$ called an *event*. Every event is assigned a probability $\Pr[B]$, and Pr is a *probability measure* on $\Omega$. That is,

$$\begin{aligned} \Pr\left(\bigcup_{i=1}^{\infty} B_i\right) &= \sum_{i=1}^{\infty} \Pr(B_i) \qquad \text{provided } B_i \cap B_j = \emptyset \text{ for } i \neq j, \\ \Pr(\Omega) &= 1, \\ \Pr(B) &\geq 0 \qquad \text{for all } B \in \mathfrak{B}. \end{aligned}$$

A consequence of the properties listed for a probability measure is that $\Pr(\emptyset) = 0$.

A *random variable* $X$ with values in a measurable set $A$ is defined as a measurable function $X : \Omega \to A$; that the function is measurable means that for every measurable set $F \subseteq A$, the set $\{\, \omega \mid X(\omega) \in F \,\} \in \mathfrak{B}$ and is measurable in $\Omega$. The event "$X(\omega) \in F$" is often simply written "$X \in F$", which has the probability

$$\Pr(\{\omega \mid X(\omega) \in F\}) = \Pr(X \in F).$$

Two random variables $X : \Omega \to A$ and $Y : \Omega \to B$ are *independent* if for any measurable $E \subseteq A$ and $F \subseteq B$,

$$\Pr(X \in E \& Y \in F) = \Pr(X \in E) \cdot \Pr(Y \in F). \tag{7.1.1}$$

Two consecutive dice rolls are considered independent as the outcome of one (apparently) does not affect the outcome of the other. Random variables that are related are not independent, but we can talk about the conditional probability of one event given another. Consider the outcome of getting an "A" on Calculus I and getting an "A" on Calculus II. While getting an "A" on one does not guarantee getting it on the other, it certainly makes it more likely. To measure this, we use *conditional probabilities*:

$$\Pr(X \in E \mid Y \in F) = \frac{\Pr(X \in E \& Y \in F)}{\Pr(Y \in F)}. \tag{7.1.2}$$

Provided $\Pr(Y \in F) > 0$ this gives probabilities "assuming that $Y \in F$". A little care must be taken with this concept in the case of continuous random variables: if $Y$ is, for example, uniformly distributed on an interval $[a, b]$ with $a < b$, then $\Pr(Y = c) = 0$ for any $c \in [a, b]$. But we can use $\Pr(Y \in [c,\ c + \delta])$ for $\delta > 0$ and define

$$\begin{aligned}\Pr(X \in E \mid Y = c) &= \lim_{\delta \downarrow 0} \Pr(X \in E \mid Y \in [c,\ c+\delta]) \\ &= \lim_{\delta \downarrow 0} \frac{\Pr(X \in E \& Y \in [c,\ c+\delta])}{\Pr(Y \in [c,\ c+\delta])}. \end{aligned} \tag{7.1.3}$$

The *probability distribution* of $X$ is the probability measure $\pi_X$ given by

$$\pi_X(F) = \Pr(X \in F). \tag{7.1.4}$$

Note that $\pi_X$ is a probability measure over the set of possible values $A = \text{range}(X) = \{ X(\omega) \mid \omega \in \Omega \}$ of $X$, not over $\Omega$. If $\text{range}(X) = \mathbb{R}$, then we have the possibility that $\pi_X$ might be represented by a *probability density function* or *pdf* $p_X$:

$$\pi_X(F) = \int_F p_X(x)\, dx. \tag{7.1.5}$$

The probability distribution of a random variable may be a sum of Dirac $\delta$-functions for a discrete probability distribution, or represented by a probability density function for a continuous probability distribution.

We write $X \sim \pi$ where $\pi$ is a probability measure to mean

$$\Pr(X \in E) = \pi(E) \qquad \text{for all } E \text{ measurable in range}(X).$$

We then say that $\pi$ is the *probability distribution* of $X$.

Examples of probability distributions include:

- *Bernoulli distribution*: $\text{range}(X) = \{0, 1\}$ and if $p = \Pr(X = 0)$ then $1 - p = \Pr(X = 1)$. The probability measure of $X$ is $\pi_X(\{0\}) = 1 - p$ and $\pi_X(\{1\}) = p$. If $X$ is the result of a fair coin toss (heads for one, tails for zero) then the probability distribution is a Bernoulli distribution with $p = 1/2$.
  We write $X \sim \text{Bernoulli}(p)$.
- *Binomial distribution*: $\text{range}(X) = \{0, 1, 2, \ldots, n\}$ with $\Pr(X = k) = \binom{n}{k} p^k (1-p)^{n-k}$. This is the sum of $n$ independent random variables $X_i$, $i = 1, 2, \ldots, n$, each of which has the Bernoulli probability distribution with $\Pr(X_i = 1) = p$.
  We write $X \sim \text{Binom}(n, p)$.
- *Poisson distribution*: $\text{range}(X) = \{0, 1, 2, 3, \ldots\}$ with $\Pr(X = k) = e^{-\lambda}(\lambda^k / k!)$ for a given parameter $\lambda > 0$. This corresponds to the limit as $n \to \infty$ of a sum of $n$ independent Bernoulli random variables $X_i$, each with $\Pr(X_i = 1) = \lambda / n$.
  We write $X \sim \text{Poisson}(\lambda)$.
- *Uniform distribution*: $\text{range}(X) = [a, b]$ with constant probability density function $p_X(x) = 1/(b-a)$ for $a \le x \le b$.
  We write $X \sim \text{Uniform}(a, b)$.

- *Exponential distribution*: $\text{range}(X) = [0, \infty)$ with probability density function $p_X(x) = \alpha\, e^{-\alpha x}$ for $x \geq 0$. This describes the waiting time for a repeatable event to occur the first time, where the event could occur equally likely at any moment. We write $X \sim \text{Exponential}(\alpha)$.
- *Normal (or Gaussian) distribution*: $\text{range}(X) = \mathbb{R}$ with probability density function $p_X(x) = (2\pi)^{-1/2}\sigma^{-1} \exp(-(x-\mu)^2/(2\sigma^2))$. This distribution is famous as the "bell-shaped curve" and is important in many applications because of the Central Limit Theorem (Theorem 7.4).
  We write $X \sim \text{Normal}(\mu, \sigma^2)$.
  The multivariate normal distribution over $\mathbb{R}^n$ has probability distribution function $p_{\boldsymbol{X}}(\boldsymbol{x}) = (2\pi)^{-d/2}(\det V)^{-1/2} \exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T V^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))$ where $\boldsymbol{\mu} = \mathbb{E}[\boldsymbol{X}]$ and $V$ is the variance-covariance matrix: $V = \mathbb{E}\left[(\boldsymbol{X}-\boldsymbol{\mu})(\boldsymbol{X}-\boldsymbol{\mu})^T\right]$.
  We write $\boldsymbol{X} \sim \text{Normal}(\boldsymbol{\mu}, V)$.

There are, of course, many other important probability distributions. This is simply a selection of some of the most important ones.

Note that the conditional probability $\Pr(X \in E \mid Y = y)$ in (7.1.3) can be defined implicitly via

$$\Pr(X \in E \mid Y \in F) = \int_F \Pr(X \in E \mid Y = y)\, \pi_Y(dy), \tag{7.1.6}$$

where $\pi_Y$ is the probability distribution of $Y$.

If $X$ is a real-valued random variable, then the *cumulative distribution function* of $X$ is $\text{cdf}_X : \mathbb{R} \to [0, 1]$ where

$$\text{cdf}_X(s) = \Pr(X \leq s). \tag{7.1.7}$$

If $X$ has a probability density function $p_X$ then $\text{cdf}_X(s) = \int_{-\infty}^{s} p_X(x)\, dx$. Note that we can recover $p_X$ from $\text{cdf}_X$ as $p_X(s) = \text{cdf}_X'(s)$. Cumulative distribution functions can be useful, for example, to find the probability distributions of the maximum of two independent random variables:

$$\begin{aligned}\text{cdf}_{\max(X,Y)}(s) &= \Pr(\max(X, Y) \leq s) = \Pr(X \leq s \,\&\, Y \leq s) \\ &= \Pr(X \leq s) \cdot \Pr(Y \leq s) = \text{cdf}_X(s) \cdot \text{cdf}_Y(s).\end{aligned}$$

Random variables can be transformed by functions: $Y = f(X)$ is a new random variable obtained from $X$ where $f : \mathbb{R} \to \mathbb{R}$ is a deterministic (that is, not random) function. Then

$$\pi_Y(F) = \Pr(Y \in F) = \Pr(f(X) \in F) = \Pr(X \in f^{-1}(F)) = \pi_X(f^{-1}(F))$$

where $f^{-1}(F) = \{\, x \mid f(x) \in F \,\}$. If $f$ is an increasing function, $f^{-1}$ is a function, and

$$\text{cdf}_Y(s) = \Pr(Y \leq s) = \Pr(f(X) \leq s) = \Pr(X \leq f^{-1}(s)) = \text{cdf}_X(f^{-1}(s)).$$

### 7.1.2 Expectation and Variance

If $X$ has real values (that is, $X\colon \Omega \to \mathbb{R}$), then the *expected value* of $X$ is

$$\mathbb{E}[X] = \int_{\Omega} X(\omega)\,\Pr(d\omega), \tag{7.1.8}$$

provided $X$ is an integrable function with respect to the Pr measure. In that case we can also express the expected value as the integral

$$\mathbb{E}[X] = \int_{-\infty}^{+\infty} x\,\pi_X(dx), \tag{7.1.9}$$

and if $X$ has a probability density function,

$$\mathbb{E}[X] = \int_{-\infty}^{+\infty} x\,p_X(x)\,dx. \tag{7.1.10}$$

For a discrete random variable with values $x_1, x_2, \ldots,$

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} x_i \Pr(X = x_i). \tag{7.1.11}$$

The value $\mathbb{E}[X]$ is also called the *mean* of $X$.

If we wish to denote the expectation with respect to a given probability distribution $\pi$ or probability density function $p$, we use the notation

$$\mathbb{E}_{X\sim\pi}[f(X)] \qquad \text{or} \qquad \mathbb{E}_{X\sim p}[f(X)].$$

There are random variables for which there is no mean (or that it is infinite). Consider the random variable $X$ which has the value $X = 2^k$ with probability $2^{-k}$ for $k = 1, 2, 3, \ldots$. This is a discrete probability distribution, so

$$\mathbb{E}[X] = \sum_{k=1}^{\infty} 2^k \times 2^{-k} = +\infty.$$

It should be noted that expectation is linear, just as integration is linear:

$$\mathbb{E}[\alpha X + \beta Y] = \alpha\,\mathbb{E}[X] + \beta\,\mathbb{E}[Y]$$

for non-random $\alpha$ and $\beta$.

The *variance* of the random variable $X$ is defined as

$$\mathbb{V}\mathrm{ar}\,[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}\left[X^2\right] - \mathbb{E}\,[X]^2 \geq 0.$$

The *standard deviation* of $X$ is

$$\mathrm{stddev}\,[X] = \sqrt{\mathbb{V}\mathrm{ar}\,[X]}.$$

The standard deviation is useful as it has the same physical units as $X$, and scales the same way: $\mathrm{stddev}\,[\alpha X] = |\alpha|\,\mathbb{E}\,[X]$, while $\mathbb{V}\mathrm{ar}\,[\alpha X] = \alpha^2 \mathbb{V}\mathrm{ar}\,[X]$.

A very important result is that for *independent* random variables $X$ and $Y$, $\mathbb{V}\mathrm{ar}\,[X + Y] = \mathbb{V}\mathrm{ar}\,[X] + \mathbb{V}\mathrm{ar}\,[Y]$.

**Theorem 7.1** *If $X$ and $Y$ are independent random variables for which the expected values of both $X^2$ and $Y^2$ are finite, then*

$$\mathbb{V}\mathrm{ar}\,[X + Y] = \mathbb{V}\mathrm{ar}\,[X] + \mathbb{V}\mathrm{ar}\,[Y]\,. \tag{7.1.12}$$

Before we prove that variances of sums of independent random variables add, we need a Lemma that is useful in other circumstances.

**Lemma 7.2** *If $X$ and $Y$ are independent random variables for which $\mathbb{E}\,[X \cdot Y]$ is defined (for example if $X^2$ and $Y^2$ both have finite expected value),*

$$\mathbb{E}\,[X \cdot Y] = \mathbb{E}\,[X] \cdot \mathbb{E}\,[Y]\,. \tag{7.1.13}$$

***Proof*** We first define the joint probability distribution $\pi_{X,Y}$ via

$$\pi_{X,Y}(E \times F) = \Pr(X \in E \& Y \in F),$$

which is a measure over $\mathbb{R} \times \mathbb{R}$. Then

$$\mathbb{E}\,[X \cdot Y] = \int_{\mathbb{R}\times\mathbb{R}} x\, y\, \pi_{X,Y}(dx, dy).$$

Since $|x\, y| \leq \frac{1}{2}(x^2 + y^2)$, provided $\mathbb{E}\left[X^2\right]$ and $\mathbb{E}\left[Y^2\right]$ are both finite, $\mathbb{E}\,[X \cdot Y]$ is also finite. But if $X$ and $Y$ are independent,

$$\begin{aligned}\pi_{X,Y}(E \times F) &= \Pr(X \in E \& Y \in F) = \Pr(X \in E) \cdot \Pr(Y \in F) \\ &= \pi_X(E) \cdot \pi_Y(F).\end{aligned}$$

Thus

$$\mathbb{E}[X \cdot Y] = \int_{\mathbb{R}\times\mathbb{R}} x\, y\, \pi_{X,Y}(dx, dy) = \int_{\mathbb{R}}\int_{\mathbb{R}} x\, y\, \pi_X(dx)\, \pi_Y(dy)$$
$$= \int_{\mathbb{R}} x\, \pi_X(dx) \int_{\mathbb{R}} y\, \pi_Y(dy) = \mathbb{E}[X] \cdot \mathbb{E}[Y],$$

as we wanted.

Now we can proceed with the proof of the main result.

***Proof*** If $\mathbb{V}\text{ar}[X]$ and $\mathbb{V}\text{ar}[Y]$ are both finite and $X,\ Y$ independent, then

$$\begin{aligned}
\mathbb{V}\text{ar}[X+Y] &= \mathbb{E}\left[(X+Y)^2 - \mathbb{E}[X+Y]^2\right] \\
&= \mathbb{E}\left[X^2 + 2XY + Y^2 - \mathbb{E}[X]^2 - 2\mathbb{E}[X]\,\mathbb{E}[Y] - \mathbb{E}[Y]^2\right] \\
&= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 + \mathbb{E}\left[Y^2\right] - \mathbb{E}[Y]^2 \\
&\quad + 2\mathbb{E}[XY] - 2\mathbb{E}[X]\,\mathbb{E}[Y] \\
&\qquad \text{then by Lemma 7.2,} \\
&= \mathbb{V}\text{ar}[X] + \mathbb{V}\text{ar}[Y] + 2\mathbb{E}[X]\,\mathbb{E}[Y] - 2\mathbb{E}[X]\,\mathbb{E}[Y] \\
&= \mathbb{V}\text{ar}[X] + \mathbb{V}\text{ar}[Y],
\end{aligned}$$

as we wanted.

The variance can be used to bound how much variation or randomness a random variable $X$ has around its mean. One bound comes from Chebyshev's bounds [246]:

**Theorem 7.3** *If $X$ is a random variable with finite variance, then for $\mu = \mathbb{E}[X]$ and $\sigma = stddev[X]$, we have for any $k > 0$,*

$$\Pr(|X - \mu| \geq k\,\sigma) \leq 1/k^2. \tag{7.1.14}$$

***Proof*** Noting that

$$\begin{aligned}
\sigma^2 = \mathbb{V}\text{ar}[X] &= \int_{\mathbb{R}} (x-\mu)^2\, \pi_X(dx), \qquad \text{dividing by } \sigma^2, \\
1 &= \int_{\mathbb{R}} \left(\frac{x-\mu}{\sigma}\right)^2 \pi_X(dx) \\
&\geq \int_{(-\infty,\mu-k\sigma]\cup[\mu+k\sigma,+\infty)} \left(\frac{x-\mu}{\sigma}\right)^2 \pi_X(dx) \\
&\geq \int_{(-\infty,\mu-k\sigma]\cup[\mu+k\sigma,+\infty)} k^2\, \pi_X(dx) = k^2 \Pr\left(\frac{|X-\mu|}{\sigma} \geq k\right).
\end{aligned}$$

Rearranging gives

$$\Pr(|X - \mu| \geq k\,\sigma) \leq 1/k^2,$$

as we wanted.

### 7.1.3 Averages

The expected value of a random variable $X$ is given as an integral

$$\mathbb{E}[X] = \int_{\mathbb{R}} x\, \pi_X(dx) = \mu.$$

We can approximate it by the average of *independent* samples from the same distribution $X_k \sim \pi_X$ for all $k$:

$$A_n = \frac{1}{n}(X_1 + X_2 + \cdots + X_n).$$

The average $A_n$ is itself a random variable. While

$$\mathbb{E}[A_n] = \frac{1}{n}\left(\mathbb{E}[X_1] + \mathbb{E}[X_2] + \cdots + \mathbb{E}[X_n]\right) = \mathbb{E}[X],$$

the main question here is: How close $A_n$ is to $\mu$? The *Laws of Large Numbers* show that $A_n$ converges to the mean $\mu$ under mild assumptions: the weak Law of Large Numbers shows that, provided $\mathbb{E}[|X|]$ is finite, for $\epsilon > 0$ we have

$$\Pr(|A_n - \mu| > \epsilon) \to 0 \qquad \text{as } n \to \infty. \tag{7.1.15}$$

The strong Law of Large Numbers states that assuming that $\mathbb{E}[|X|]$ is finite,

$$\Pr\left(\lim_{n\to\infty} A_n = \mu\right) = 1. \tag{7.1.16}$$

If $\mathbb{V}\mathrm{ar}[X]$ is finite, then we have more detail about the variation of $A_n$ around the mean $\mu$ from the Central Limit Theorem:

**Theorem 7.4** *Provided the random variables $X_k$ are independent and identically distributed with finite variance $V$, the probability distribution $\pi_n$ of*

$$S_n := \sqrt{n}\,(A_n - \mu) = \frac{\sum_{k=1}^{n} X_k - n\,\mu}{\sqrt{n}}$$

*converges to the normal distribution in the sense that for any bounded continuous function $\varphi$,*

$$\mathbb{E}[\varphi(S_n)] \to \int_{-\infty}^{+\infty} \varphi(s)\, \frac{\exp(-s^2/(2V))}{\sqrt{2\pi}}\, ds \qquad \text{as } n \to \infty.$$

**Exercises.**

(1) Show that if $X_1 \sim \text{Poisson}(\lambda_1)$ and $X_2 \sim \text{Poisson}(\lambda_2)$ are independent, then $X_1 + X_2 \sim \text{Poisson}(\lambda_1 + \lambda_2)$.

(2) Suppose that $X_1, X_2, \ldots, X_n$ are independent random variables, but all are distributed according to a cumulative probability function $\Pr[X_k \leq x] = F(x)$. Show that

$$\Pr[\max\{X_k \mid 1 \leq k \leq n\} \leq x] = F(x)^n.$$

(3) Suppose that $X \sim \text{Uniform}(a, b)$. What are $\mathbb{E}[X]$, $\mathbb{V}\text{ar}[X]$ and $\mathbb{E}\left[X^2\right]$?

(4) For a random variable $X$ with real values the function $\chi_X(t) = \mathbb{E}\left[e^{itX}\right]$ is called the *characteristic function* of $X$. Show that the characteristic function is the Fourier transform of the probability density function of $X$, assuming that $X$ has a density function. Show that if $X$ and $Y$ are independent random variables, then $\chi_{X+Y}(t) = \chi_X(t)\,\chi_Y(t)$. Also show that $\chi_{aX}(t) = \chi_X(at)$ and $\chi_{X+b}(t) = e^{ibt}\,\chi_X(t)$.

(5) Show that $\mathbb{E}[X] = -i\,\chi_X'(0)$, and $\mathbb{V}\text{ar}[X] = -\chi_X''(0) + \chi_X'(0)^2$.

(6) Given random variables $X$ and $Y$ with real values, show that $X$ and $Y$ are independent only if $\mathbb{E}\left[e^{i(sX+tY)}\right] = \chi_X(s)\,\chi_Y(t)$ for all real $s$ and $t$. [**Note:** The converse is also true, but takes more work.]

(7) Show that the characteristic function of $\alpha X$ for a random variable $X$ is $\chi_{\alpha X}(t) = \chi_X(\alpha t)$. Use this to show that if $X_i \sim \text{Uniform}(-1, 1)$ for $i = 1, 2, \ldots, n$ are independent, and $S_n = (1/n)\sum_{i=1}^n X_i$, then $\chi_{S_n}(t) = [\sin(t/n)/(t/n)]^n$. Using $(\sin u)/u = 1 - \frac{1}{6}u^2 + \mathcal{O}(u^4)$, show that $\chi_{S_n}(t) \to 1$ as $n \to \infty$. Also show that $\chi_{\sqrt{n}S_n}(t) \to \exp(-t^2/6)$ as $n \to \infty$ showing that $\sqrt{n}S_n$ has an asymptotically normal distribution as $n \to \infty$.

(8) Show that if $X \sim \text{Normal}(\mu, \sigma^2)$ then $\chi_X(t) = \exp(i\mu t - \frac{1}{2}\sigma^2 t^2)$. [**Hint:** If $Z = (X - \mu)/\sigma$ then $Z \sim \text{Normal}(0, 1)$. Show that $\chi_Z(t) = \exp(-\frac{1}{2}t^2)$ and use the rules from Exercise 4.]

(9) Show that if $X$ and $Y$ are independent random variables, then so are $f(X)$ and $g(Y)$ for any functions $f$ and $g\colon \mathbb{R} \to \mathbb{R}$. [**Hint:** To show $\Pr[f(X) \in E \,\&\, g(Y) \in F] = \Pr[f(X) \in E]\Pr[g(Y) \in F]$ we should use $f(X) \in E$ if and only if $X \in f^{-1}(E) := \{z \mid f(z) \in E\}$.]

(10) Show that if $X$ is a random variable that is never negative, then $\Pr[X \geq a] \leq \mathbb{E}[X]/a$.

## 7.2 Pseudo-Random Number Generators

*Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin.*

John von Neumann

"Think of a random number between one and a hundred."

**Algorithm 64** Example of pseudo-code for a generator

```
1   generator mylist(n)
2     while true
3       yield n
4       n ← n + 1
5     end
6   end generator
7
8   function print_natural_numbers()
9     for x in mylist(1)
10      print x
11    end for
12  end function
13  /* alternative version */
14  function print_natural_numbers_alt()
15    g ← mylist(1)
16    while true
17      print(next(g))
18    end while
19  end function
```

Suppose you thought of 37. Is it random? It might appear so, but if we understand "random" to include the fact that any two numbers are equally likely, then so are 1, 10, and 99. The point is that what a number is, does not make it random. A number is random if it is generated randomly.

Lotteries often use a collection of light balls that are rotated in a drum, and at certain times a ball is drawn through a tube into a tray where they can be identified more easily. Is this random? Is tossing a coin random? These can be difficult questions to answer. There are mechanical coin tosses that are guaranteed to give either heads or tails as desired. The motion of the balls is governed by macroscopic physical processes, and so should be deterministic. If we knew the initial conditions and the motion of the drum precisely, then it should be in principle possible to determine the entire state of motion of the balls. However, very small perturbations of these initial values can give very different results. When we toss a coin, then the variation and imprecision of how we hold and toss a coin gives enough variation in the outcomes to make the result unpredictable—essentially a random outcome.

Some physical systems such as those governed by quantum mechanics are *in principle* random. However, obtaining random numbers from these systems requires very sensitive detectors and delicate systems. Some physical systems involve thermal noise. Heat is, after all, energy in the form of kinetic energy of atoms, molecules, and their parts. Electronic systems are often subject to thermal noise that gives a background hiss as occurs on AM radio, for example.

Computers are precise systems, so give precise deterministic outcomes. However, we often want to do computations with random numbers. What we create are not truly random numbers, but rather *pseudo-random numbers*. Pseudo-random numbers are generated deterministically and yet appear to have many properties of truly random numbers.

Rather than describe ways of creating sequences of pseudo-random numbers by means of functions, we use the concept of *generators*. A generator is like a function except that it returns a potentially infinite sequence of values. For the pseudo-code in this book, we use the keyword `yield` to return the new value in the sequence. However, the generator can continue doing computations after the `yield`, until the next `yield` statement when it returns the following value in the sequence. A generator will continue performing operations as needed until it terminates. A generator will terminate when it meets a `return` statement. A generator is an example of a *co-routine* or *task* [247]. Generators are supported explicitly in a number of modern programming languages, such as Python, and can be implemented conveniently in other languages, such as Julia.

To illustrate how generators work, Algorithm 64 shows an example of a generator and how to use it. This example would print 1, 2, 3, … in order and without ending. In the pseudo-code here, we use either "`for x in g`…" to enumerate the values generated by a generator $g$, or "$next(g)$" to access the next value yielded by the generator.

Pseudo-random number sequences are most naturally implemented as generators as described above. Algorithms that use pseudo-random sequences are also often best implemented in terms of generators.

### 7.2.1 The Arithmetical Generation of Random Digits

The generation of pseudo-random numbers often involves number theory, which is used to analyze their behavior. John von Neumann's favored method was the so-called "*middle-square*" method [255]: take an $n$-digit integer $x$ with $n$ even. We pad $x$ on the left with zeros if necessary. Compute $y = x^2$ and then take the middle $n$ digits as the next item in the pseudo-random sequence. Unfortunately, this method often ends up in short cycles or becomes constant.

An approach that creates better behaved pseudo-random generators is to use linear congruential generators:

$$x_{k+1} \leftarrow m\, x_k + b \pmod{n}. \tag{7.2.1}$$

Note that $a \pmod{n}$ is taken to mean the reminder when $a$ is divided by $n$: $a \pmod{n} = r$ means that $a = q\, n + r$ with $q$ an integer and $0 \leq r < n$. If $a = q\, b$ for integers $a$, $b$ and $q$, we say that $b$ divides $a$, denoted $b \mid a$.

Analyzing these pseudo-random generators involves basic number theory [216]. Since there are only finitely many possible values of $x_k$ ($0 \leq x_k < n$ for all $k$), there must eventually be a cycle of values $x_k = x_{k+t}$. The cycle length $t > 0$ should be much larger than the number of samples taken. Modern pseudo-random number generators should have extremely long cycle lengths.

To carry out the analysis, we need to introduce some concepts from number theory: the *greatest common divisor* (gcd) of two integers $x$ and $y$, denoted $d = \gcd(x, y)$,

**Algorithm 65** Extended Euclidean algorithm (recursive version)

```
1   function exteuclid(x, y)
2     if y = 0: return (|x|, sign x, 0); end if
3     q ← ⌊x/y⌋; r ← x − q y
4     (d, u, v) ← exteuclid(y, r)
5     return (d, v, u − q v)
6   end function
```

which is the largest positive integer $d$ that divides both $x$ and $y$, or zero if $x = y = 0$. This can be computed by the *Euclidean algorithm* or the *extended Euclidean algorithm*. The extended Euclidean algorithm is shown as Algorithm 65, which not only computes $d = \gcd(x, y)$ but also integers $r$ and $s$ where $d = r\,x + s\,y$. Two non-zero integers $x$ and $y$ are *relatively prime* if $\gcd(x, y) = 1$.

*Lagrange's theorem* in group theory implies that if $a$ and $n$ are relatively prime, then $a^{\phi(n)} = 1 \pmod{n}$ where $\phi(n)$ is the number of elements of the set

$$\{\, x \in \mathbb{Z} \mid \gcd(x, n) = 1 \;\&\; 0 < x < n \,\}.$$

The function $\phi(n)$ here is called the *Euler totient function*. If $n = p_1^{r_1} p_2^{r_2} \cdots p_\ell^{r_\ell}$ where each $p_j$ is a prime number and $r_j \geq 1$, then

$$\phi(n) = \prod_{j=1}^{\ell} p_j^{r_j - 1}(p_j - 1). \tag{7.2.2}$$

We can start our analysis of (7.2.1) by noting that if $\gcd(m, n) = 1$ and $b = 0$ then $x_k = m^k\, x_0 \pmod{n}$. If $\gcd(x_0, n) = 1$ and $x_{k+t} = x_k$ then $m^t = 1 \pmod{n}$ which is true if $\phi(n)$ divides $t$.

For most modern computers, it is convenient for $n$ to be a power of two: $n = 2^r$. Then $\phi(n) = 2^{r-1}(2 - 1) = 2^{r-1}$. The maximum period of the iteration $x_{k+1} = m\, x_k \pmod{n}$ is then $\phi(n) = \phi(2^r) = 2^{r-1} = n/2$, which can only be achieved if $m$ and $x_0$ are odd. To get a full period of length $n$ we need $b \neq 0 \pmod{n}$. In fact, the period can be $n$, which is the maximum possible, of course. Indeed, this is desirable as then any $x_0 \in \{0, 1, 2, \ldots, n - 1\}$ is in the cycle of length $n$.

The conditions for having cycle length $n$ are given by Hull and Dobell [129]:

**Theorem 7.5** *The iteration (7.2.1) has period n provided*

- $gcd(b, n) = 1$*;*
- $m = 1 \pmod{p}$ *if p is a prime divisor of n; and*
- $m = 1 \pmod{4}$ *if 4 is a factor of n.*

The proof is an application of basic number theory. If $n$ is a power of 2 then this reduces to $m$ and $b$ both being odd.

Having period $n$ for (7.2.1) has one very practical consequence: no matter what $x_0$ (the seed) is chosen to be, the average number of times that each possible value in $\{0, 1, 2, \ldots, n - 1\}$ occurs is $1/n$.

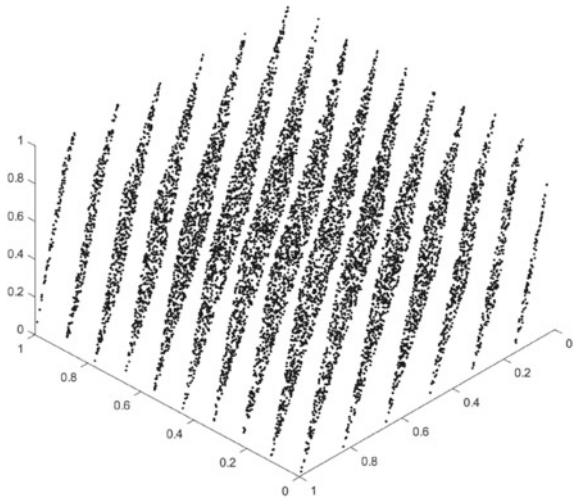



**Fig. 7.2.1** Planes evident for a subsequence of a linear congruential generator

This is not sufficient for the development of practical pseudo-random number generators. For one thing, because $x_{k+1}$ is a function of $x_k$, these are not independent. We would like them, however, to *appear* to be nearly independent. Quantities such as correlations should be close to zero. If $r_k = x_k/n$ and $n$ is large, then $r_k$ will appear to be approximately uniformly distributed (in the long run) in the range $[0, 1)$. If $r_{k+1} = m\, r_k + b' \pmod 1$ with $b' = b/n$, then $\mathbb{E}\left[r_{k+1}\, r_k\right] = \mathbb{E}\left[r_{k+1}\right]\, \mathbb{E}\left[r_k\right] + (1/m)[1 - 6\, b'(1 - b')]$. This differs from what is expected for *independent* $r_{k+1}$ and $r_k$ by $\mathcal{O}(1/m)$. So large $m$ gives a correlation that is close to zero.

Linear congruential pseudo-random number generators are, however, not much used now for pseudo-random number generation as they have other weaknesses. Knuth [145] analyzed linear congruential generators from the point of view of cryptography. He was able to show that it was possible if $n$ is a power of two, to determine $m$ and $b$ from just the leading digits $\lfloor x_k/2^\ell \rfloor$ with relatively small number of elements of the sequence $\lfloor x_k/2^\ell \rfloor$, $k = 0, 1, 2, \ldots$. Of course, the issues involving cryptographic use of pseudo-random generators are different from the numerical issues, but some of the same questions arise.

One of the weaknesses that linear congruential generators have is exposed by looking to higher dimensions. Entacher [85] gives examples of high period linear congruential generators that have serious weaknesses. One example is $n = 2^{32}$, $m = 2\,396\,548\,189$, and $b = 0$ with $x_0 = 1$. This method appears to be a serious pseudo-random number generator. But iterates of a subsequence $(x_{jr},\ x_{(j+1)r},\ x_{(j+2)r})/m$ with $r = 105$ are plotted in Figure 7.2.1 as points in $\mathbb{R}^3$, clearly showing that this subsequence clearly falls into planes.

This illustrates a related result of Marsaglia [171]:

**Theorem 7.6** *If $b = 0$ then the points $(x_k, x_{k+1}, \ldots, x_{k+d-1}) \in \{0, 1, 2, \ldots, n-1\}^d$ for $k = 0, 1, 2, \ldots$ generated by (7.2.1) lie within no more than $(d!\, n)^{1/d}$ hyperplanes of the form $c_0 x_k + c_1 x_{k+1} + \cdots + c_{d-1} x_{k+d-1} \in \mathbb{Z}$.*

Some linear congruential generators can result in the points $(x_k, x_{k+1}, \ldots, x_{k+d-1})$ lying in far fewer hyperplanes. These kinds of defects can be detected through spectral tests: Let $\boldsymbol{j} = (j_1, j_2, \ldots, j_d) \in \mathbb{Z}^d$ with $0 \leq j_\ell < n/r$ for $\ell = 1, 2, \ldots, d$. Create a histogram over the set $\{0, 1, 2, \ldots, n-1\}^d$ for a partition into hypercubes

$$H_{\boldsymbol{j}} = \{i_1 \mid j_1(n/r) \leq i_1 < (j_1+1)(n/r)\} \times \cdots \times \{i_d \mid j_d(n/r) \leq i_d < (j_d+1)(n/r)\}$$

for some divisor $r$ of $n$. We let $h_{\boldsymbol{j}}$ be the number of points $(x_k, x_{k+1}, \ldots, x_{k+d-1})$. Taking the discrete Fourier transform of the $d$-dimensional array $h_{\boldsymbol{j}}$ gives information about whether the points are restricted to a set of parallel planes.

Another property of pseudo-random number generators that is greatly desired is the lack of long-range correlation. That is, we want $x_k$ and $x_{k+p}$ to appear to be independent for $k = 0, 1, 2, \ldots$ for fairly large $p$. Part of this is wanting to have long-period generators ($x_k \neq x_{k+p}$) but might include other conditions such as $x_k \neq n - x_{k+p}$ for modest $p$. Because of these conditions, designing a good pseudo-random number generator can be a difficult task. Most modern pseudo-random number generators are not linear congruential generators. We will look at some.

### *7.2.2 Modern Pseudo-Random Number Generators*

In this section, we will look at a few modern pseudo-random number generators. All of them are essentially linear in that they can be represented fairly compactly as having a state vector update of the form $\boldsymbol{s}_{k+1} \leftarrow A\,\boldsymbol{s}_k \pmod n$. The output of these methods is often tempered in the sense that the actual output is not $\boldsymbol{s}_k$ but rather $\text{trunc}(T\,\boldsymbol{s}_k \pmod n)$ where $T$ is an invertible matrix modulo $n$, and $\text{trunc}(z)$ extracts leading bits from $z$.

#### 7.2.2.1 Mersenne Twister

Mersenne Twister [174] is a generator developed to overcome the limitations of linear congruential generators with a period that is a Mersenne prime: $2^p - 1$ where $p$ is itself prime. The specific method described uses $p = 19\,937$ giving an extremely long period. The basic idea can be described in terms of polynomials over $\mathbb{Z}_2 = \{0, 1\}$. Arithmetic in $\mathbb{Z}_2$ can be implemented very easily in modern computer hardware: addition in $\mathbb{Z}_2$ is implemented as exclusive or, while multiplication is simply "and"-ing the two values.

To describe this method, let $\mathbb{Z}_2[t]$ be the set of polynomials in $t$ with coefficients in $\mathbb{Z}_2$. We can do computations in $\mathbb{Z}_2[t]$ modulo $f(t)$ for a polynomial $f(t)$ using synthetic division. If $f(t)$ is irreducible (that is, cannot be written as $f(t) = g(t)\,h(t)$ with non-constant polynomials $g(t)$ and $h(t)$), then $\mathbb{Z}_2[t]/f(t)$ (the polynomials in $\mathbb{Z}_2[t]$ reduced modulo $f(t)$) forms a field – that is, every polynomial in $\mathbb{Z}_2[t]$ has an inverse modulo $f(t)$. The number of elements of this field is $2^{\deg f}$. So if $f(t)$ has degree $p$, then the number of non-zero elements of $\mathbb{Z}_2[t]/f(t)$ is $2^{\deg f} - 1 = 2^p - 1$. The non-zero elements of a field form a group under multiplication, and so the order

of any non-zero element of $\mathbb{Z}_2[t]/f(t)$ under multiplication must divide $2^p - 1$. As $p$ was chosen to make $2^p - 1$ a prime, this means that there are only two possible orders: one and $2^p - 1$. The only element of $\mathbb{Z}_2[t]/f(t)$ with order one under multiplication is the constant polynomial 1. All other non-zero elements of $\mathbb{Z}_2[t]/f(t)$ have order $2^p - 1$.

The hardest problem in creating a method of this type is to find a suitable irreducible polynomial $f(t) \in \mathbb{Z}_2[t]$ of the desired degree. The authors of [174] found an efficient way to test for irreducibility of polynomials in $\mathbb{Z}_2[t]$. Part of the trick used is noting that for any polynomial $g(t)$ in $\mathbb{Z}_2[t]$ we have $g(t^2) = g(t)^2$. The irreducible polynomial found is a sparse polynomial in the sense that most coefficients are zero, which is important for efficiency of the generator. Finding an irreducible polynomial in $\mathbb{Z}_2[t]$ of degree $p$ consists of generating polynomials of this degree and then testing to see if the generated polynomial is irreducible. Fortunately, irreducible polynomials of degree $p$ in $\mathbb{Z}_2[t]$ are fairly common. The number of irreducible polynomials of degree $n$ in $\mathbb{Z}_2[t]$ is given by the formula (see [217, Chap. 2]):

$$\frac{1}{n}\sum_{d|n}\mu(n/d)\,2^d,$$

where $\mu$ is the Möbius function: $\mu(k)$ is zero if $k$ has a non-trivial square factor, is $+1$ if $k$ is the product of an even number of distinct primes, and is $-1$ if $k$ is the product of an odd number of distinct primes. In particular, if $n = p$ is prime, then the number of irreducible polynomials of degree $p$ in $\mathbb{Z}_2[t]$ is $(2^p - 1)/p$ out of $2^p$ possible polynomials of degree $p$ in $\mathbb{Z}_2[t]$. Thus "randomly" selecting polynomials of degree $p$ has a probability of $\approx 1/p$ of selecting an irreducible polynomial. For $p = 19\,937$, finding an irreducible polynomial would still require an automated search, but it is still quite feasible.

The actual algorithm implements an iteration on vectors in $\mathbb{R}^{n\,w-r}$ ($nw - r = 2^p - 1$). We split the vectors into parts $\boldsymbol{x}_{\ell+k} \in \mathbb{Z}_2^w$ and $\boldsymbol{x}_\ell^u \in \mathbb{Z}_2^{w-r}$:

(7.2.3)

$$\begin{bmatrix} \boldsymbol{x}_{\ell+n} \\ \boldsymbol{x}_{\ell+n-1} \\ \boldsymbol{x}_{\ell+n-2} \\ \vdots \\ \boldsymbol{x}_{\ell+m+1} \\ \vdots \\ \boldsymbol{x}_{\ell} \\ \boldsymbol{x}^u_{\ell+1} \end{bmatrix} = \begin{bmatrix} 0 & & & & I_w & S \\ I_w & 0 & & & & \\ & I_w & \ddots & & & \\ & & \ddots & \ddots & & \\ & & & \ddots & \ddots & 0 \\ & & & & I_w & I_{w-r}\ \ 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{\ell+n-1} \\ \boldsymbol{x}_{\ell+n-2} \\ \boldsymbol{x}_{\ell+n-3} \\ \vdots \\ \boldsymbol{x}_{\ell+m+m} \\ \vdots \\ \boldsymbol{x}_{\ell-1} \\ \boldsymbol{x}^u_{\ell} \end{bmatrix} = B \begin{bmatrix} \boldsymbol{x}_{\ell+n-1} \\ \boldsymbol{x}_{\ell+n-2} \\ \boldsymbol{x}_{\ell+n-3} \\ \vdots \\ \boldsymbol{x}_{\ell+m+m} \\ \vdots \\ \boldsymbol{x}_{\ell-1} \\ \boldsymbol{x}^u_{\ell} \end{bmatrix}$$

where $S = A \begin{bmatrix} & I_{w-r} \\ I_w & \end{bmatrix}$ and

$$A = \begin{bmatrix} 0 & & & a_{w-1} \\ 1 & 0 & & a_{w-2} \\ & 1 & \ddots & \vdots \\ & & \ddots \; \ddots & a_1 \\ & & 1 & a_0 \end{bmatrix}. \tag{7.2.4}$$

Note that $\boldsymbol{x}_k^u$ denotes the upper $w - r$ components of $\boldsymbol{x}_k$.

The matrix in (7.2.3) is sparse for efficiency. Its characteristic polynomial is easily computed explicitly:

$$\chi_B(t) = (t^n + t^m)^{w-r}(t^{n-1} + t^{m-1})^r + \sum_{j=0}^{r-1} a_j (t^n + t^m)^{w-r}(t^{n-1} + t^{m-1})^{r-j-1} + \sum_{j=r}^{w-1} a_j (t^n + t^m)^{w-j-1}.$$

Note that the specific version of Mersenne Twister can be represented by the vector

$$\boldsymbol{a} = [a_0, a_1, \ldots, a_{w-1}]^T \in \mathbb{Z}_2^w$$

along with $w$, $m$, $n$, and $r$. In implementations, $\boldsymbol{a}$ can be represented by a single bit string, or integer. For example, the method MT19937 has $w = 32$ (exploiting 32 bit architectures), $n = 624$ (the number of 32 bit "words" used), $m = 397$, $r = 31$ (one bit in $\boldsymbol{x}_k^u$ for efficiency), and $\boldsymbol{a}$ represented in hexadecimal by `0x9908B0DF`.

Rather than use the raw bits produced by the algorithm, the output is *tempered*. Instead of returning the raw bits produced, a tempering function is applied to the raw output of the generator. The paper [174] recommends giving the output $\boldsymbol{y}_k = T\,\boldsymbol{x}_k$ with $T$ an integer matrix given in terms of bit shifts and additions modulo 2.

The Mersenne Twister generator has been criticized for various defects. For example, if the initial state has many zeros, it can take many steps before the output starts appearing to be random. This problem can be offset by an improved initialization scheme [173]. A deeper criticism of the Mersenne Twister generator can be found in Vigna [254]. Improvements on the original Mersenne Twister which scramble the bits more effectively can be found in [197]; Vigna also proposed other bit-scrambling methods in [253] which are not subject to these criticisms.

#### 7.2.2.2 Permuted Congruential Generators

*Permuted congruential generators* (PCG's) [193] are a way of adapting linear congruential generators to give better output. Essentially PCG's focus on tempering linear congruential generators by means of nonlinear functions (with respect to $\mathbb{Z}_2$) of tuples of consecutive outputs $(\boldsymbol{x}_{k-r}, \boldsymbol{x}_{k-r+1}, \ldots, \boldsymbol{x}_{k-1}, \boldsymbol{x}_k)$ of the linear congru-

**Algorithm 66** Combining generators

```
1   generator combine_xor(g1, g2)
2     while true
3       yield next(g1) + next(g2) (mod 2)
4     end
5   end
```

ential generator based on bit shifts and rotations. These clearly cannot change the period of the underlying generator but can avoid the problem of consecutive outputs lying on planes as noticed by Marsaglia. The tempering functions used in [193] are guaranteed to be bijections, so the period of the output cannot be any less than the period of the underlying generator.

#### 7.2.2.3 New Generators From Old

Tempering can be used to improve the quality of output from underlying generators. The tempering function should be a bijection so that the period of the tempered output is the same as the period of the underlying generator.

There are also methods of combining generators to increase the period of the combined generator. For example, Algorithm 66 shows one way to combine vectors of bits generated by two generators $g_1$ and $g_2$.

If $t_1$ and $t_2$ are the periods of generators $g_1$ and $g_2$, respectively, then the period of the combined generator is the least common multiple of $t_1$ and $t_2$: $\text{lcm}(t_1, t_2) = (t_1 t_2)/\text{gcd}(t_1, t_2)$. This would be most effective when applied to, for example, Mersenne Twister type methods of different periods but not to linear congruential generators with moduli that are powers of two. If $g_1$ and $g_2$ uniform generators of real numbers in the interval $[0, 1]$, then returning the fractional part of $next(g_1) + next(g_2)$ would also be an effective way of combining two generators of this type.

### *7.2.3 Generating Samples from Other Distributions*

Often it is important to generate samples that have probability distributions other than a uniform distribution. However, we can use generators that produce outputs uniformly over the interval $[0, 1]$ to produce real outputs that have a specified distribution. One of the most straightforward methods to do this by using the inverse function to the cumulative distribution function. To sample from a probability distribution with density function $p(x)$, we can use the cumulative distribution function $F(x) = \int_{-\infty}^{x} p(s)\, ds$: Suppose $U$ is uniformly distributed on $[0, 1]$, and $X = F^{-1}(U)$. Assuming that $F$ is strictly increasing,

**Algorithm 67** Box–Muller method for generating normally distributed samples

```
1   generator boxmuller(U)
2     while true
3       u1 ← next(U); u2 ← next(U)
4       r ← √(−2 ln(u1))
5       yield r cos(2π u2)
6     end while
7   end generator
```

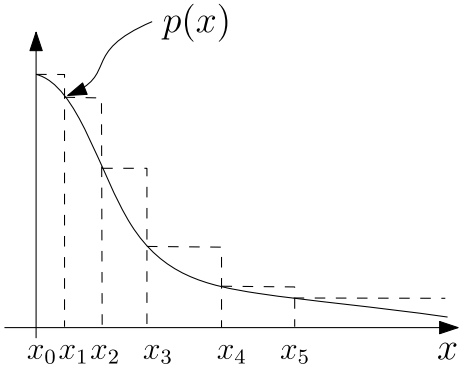



**Fig. 7.2.2** Ziggurat algorithm

$$\Pr[X \le x] = \Pr[F(X) \le F(x)] = \Pr[U \le F(x)] = F(x).$$

Thus $X$ has the cumulative distribution function $F$, as we wanted (Figure 7.2.2).

This can be applied to computing normally distributed pseudo-random variable: simply set $F(x) = \frac{1}{2}(1 + \operatorname{erf}(x))$ where $\operatorname{erf}(x) = (2/\sqrt{\pi}) \int_0^x \exp(-s^2/2)\, ds$ is the error function. Given $U$ sampled from a uniform distribution we solve $F(X) = U$ for $X$. Since pseudo-random uniformly distributed values can be zero or one, which would result in infinite results, we need to treat these extreme values carefully. If we use pseudo-random generators with values $k/n$, $k = 0, 1, 2, \ldots, n-1$, then the value $0/n$ should be replaced by either $1/n$ or $\frac{1}{2}/n$. The value $n/n = 1$ might also need to be replaced by $(n-1)/n = 1 - 1/n$ or $(n - \frac{1}{2})/n$.

An alternative method for creating normally distributed pseudo-random samples from uniformly distributed samples is the *Box–Muller method* [27], as shown in Algorithm 67. Note that $U$ is a uniform generator providing samples uniformly distributed over $[0, 1]$. Care should be taken to ensure that $U$ does not generate exactly zero. This can sometimes be achieved by replacing $U$ by $1 - U$ if $U$ itself can exactly generate zero.

An alternative approach is the *ziggurat algorithm* [172].

The ziggurat algorithm assumes the probability density function $p(x)$ is a monotone decreasing function. Assuming $p$ is zero outside $[x_0, \infty)$, we subdivide the interval into pieces $[x_k, x_{k+1})$, $k = 0, 1, 2, \ldots, n-2$, where $\int_{x_k}^{x_{k+1}} p(x)\, dx = 1/n$.

**Algorithm 68** Ziggurat algorithm

```
1   generator ziggurat(U, p, x, fallback)
2     while true
3       u ← n next(U); k ← ⌊u⌋
4       if u ≥ n − 1
5         yield fallback(u/n)
6       else
7         x ← x_k + (x_{k+1} − x_k)(u − k)
8         y ← p(x_k) next(U)
9         if y ≤ p(x): yield x; end if
10      end if
11    end while
12  end generator
```

There is one infinite interval $[x_{n-1}, \infty)$ where $\int_{x_{n-1}}^{\infty} p(x)\,dx = 1/n$. These $x_k$'s are pre-computed and passed to the ziggurat algorithm.

The ziggurat algorithm is shown as Algorithm 68. In the algorithm $U$ generates samples from the uniform distribution Uniform(0, 1). The basic idea is to first select one of the intervals $[x_k, x_{k+1})$, which are equally probable. Once this is done, we create a point $(x, y)$ uniformly distributed over the rectangle $[x_k, x_{k+1}) \times [0, p(x_k)]$. If $(x, y)$ is under the graph of $p(x)$ then the point $(x, y)$ is accepted and $x$ returned. Otherwise, the sample is rejected. Rejection is relatively rare if the intervals $[x_k, x_{k+1})$ are small as the rejection probability is the area of the rectangle $[x_k, x_{k+1}) \times [0, p(x_k)]$ that is *not* under the curve $y = p(x)$ divided by the area of $[x_k, x_{k+1}) \times [0, p(x_k)]$. This means that with narrow rectangles on average little more than two samples of a uniform distribution are needed to generate a new sample of the given distribution. There is still the necessity of handling the unbounded interval $[x_{n-1}, \infty)$ using an alternative (or fallback) function, when this is needed. In these relatively infrequent cases, the inverse of the cumulative distribution function can be used.

### *7.2.4 Parallel Generators*

In many applications, such as for Monte Carlo methods, it is desirable to have multiple processors running pseudo-random number generators. These generators should be statistically independent, or at least appear to be independent. In a parallel computing environment, the same generator is typically used by each processor. Since pseudo-random generators are deterministic processes, we need to initialize each copy of the generator differently so as to at least avoid overlap in the generated sequences.

There is a way of doing this efficiently for large period linear generators. For example, the Mersenne Twister generator (see Section 7.2.2.1) has period $2^p - 1$ where this is a Mersenne prime, with the value $p = 19\,937$. We do not expect even very long running Monte Carlo methods to use more than, say, $2^{100} \approx 1.27 \times 10^{30}$ samples. (*Avagadro's number* $\approx 6.0 \times 10^{23}$, is the number of hydrogen atoms in

**Algorithm 69** Matrix powers via repeated squaring

```
1   function mpower(A, s)
2     if s = 1: return A; end if
3     if s even
4       return mpower(A^2, s/2)
5     else
6       return A mpower(A^2, (s − 1)/2)
7     end if
8   end function
```

one gram of hydrogen. This is a rough upper bound to the number of objects that any current or future computer memory will be able to hold.) So if we ensure that the generators are initialized to have this spacing (or more) of the generated sequences, then there is unlikely to be any overlap between the sequences. It would also be wise to avoid spacing that is a divisor of the period of the generator.

Of course, with spacing $s$ we could, in principle, run the generator with the standard initialization $k$ $s$ times to prepare the generator for processor $k$. If $s$ is as large as $2^{100}$, this would be unacceptably long. However, for linear generators, this can be done efficiently by repeated squaring. Consider first, linear congruential generators (7.2.1)

$$x_{k+1} \leftarrow m\, x_k + b \pmod{n}.$$

This can be represented as a linear update of the state:

$$\begin{bmatrix} x_{k+1} \\ 1 \end{bmatrix} = \begin{bmatrix} m & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_k \\ 1 \end{bmatrix} \pmod{n}.$$

We can then compute

$$\begin{bmatrix} x_{k+s} \\ 1 \end{bmatrix} = \begin{bmatrix} m & b \\ 0 & 1 \end{bmatrix}^s \begin{bmatrix} x_k \\ 1 \end{bmatrix} \pmod{n}.$$

We can use repeated squaring to compute $A^s$ for a matrix $A$ using $\mathcal{O}(\log s)$ matrix multiplications, as shown in Algorithm 69.

This same idea can be applied to the Mersenne Twister generator, although the matrix is a $p \times p$ binary matrix, and for $p = 19\,937$ each matrix multiplication can be expensive. An alternative is to combine several Mersenne Twister generators with Mersenne prime periods $2^{p_1} - 1$, $2^{p_2} - 1$, ..., $2^{p_r} - 1$ and with smaller distinct values for $p_1$, $p_2$, ..., $p_r$, combining the outputs using Algorithm 66 for example. The period of the combined generator would be $\prod_{j=1}^{r}(2^{p_j} - 1) \approx 2^q$ where $q = \sum_{j=1}^{r} p_j$. Initializing a generator to start at $x_s$ would then require multiplying $r$ matrices of sizes $p_j \times p_j$ $(j = 1, 2, \ldots, r)$ $\mathcal{O}(\log s)$ times for a cost of $\mathcal{O}((\log s) \sum_{j=1}^{r} p_j^3)$ time and $\mathcal{O}((\log s) \sum_{j=1}^{r} p_j^2)$ memory.

**Exercises.**

(1) Consider a linear congruential pseudo-random number generator: $x_{k+1}=mx_k+b \pmod{n}$. Let $d = \gcd(m, n)$. Show that $x_{k+1} = b \pmod{d}$ for all $k$.
(2) Show that if $\gcd(b, n) = 1$ and $\gcd(m - 1, n) = 1$, then the generator $x_{k+1} = mx_k + b \pmod{n}$ has the same period as $y_{k+1} = my_k \pmod{n}$.
(3) Most pseudo-random number generators do not reveal their entire state. For example, a linear congruential pseudo-random number generator working modulo $2^{64}$ will usually not return $x_k$, but only the most significant 32 bits of $x_k$. Explain why returning the least significant 32 bits of $x_k$ is not helpful.
(4) Consider the vector linear congruential pseudo-random number generator: $\boldsymbol{x}_{k+1} = M\,\boldsymbol{x}_k + \boldsymbol{b} \pmod{n}$. If $n$ is prime (so that the integers modulo $n$ form a field) and $M - I$ is invertible modulo $n$, show that $\boldsymbol{y}_k = \boldsymbol{x}_k + (M - I)^{-1}\boldsymbol{b} \pmod{n}$ satisfies $\boldsymbol{y}_{k+1} = M\,\boldsymbol{y}_k \pmod{n}$. If $\boldsymbol{z}_k = T\,\boldsymbol{y}_k$ with $T$ invertible modulo $n$, show that $\boldsymbol{z}_{k+1} = TMT^{-1}\boldsymbol{z}_k \pmod{n}$.
(5) Show that if $f(x)$ is a polynomial in $x$ with integer coefficients, then $f(x)^2 = f(x^2) \pmod{2}$.
(6) Most $n \times n$ integer matrices are invertible modulo 2. We can more precisely estimate the probability of generating an invertible $n \times n$ integer matrix $A$ modulo 2, with each entry being chosen as either zero or one with probability $\frac{1}{2}$ independently. Show that the probability that $A$ is invertible modulo 2 is $p_n = \prod_{j=1}^{n-1}(1 - 2^{-j})$. [**Hint:** Let $A = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n]$ be the matrix generated as indicated above. The column $\boldsymbol{a}_1$ must be a non-zero vector: there are $2^n - 1$ of these. For $A$ to be invertible, $\boldsymbol{a}_2$ must not be a multiple of $\boldsymbol{a}_1$; once $\boldsymbol{a}_1$ is determined, there are $2^n - 2^1$ such vectors. Then $\boldsymbol{a}_3$ must not be a linear combination of $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$; once $\boldsymbol{a}_1$ and $\boldsymbol{a}_2$ are determined, there are $2^n - 2^2$ such vectors. Repeating this argument, there are $\prod_{j=1}^{n-1}(2^n - 2^j)$ such choices of $A$, out of $2^{n^2}$ possible binary $n \times n$ matrices.] Generate 1000 binary $5 \times 5$, $10 \times 10$, and $20 \times 20$ matrices pseudo-randomly; record the number of these generated matrices that are invertible modulo 2. How do the empirical probabilities of generating invertible binary matrices modulo 2 relate to the predicted value $p_n$? What is $\lim_{n\to\infty} p_n$ to six digits?

## 7.3 Statistics

Statistics are ways of studying probability distributions through sets of samples (usually independent) taken from that distribution. Statistics include averages of different kinds, measures of variation about an average, and other measures of properties of that distribution.

Given a sample $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$, we have an empirical probability distribution given by

$$\Pr\left[\boldsymbol{X}_{sample} \in E\right] = N^{-1}\,|\{i \mid \boldsymbol{x}_i \in E\}|\,;$$

that is, each $\boldsymbol{x}$ has probability equal to the number of times $\boldsymbol{x}$ appears in the sample (repetitions allowed) divided by the number of elements in the sample. Empirical distributions are naturally discrete, but can be used to represent or approximate continuous distributions in the sense that for bounded continuous functions $f$, $N^{-1}\sum_{i=1}^{N} f(\boldsymbol{x}_i) \to \mathbb{E}[f(\boldsymbol{X})]$ as $N \to \infty$ where $\boldsymbol{X}$ is distributed according to the underlying probability distribution.

Often probability density functions are parameterized $p(\boldsymbol{x};\boldsymbol{\theta})$ with parameter vector $\boldsymbol{\theta}$. Any method for estimating $\boldsymbol{\theta}$ from a sample is called a *statistic*. Since samples are random variables, an estimate $\widehat{\boldsymbol{\theta}}$ computed from the sample must itself be a random variable. The estimate is *unbiased* if $\mathbb{E}\left[\widehat{\boldsymbol{\theta}}\right] = \boldsymbol{\theta}$. The estimate is *asymptotically unbiased* if the estimate $\widehat{\boldsymbol{\theta}}_N$ for sample size $N$ satisfies $\mathbb{E}\left[\widehat{\boldsymbol{\theta}}_N\right] \to \boldsymbol{\theta}$ as $N \to \infty$. One class of estimators are *maximum likelihood estimators* (MLE's). The MLE estimator $\widehat{\boldsymbol{\theta}}$ maximizes the likelihood $\prod_{i=1}^{N} p(\boldsymbol{x}_i, \boldsymbol{\theta})$ over $\boldsymbol{\theta}$ for the sample $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$. Often it is easier to analyze the logarithm of the likelihood: $\sum_{i=1}^{N} \ln p(\boldsymbol{x}_i, \boldsymbol{\theta})$.

### 7.3.1 Averages and Variances

The word "average" is often used as a synonym for "mean" in the sense of (7.3.1). But "average" can also refer to a median or a mode, also described here.

The *mean* of a real- or vector-valued random variable $\boldsymbol{X}$ is the expectation $\mathbb{E}[\boldsymbol{X}]$. The mean of a sample $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ is

$$\overline{\boldsymbol{x}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i. \tag{7.3.1}$$

The sample mean is the expectation of a random variable $\boldsymbol{X}_{sample}$.

The *median* of a real-valued random variable is the value $m$ where $\Pr[X \le m] = \frac{1}{2}$; if there is no such $m$ because the probability $\Pr[X \le m]$ jumps over $\frac{1}{2}$, then we usually take $m = \frac{1}{2}(m_1 + m_2)$ where $\Pr[X \le m_2] \ge \frac{1}{2}$, $\Pr[X \ge m_1] \ge \frac{1}{2}$ with $m_2 \le m_1$. The median of a random variable $X$ is also the minimizer of $\mathbb{E}[|X - s|]$ over $s$. The median of a set of real values $\{x_1, x_2, \ldots, x_N\}$ is the median of the random variable $X_{sample}$; its value is $m$ where $|\{i \mid x_i < m\}| = |\{i \mid x_i > m\}|$. If the sample is sorted so that $x_1 \le x_2 \le \cdots \le x_n$ then the median is $x_{(N+1)/2}$ for $N$ odd and $\frac{1}{2}(x_{(N/2)} + x_{(N/2)+1})$ for $N$ even.

The *mode* of a discrete-valued random variable is value $x$ that maximizes $\Pr[X = x]$. For a continuous random variable, the mode maximizes the probability density function. The mode of a sample is not so easily defined in general.

Both the mean and the median of a real-valued random variable and for a set of samples can be determined by through optimization problems:

$$\text{mean minimizes } \mathbb{E}\left[(X-m)^2\right] \text{ over } m,$$
$$\text{median minimizes } \mathbb{E}\left[\,|X-m|\,\right] \text{ over } m.$$

The *variance* of a random variable $X$ is $\mathbb{V}\text{ar}\,[X] = \mathbb{E}\left[(X-\mu)^2\right]$ where $\mu = \mathbb{E}\,[X]$ is the mean of $X$. Usually the variance is written as $\sigma^2$ where $\sigma \geq 0$ is the *standard deviation* of $X$. If $X$ has a physical quantity, then $\mu$ and $\sigma$ both have the same physical units as $X$, but the variance has the units of $X^2$.

A commonly used alternative formula for the variance is

$$\mathbb{V}\text{ar}\,[X] = \mathbb{E}\left[X^2\right] - \mathbb{E}\,[X]^2 = \mathbb{E}\left[X^2\right] - \mu^2.$$

It is tempting to estimate $\mathbb{V}\text{ar}\,[X]$ with $\widehat{\sigma^2} = \mathbb{V}\text{ar}\left[X_{sample}\right] = (1/N)\sum_{i=1}^{N}(x_i - \widehat{\mu})^2$ and $\widehat{\mu} = (1/N)\sum_{i=1}^{N} x_i$. But this is actually a biased estimate of the variance. An unbiased estimate is

$$\widehat{\sigma^2}_{\text{Unb}} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \widehat{\mu})^2. \tag{7.3.2}$$

On the other hand, the (biased) maximum likelihood estimator for the variance using the normal probability distribution $p(x;\mu,\sigma) = (2\pi\sigma^2)^{-1/2}\exp(-(x-\mu)^2/(2\sigma^2))$ is (see Exercise 2)

$$\widehat{\sigma^2}_{\text{MLE}} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \widehat{\mu})^2. \tag{7.3.3}$$

The maximum likelihood estimator of the variance for normally distributed samples is still asymptotically unbiased.

### 7.3.2 Regression and Curve Fitting

Regression and curve fitting are also important computational tasks in statistics. A common approach is to use least squares to fit a set of data $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, N$, with a function $\boldsymbol{x} \mapsto \sum_{j=1}^{m} c_j\varphi_j(\boldsymbol{x})$. We assume a statistical model $y_i = \sum_{j=1}^{m} c_j\varphi_j(\boldsymbol{x}_i) + \epsilon_i$ where $\epsilon_i$ follows a Normal$(0,\sigma^2)$ distribution and the $\epsilon_i$'s are mutually independent. The likelihood function is

$$\begin{aligned} L(\boldsymbol{c}) &= \prod_{i=1}^{N} (2\pi)^{-1/2} \exp(-\epsilon_i^2/\sigma^2) \\ &= (2\pi)^{-N/2} \exp(-\boldsymbol{\epsilon}^T \boldsymbol{\epsilon}/\sigma^2), \qquad \text{so} \\ \ln L(\boldsymbol{c}) &= -\frac{N}{2} \ln(2\pi) - \frac{1}{\sigma^2} \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = -\frac{N}{2} \ln(2\pi) - \frac{1}{\sigma^2} (\Phi \boldsymbol{c} - \boldsymbol{y})^T (\Phi \boldsymbol{c} - \boldsymbol{y}) \end{aligned}$$

where $\Phi_{ij} = \varphi_i(\boldsymbol{x}_j)$. So maximizing $L(\boldsymbol{c})$ for $\boldsymbol{c}$ is equivalent to minimizing $(\Phi \boldsymbol{c} - \boldsymbol{y})^T (\Phi \boldsymbol{c} - \boldsymbol{y})$. That is, we are minimizing the sum of the squares of the errors. This can be done using either normal equations or the QR factorization. The optimal $\boldsymbol{c}$ is given by the normal equations $\Phi^T \Phi \widehat{\boldsymbol{c}} = \Phi^T \boldsymbol{y}$. If $\boldsymbol{c}$ is the true set coefficients, $\boldsymbol{y} = \Phi \boldsymbol{c} + \boldsymbol{\epsilon}$ and so $\Phi^T \Phi \widehat{\boldsymbol{c}} = \Phi^T \Phi \boldsymbol{c} + \Phi^T \boldsymbol{\epsilon}$ and therefore $\widehat{\boldsymbol{c}} = \boldsymbol{c} + (\Phi^T \Phi)^{-1} \Phi^T \boldsymbol{\epsilon}$. The error in the computed coefficients $\widehat{\boldsymbol{c}} - \boldsymbol{c} = (\Phi^T \Phi)^{-1} \Phi^T \boldsymbol{\epsilon}$ is distributed according to the Normal$(\boldsymbol{0}, \sigma^2 (\Phi^T \Phi)^{-1})$ distribution. This estimate is unbiased: $\mathbb{E}[\widehat{\boldsymbol{c}}] = \boldsymbol{c}$. We might want to estimate the variance $\sigma^2$ of the $\epsilon_i$'s by using $\widehat{\boldsymbol{\epsilon}} = \boldsymbol{y} - \Phi \widehat{\boldsymbol{c}}$. However, this will lead to a biased estimate of $\sigma^2$:

$$\begin{aligned} \widehat{\boldsymbol{\epsilon}} &= \boldsymbol{y} - \Phi \widehat{\boldsymbol{c}} = \boldsymbol{y} - \Phi (\Phi^T \Phi)^{-1} \Phi^T \boldsymbol{y} \\ &= \left[ I - \Phi (\Phi^T \Phi)^{-1} \Phi^T \right] \boldsymbol{y}. \end{aligned}$$

The matrix $P := I - \Phi (\Phi^T \Phi)^{-1} \Phi^T$ is the orthogonal projection onto the orthogonal complement of range $\Phi$. Since $\boldsymbol{y} = \Phi \boldsymbol{c} + \boldsymbol{\epsilon}$,

$$\widehat{\boldsymbol{\epsilon}} = P \left( \Phi \boldsymbol{c} + \boldsymbol{\epsilon} \right) = P \boldsymbol{\epsilon}$$

so $\widehat{\boldsymbol{\epsilon}}$ is in $(\text{range}\, \Phi)^{\perp}$. We can consider $\widehat{\boldsymbol{\epsilon}}$ to be distributed according to the Normal$(\boldsymbol{0}, \sigma^2 P)$ distribution as $P^T = P = P^2 = P^T P$, understood as the limit of the distribution of Normal$(\boldsymbol{0}, \sigma^2 P + \alpha I)$ as $\alpha \downarrow 0$. Also

$$\mathbb{E}\left[ \widehat{\boldsymbol{\epsilon}}^T \widehat{\boldsymbol{\epsilon}} \right] = \mathbb{E}\left[ \boldsymbol{\epsilon}^T P^T P \boldsymbol{\epsilon} \right] = \mathbb{E}\left[ \boldsymbol{\epsilon}^T P \boldsymbol{\epsilon} \right] = \text{trace}(P)\, \sigma^2. \tag{7.3.4}$$

For $\boldsymbol{y} \in \mathbb{R}^n$, $\boldsymbol{c} \in \mathbb{R}^m$, $P$ is the orthogonal projection onto $(\text{range}\, \Phi)^{\perp}$, which has dimension $n - m$. Since $P$ is symmetric because it is an *orthogonal projection*, it has real eigenvalues that are either zero or one: $n - m$ eigenvalues are one and $m$ eigenvalues are zero. Thus $\text{trace}(P) = n - m$. So $\mathbb{E}\left[ \widehat{\boldsymbol{\epsilon}}^T \widehat{\boldsymbol{\epsilon}} \right] = (n - m)\sigma^2$. An unbiased estimate of $\sigma^2$ is then $\widehat{\sigma}^2 = \mathbb{E}\left[ \widehat{\boldsymbol{\epsilon}}^T \widehat{\boldsymbol{\epsilon}} \right] / (n - m)$, not $\mathbb{E}\left[ \widehat{\boldsymbol{\epsilon}}^T \widehat{\boldsymbol{\epsilon}} \right] / n$.

What if $\epsilon_i \sim \text{Normal}(0,\ \sigma_i^2)$ with different $\sigma_i$'s? Then the likelihood function is

$$L(\boldsymbol{c}) = (2\pi)^{-N/2} \exp(-\boldsymbol{\epsilon}^T D^{-2} \boldsymbol{\epsilon})$$

where $D = \text{diag}(\sigma_1,\ \sigma_2,\ \ldots,\ \sigma_N)$. This leads to a weighted least squares problems.

$$\min_{\boldsymbol{c}} (\Phi \boldsymbol{c} - \boldsymbol{y})^T D^{-2} (\Phi \boldsymbol{c} - \boldsymbol{y}),$$

which can also be solved by normal equations $\Phi^T D^{-2}\Phi \boldsymbol{c} = \Phi^T D^{-2}\boldsymbol{y}$, or the QR factorization of $D^{-1}\Phi$.

There are also nonlinear regression problems, such as mixed exponentials with unknown rates:

$$y_i = a_1 \exp(-b_1 t_i) + a_2 \exp(-b_2 t_i) + \epsilon_i,$$

with $\epsilon_i \sim \text{Normal}(0, \sigma^2)$. If we apply maximum likelihood methods to this, we want to minimize $L(\boldsymbol{a}, \boldsymbol{b}) := \sum_{i=1}^{N} (y_i - (a_1 \exp(-b_1 t_i) + a_2 \exp(-b_2 t_i)))^2$ with respect to $a_1$, $a_2$, $b_1$, $b_2$. Note that the loss function $L(\boldsymbol{a}, \boldsymbol{b})$ is invariant under swapping the parameters for the two terms $(a_1, b_1) \leftrightarrow (a_2, b_2)$. Since we do not expect that the solution will be symmetric ($(a_1, b_1) = (a_2, b_2)$), we will probably have multiple local minimizers. This means that $L(\boldsymbol{a}, \boldsymbol{b})$ would not be convex, but can have multiple local minimizers. Standard optimization algorithms can have difficulties in these situations.

### *7.3.3 Hypothesis Testing*

Data are used to test hypotheses. There are two main ways of doing this. One is to use Bayes' theorem for conditional probability. The other is to use "$p$-values" for checking the plausibility of a basic hypothesis.

#### 7.3.3.1 Bayesian Inference

Given a data set $D$, how likely is a hypothesis $H$? This is the conditional probability $\Pr[H \mid D]$. But this is usually very difficult to compute directly. Instead, it is much easier to compute $\Pr[D \mid H]$ as the hypothesis $H$ is a statement about the nature of the data $D$. From Bayes' theorem,

$$\Pr[H \mid D] = \frac{\Pr[D \& H]}{\Pr[D]} = \frac{\Pr[D \mid H]\Pr[H]}{\sum_{H'} \Pr[D \mid H']\Pr[H']}$$

where $H'$ ranges over all plausible hypotheses. The value $\Pr[H]$ is the probability that hypothesis $H$ is true *before we have any data about it*. This probability $\Pr[H]$ is the *a priori* probability, while $\Pr[H \mid D]$ is the *a posteriori* probability of hypothesis $H$.

Estimating $\Pr[H]$ is often a subjective matter. Consider the question, "What is the probability that the sun will rise every 24 hours?" We might have personally observed these occurring tens of thousands of times and have historical records going back hundreds of thousands of times before that. But what should we assign to this hypothesis *before* we have evidence, or before we know anything about the sun? Without evidence we have very little basis for any computation of $\Pr[H]$. We can make an arbitrary assignment $\Pr[H] = \frac{1}{2}$. The observed data of centuries of observing the sun rise every day would then give $\Pr[H \mid D]$ very close to one. On

the other hand, assuming $\Pr[H] = 10^{-6}$, $\Pr[H \mid D]$ might not be close to one even with centuries of observations.

There is a way of avoiding making assumptions about the *a priori* probability, and that is to look instead at the *odds ratio* $\Pr[H]/\Pr[\text{not } H]$: Note that

$$\frac{\Pr[H \mid D]}{\Pr[\text{not } H \mid D]} = \frac{\Pr[D \mid H]}{\Pr[D \mid \text{not } H]} \frac{\Pr[H]}{\Pr[\text{not } H]}.$$

We are not looking for the absolute value of $\Pr[H \mid D]$ or even $\Pr[H \mid D]/\Pr[\text{not } H \mid D]$. Rather we see how the evidence affects the odds ratio. If the odds ratio

$$\Pr[D \mid H]/\Pr[D \mid \text{not } H] \tag{7.3.5}$$

is large, then we have strong evidence for $H$ over not $H$.

If we have continuous data, using odds ratios can be a much more effective technique: $\Pr[D \mid H]$ is typically zero if the items in the data set $D$ are sampled from a continuous probability distribution. The ratio $\Pr[D \mid H]/\Pr[D \mid \text{not } H]$ would then be "0/0" and undefined. However, if we consider data sets $D'$ generated according to the hypotheses $H$ and not $H$ and $D$ as the given (fixed) data set, we can replace $\Pr[D \mid H]/\Pr[D \mid \text{not } H]$ with

$$\lim_{\epsilon \downarrow 0} \frac{\Pr\left[\|D' - D\| \leq \epsilon \mid H\right]}{\Pr[\|D' - D\| \leq \epsilon \mid \text{not } H]} = \frac{p_{D'|H}(D)}{p_{D'|\text{not } H}(D)} \tag{7.3.6}$$

where $p_{D'|H}$ and $p_{D'|\text{not } H}$ are the probability density functions for the data set $D$ given $H$ and not $H$, respectively. The appropriate odds ratio for continuous data is $p_{D'|H}(D)/p_{D'|\text{not } H}(D)$.

Suppose we take $N$ independent samples $\boldsymbol{x}_i$, $i = 1, 2, \ldots, N$. We wish to determine whether the samples come from either a probability distribution with density function $p_1(\boldsymbol{x})$ (hypothesis $H$) or with density function $p_2(\boldsymbol{x})$ (hypothesis not $H$). Then $p_{D'|H}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \prod_{i=1}^N p_1(\boldsymbol{x}_i)$ and $p_{D'|\text{not } H}(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N) = \prod_{i=1}^N p_2(\boldsymbol{x}_i)$. Thus

$$\frac{\Pr[H \mid D]}{\Pr[\text{not } H \mid D]} = \left(\prod_{i=1}^N \frac{p_1(\boldsymbol{x}_i)}{p_2(\boldsymbol{x}_i)}\right) \frac{\Pr[H]}{\Pr[\text{not } H]}.$$

The odds ratio for hypothesis $H$ is improved by a factor of $\prod_{i=1}^N (p_1(\boldsymbol{x}_i)/p_2(\boldsymbol{x}_i))$.

#### 7.3.3.2 Hypothesis Testing Using $p$-values

The idea here is to compute a statistic $S$ from the data set $D$, and see how consistent the value of this statistic is with a hypothesis to be tested called the *null hypothesis* $H_0$. If the computed value of the statistic $S$ for data set $D$ is $s$, then we test if

$$\Pr[S \geq s \mid H_0] < p \tag{7.3.7}$$

where $0 < p < 1$ is a pre-specified $p$-value. Typically, $p$ is taken to be 0.05 or 0.01, representing testing at the 5% and 1% levels, respectively. If $\Pr[S \geq s \mid H_0] < p$ then we reject $H_0$, the null hypothesis. The idea is that under the null hypothesis, the chance of seeing the value of the statistic $S$ as extreme as the observed value $s$ is less than $p$.

For example, suppose a number of values $x_1, x_2, \ldots, x_N$ are measured. Assume that the null hypothesis $H_0$ is that each $x_i \sim \text{Normal}(0, \sigma^2)$ independently with $\sigma^2$ given. If the statistic $S$ is the mean of the values, $S = (1/N)\sum_{i=1}^{N} x_i$, under the null hypothesis, $S \sim \text{Normal}(0, \sigma^2/N)$. If $\Pr[S \geq s \mid H_0] = \int_s^\infty (2\pi\sigma^2/N)^{-1/2} \exp(-Nz^2/(2\sigma^2))\, dz < p$ then we reject the null hypothesis.

While this is a standard method for identifying when "something interesting" is happening, there are a number of ways in which this approach can fail. This is particularly true where there are multiple tests or tests of multiple hypotheses. Situations like this often arise with, for example, high-throughput testing. For example, genetic markers can be tested for connection with, say, cancer likelihood. The null hypothesis for a given genetic marker would be that the genetic marker has no effect on the cancer likelihood. If we have an estimate $p_0$ of the "background" probability of a certain cancer occurring, the null hypothesis would be that a person with the specified genetic marker has probability $p_0$ of having cancer. Under the null hypothesis, the number of people in a random sample of $N$ people with the genetic marker that also have cancer is a random variable with the Binomial$(N, p_0)$ distribution. The statistic $S$ used would be the number of people in the sample with cancer. Then

$$\Pr[S \geq k] = \sum_{j=k}^{N} \binom{N}{j} p_0^j (1 - p_0)^{N-j}.$$

If $\Pr[S \geq k] < p$ for a threshold $p$-value, then the null hypothesis would be rejected: the genetic marker seems to be positively correlated with cancer.

With high throughput testing, multiple genetic markers are tested simultaneously. If there are $m$ distinct genetic markers tested, then (assuming independence of the genetic markers) then the probability that *some* genetic marker is deemed significant for cancer for the given value of $p \ll 1$ is $1 - (1 - p)^m \approx 1 - \exp(-mp)$. If $mp$ is *not* small, then there is a significant probability of a "positive" result even if the null hypothesis holds for all tested genetic markers. Positivity bias is also important: one study might look at $m$ potential genetic markers. But different groups and laboratories might also be looking at other genetic markers. If there are $M$ groups, each studying $m$ genetic markers, the probability that *someone* will find a "positive" result is fairly high if $Mmp$ is not small even if the null hypothesis is true. Even if each group is careful to make sure that $mp$ is small, the group that finds a "positive" result would not know about all the other $M - 1$ groups with the negative results: the groups with negative results usually do not publish their results, but the group with the positive result would.

Bayesian inference is not immune to these problems. If enough studies are done, randomness will ensure that spurious "positive" results will eventually occur whether Bayesian inference or $p$-value style hypothesis testing is used. Independent confirmation should be performed, however the original results are obtained.

**Exercises.**

(1) Given that a collection of random variables $X_i \sim \text{Poisson}(\lambda)$, $i = 1, 2, \ldots, N$, are independent, show that the MLE for $\lambda$ is $\widehat{\lambda}_{MLE} = (\sum_{i=1}^{N} X_i)/N$.

(2) Suppose that $Y_i = \mu + \epsilon_i, i = 1, 2, \ldots, N$, where the $\epsilon_i$ are independent random variables with $\epsilon_i \sim \text{Normal}(0, \sigma^2)$. Show that the MLE's for $\mu$ and $\sigma^2$ minimize $(1/(2\sigma^2)) \sum_{i=1}^{N} (Y_i - \mu)^2 + N \ln(\sqrt{2\pi}\,\sigma)$. Use this to show that the MLE estimators are $\widehat{\mu}_{MLE} = (1/N) \sum_{i=1}^{N} Y_i$ and $\widehat{\sigma}^2_{MLE} = (1/N) \sum_{i=1}^{N} (Y_i - \widehat{\mu}_{MLE})^2$.

(3) From the previous Exercise, show that $\mathbb{E}\left[\widehat{\sigma}^2_{MLE}\right] = (1 - 1/N)\,\sigma^2$ so that $\widehat{\sigma}^2_{MLE}$ is not an unbiased estimator of $\sigma^2$. However, note that it is an asymptotically unbiased estimator as $N \to \infty$.

(4) ⚠ A radioactive tracer is used to identify how much of a certain molecule is within an organ of the body. A Geiger counter is used to estimate this over periods of time. The rate at which the Geiger counter "counts" is $\rho(t) = a\, e^{-bt}$, which decays exponentially as the traced molecule leaves the relevant organ. This means that the total count $C_i$ for a time interval $[t_i, t_{i+1}]$ has a Poisson distribution $C_i \sim \text{Poisson}(\lambda_i)$ where $\lambda_i = \int_{t_i}^{t_{i+1}} a\, e^{-bt}\, dt$. Note that counts $C_i$ and $C_j$ with $i \neq j$ are assumed to be independent. From this develop, an MLE estimate for $a$ and $b$ in terms of the counts $C_i$ assuming that $t_i = i\,h, i = 0, 1, 2, \ldots, N - 1$, where $h$ is the spacing between Geiger counter readings.

(5) ⚠ Another approach to the problem in the previous Exercise of estimating $(a, b)$ for an exponential decay process is to take logarithms to get $\lambda_i = (a/b)[e^{-bt_i} - e^{-bt_{i+1}}] \approx (a/b)(bh)e^{-bt_i} = ah\, e^{-bt_i}$. Taking logarithms gives $\ln \lambda_i \approx \ln(ah) - b\, t_i$. Using $\ln C_i$ as our estimates for $\ln \lambda_i$ we could use a linear fit of $\ln C_i$ against $t_i = i\,h$, and we can use simple least squares for this. However, as $bt_i$ becomes large, $e^{-bt_i}$ and $\lambda_i$ become small and the variance $\mathbb{V}\text{ar}\,[\ln C_i]$ can become large. This can be (partly) compensated for by using a *weighted* least squares linear fit where the data points with small values of $C_i$ are weighted less than larger values of $C_i$. (Data points with $C_i = 0$ should probably be removed.) Develop a method that involves minimizing $\sum_{i=1}^{N} w_i (\ln(C_i) - \ln(ah) - b\, t_i)^2$ where $w_i$ is roughly proportional to $1/\mathbb{V}\text{ar}\,[\lambda_i]$ as estimated from $C_i$.

(6) A cancer test has a false positive probability of 0.1% and a false negative probability of 0.5%. The cancers detected by the test occur in about 2% of the population. If a random person taking the test gets a positive result, what is the chance that the person actually has one of the cancers tested for? If a random person taking the test gets a negative result, what is the change that the person does not have any of the detectable cancers?

(7) A gambler suspects that a tossed coin is biased to come down heads 55% of the time instead of 50%. How many coin tosses would the gambler likely to need to affirm with probability $p < 1\%$ that the coin is unbiased if the coin is,

indeed, biased as suspected? [**Hint:** Assume that biased coin comes down heads for exactly (or nearly exactly) 55% of tosses.]

## 7.4 Random Algorithms

### *7.4.1 Random Choices*

Many practical algorithms use random choices. Here we will look at three algorithms that use random choices: quicksort, primality testing, and estimating $\pi$. The reasons for the random choices differ according to the algorithm. In quicksort, the random choices means that the average case performance occurs with high probability. In primality testing, a test is applied to the random choices made. If the test for any specific choice is negative, then there is no need for any other test. For estimating $\pi$, many random choices are made, and averaging is used to obtain a more accurate estimate.

The $\pi$ estimation algorithm is called a *Monte Carlo algorithm*. Monte Carlo algorithms are inherently random; at no point does a Monte Carlo algorithm have a definite answer. For estimating $\pi$, we have only approximate value of $\pi$ at any stage of the algorithm. On the other hand, the primality testing algorithm can stop with a definite result once a single test is negative. This kind of algorithm is called a *Las Vegas algorithm*. The results of the quicksort algorithm, however, do not depend on the random choices: the result is the input list, but sorted. Random choices do not affect the final result, just the speed of obtaining it; it is neither a Monte Carlo nor a Las Vegas algorithm.

#### 7.4.1.1 Quicksort

One example is the quicksort algorithm [59, Ch. 8] that first selects an element (called the pivot) of a list, and then splits the rest of the list into two sublists consisting of elements less than the pivot, and elements above the pivot. Each sublist is then recursively sorted. If we choose a fixed element as the pivot element, then for some input lists of length $n$ the method takes $\mathcal{O}(n^2)$ comparisons to sort the list, while for most input lists the method takes $\mathcal{O}(n \log n)$ comparisons. If instead, we choose the pivot entry *randomly* and uniformly from the list to sort, the method takes $\mathcal{O}(n \log n)$ comparisons with a very high probability for large $n$.

#### 7.4.1.2 Primality Testing

Another randomized algorithm is a well-known primality testing algorithm. If an odd number $n$ is prime then for every $1 \leq x \leq n-1, x^{(n-1)/2} \equiv (x \mid n) \pmod{n}$ where

$(x \mid z)$ is the Legendre symbol. The Legendre symbol has a number of properties, the most important of which are

- for $n$ prime, if $(x \mid n) = +1$ then $x = y^2 \pmod{n}$ for some $y$, and if $(x \mid n) = -1$ there is no such $y$,
- $(x \mid n) \equiv \pm 1 \pmod{n}$ for any $x \not\equiv 0 \pmod{n}$,
- $(x \mid n) \equiv (z \mid n) \pmod{n}$ whenever $x \equiv z \pmod{n}$, and
- $(x \mid z) \equiv (-1)^{(x-1)(z-1)/4}(z \mid x) \pmod{n}$.

If $n$ is *not* prime, then for half of the integers $x$ between one and $n-1$, $x^{(n-1)/2} \not\equiv (x \mid n) \pmod{n}$. But we do not know which $x$ have this property. The randomization step is to randomly choose $x$ in the range one to $n-1$. The value of $x^{(n-1)/2}$ modulo $n$ can be computed in $\mathcal{O}(\log n)$ multiplications modulo $n$. The value of $(x \mid n)$ can be computed in $\mathcal{O}(\log n)$ arithmetic operations using the above properties of the Legendre symbol. So if $n$ is *not* prime, with probability $1/2$, $x^{(n-1)/2} \equiv (x \mid n) \pmod{n}$ and with probability $1/2$ $x^{(n-1)/2} \not\equiv (x \mid n) \pmod{n}$. The randomized algorithm repeatedly picks $x \in \{1, 2, \ldots, n-1\}$ and checks if $x^{(n-1)/2} \equiv (x \mid n) \pmod{n}$. This is repeated up to $m$ times, but terminated immediately if $x^{(n-1)/2} \not\equiv (x \mid n) \pmod{n}$ as this indicates that $n$ is *not* prime. If all of the randomly chosen $x$ we have used satisfy $x^{(n-1)/2} \equiv (x \mid n) \pmod{n}$, we can claim that $n$ is *probably* prime.

#### 7.4.1.3 Buffon Needle Problem

In 1733, the Comte de Buffon (also known as Georges-Louis Leclerc) posed the problem of determining the probability of a needle of length $\ell$ crossing a set of parallel lines with common spacing $s$ [36]. This probability can be computed to be $2\ell/(\pi s)$ for $\ell \leq s$. Performing this experiment with many "needles" and recording the fraction of the needles that cross one of the parallel lines can give approximate values for $\pi$. This was supposedly done by the Italian mathematician Mario Lazzarini dropping a needle 3408 times to obtain the value 335/113, which is correct to six digits [15]. To say that Lazzarini was rather lucky to get such an accurate value of $\pi$ is something of an understatement. To expect this kind of accuracy would require more like $10^{12}$ needle drops than a few thousand. However, the method can be implemented in software as a Monte Carlo method to estimate $\pi$.

### *7.4.2 Monte Carlo Algorithms and Markov Chains*

Markov chains are examples of stochastic processes, random processes where the state of the process changes over time. Monte Carlo algorithms use randomness to estimate deterministic quantities, usually of the form $\mathbb{E}[f(\boldsymbol{X})]$ where $\boldsymbol{X}$ is a random variable with a prescribed probability distribution. Monte Carlo Markov Chain (MCMC) methods use Markov chains to generate samples from a target probability

distribution. However, the successive states of a Markov chain are usually not independent. Thus, a Monte Carlo method based on a Markov chain may need to either use widely spaced samples from the Markov chain or compensate in some other way for the lack of independence.

#### 7.4.2.1 Markov Chains

A discrete time *Markov chain* consists of a set of possible states $S$ and a sequence of random variables $X_0,\ X_1,\ X_2,\ \ldots$ with the property that for any measurable subset $E$ of $S$,

$$\Pr\left[X_{t+1} \in E \mid X_0,\ X_1,\ \ldots,\ X_t\right] = \Pr\left[X_{t+1} \in E \mid X_t\right]. \tag{7.4.1}$$

That is, the probability distribution of $X_{t+1}$ depends only on the value of $X_t$, and not on any previous values $X_{t-k}$, $k = 1, 2, \ldots$. The value of $X_t$ is called the *state* of the Markov chain.

If $S$ is also a finite set, then a discrete time Markov chain can be represented by a fixed matrix $P$ where $p_{ij} = \Pr\left[X_{t+1} = i \mid X_t = j\right]$ is the *transition probability* for the transition $j \mapsto i$. Now

$$\Pr\left[X_{t+1} = i\right] = \sum_{j \in S} \Pr\left[X_{t+1} = i \mid X_t = j\right] \Pr\left[X_t = j\right].$$

If $\pi_{t,i} = \Pr\left[X_t = i\right]$ for any $t$ and $\boldsymbol{\pi}_t = \left[\pi_{t,i} \mid i \in S\right]$ is the vector of probabilities of $X_t$, then

$$\boldsymbol{\pi}_{t+1} = P\,\boldsymbol{\pi}_t;$$

$P$ is the *transition matrix* for the Markov chain. If $\boldsymbol{e}$ is the vector of ones of the dimension of $\boldsymbol{\pi}_t$, then since the total probability is one, $\boldsymbol{e}^T\boldsymbol{\pi}_t = 1$ for all $t$. In particular,

$$1 = \boldsymbol{e}^T\boldsymbol{\pi}_{t+1} = \boldsymbol{e}^T P\boldsymbol{\pi}_t$$

for any probability vector $\boldsymbol{\pi}_t$; thus $\boldsymbol{e}^T P = \boldsymbol{e}^T$. The other property of $P$ is that each entry of $P$ is non-negative since all probabilities are non-negative. Any square matrix $P$ of non-negative entries where $\boldsymbol{e}^T P = \boldsymbol{e}^T$ is called a *stochastic matrix*, and can be the matrix of a Markov chain. A stochastic matrix $P$ where also $P\boldsymbol{e} = \boldsymbol{e}$ is called *doubly stochastic*.

In general, we have four possible combinations of continuous and discrete Markov chains: discrete time and discrete states, discrete time and continuous state, continuous time and discrete state, and also continuous time and continuous state. Continuous time and continuous state problems are best understood as stochastic differential equations. Continuous time and discrete state Markov chains are linear constant coefficient systems of differential equations:

**Algorithm 70** Simulating a discrete time & discrete state Markov chain

```
generator markovdiscrete(x_0, P, U)
  x ← x_0
  while true
    u ← next(U)
    find y: Σ_{i=1}^{y-1} p_ix ≤ u < Σ_{i=1}^{y} p_ix
    x ← y
    yield x
  end while
end generator
```

$$\frac{d\boldsymbol{\pi}}{dt} = A\boldsymbol{\pi},$$

where $a_{ii} \leq 0$, $a_{ij} \geq 0$ if $i \neq j$, and $\boldsymbol{e}^T A = \mathbf{0}^T$. The matrix $A$ is called the *transition rate matrix* for the Markov chain. Discrete time and continuous state can often be represented in terms of integrals for probability distribution functions:

$$\pi_{t+1}(\boldsymbol{x}) = \int_S p(\boldsymbol{x}, \boldsymbol{y})\, \pi_t(\boldsymbol{y})\, d\boldsymbol{y},$$

with the condition that $\int_S p(\boldsymbol{x}, \boldsymbol{y})\, d\boldsymbol{x} = 1$ for all $\boldsymbol{y} \in S$. More generally, probability distributions should be represented by measures, in which case the formula should be given by

$$\pi_{t+1}(E) = \int_E \int_S \pi_t(d\boldsymbol{y})\, \mu(d\boldsymbol{x}, \boldsymbol{y})$$

where $\mu$ is a function from $S$ to probability measures on $S$ with the property that $\mu(S, \boldsymbol{y}) = 1$ for all $\boldsymbol{y} \in S$. We call $\mu$ the *transition measure*.

#### 7.4.2.2 Simulating Markov Chains

Computing the probability distributions $\pi_t$ gives a great deal of information about the Markov chain, but this is often impractical if the state space is large. For example, if $S$ is a discrete state space, then for a discrete time Markov chain with transition matrix $P$ we can simulate the Markov chain as follows: given $X_t = j$, we sample $X_{t+1}$ from the distribution where $X_{t+1} = i$ with probability $p_{ij}$. For $S = \{1, 2, \ldots, N\}$, we can implement this method in Algorithm 70, where $U$ is a generator that uniformly samples from $[0, 1]$.

Continuous time but discrete state Markov chains with transition rate matrix $A$ can be simulated in different ways. One is to pick a step size $h > 0$ and then set $P_h = (I - h\, A)^{-1}$. The matrix $P_h$ is a stochastic matrix as

$$\begin{aligned}\boldsymbol{e}^T &= \boldsymbol{e}^T I = \boldsymbol{e}^T (I - h\,A) P_h = (\boldsymbol{e}^T - h\,\boldsymbol{e}^T A) P_h \\ &= (\boldsymbol{e}^T - h\,\mathbf{0}^T) P_h = \boldsymbol{e}^T P_h.\end{aligned}$$

We can then apply Algorithm 70 using $P = P_h$ to generate $X_{k\,h}, k = 1, 2, 3, \ldots$. This approach is closely related to the implicit Euler method for differential equations (see 6.1.8). An alternative is inspired by the explicit Euler method and set $P = I + h\,A$. In order for $P$ to be a stochastic matrix, we need $1 + h a_{ii} \geq 0$ for all $i$. Since $a_{ii} \leq 0$, this puts an upper limit on the value of $h$.

Another approach is to identify the transition times: given $X_t = j$, the transition time $\tau$ is the smallest $\tau > 0$ where $X_{t+s} = j$ for all $0 \leq s < \tau$, but there are arbitrarily small $\epsilon > 0$ where $X_{t+\tau+\epsilon} \neq j$. The transition time $\tau$ is a random variable and is distributed according to the exponential distribution with parameter $\lambda = -a_{jj}$: $\tau \sim \text{Exponential}(-a_{jj})$. Note that $a_{jj} \leq 0$ so $\lambda \geq 0$. Then if $0 \leq \alpha < \beta$, $\Pr[r \leq \tau \leq s] = \exp(-\lambda r) - \exp(-\lambda s)$. We can sample from this distribution by setting $\tau \leftarrow -\ln(U)/\lambda$ where $U$ is a random variable uniformly distributed over $[0, 1]$. The state $X_{t+\tau+}$ is then sampled from $S$ with $\Pr\left[X_{t+\tau+} = i\right] = a_{ij} / \sum_{k \neq j} a_{kj}$ for $i \neq j$. All of these samples can be made independently. If $a_{ii} = 0$ then $\tau = +\infty$ and the simulation stops. In this case, the state $i$ is an absorbing state and no transition out of this state is possible.

### 7.4.2.3 Metropolis–Hastings Algorithm

The *Metropolis–Hastings algorithm* gives iterates $x_k$ of a Markov chain over a discrete state space $X$. The inputs to the Metropolis algorithm are a function $q: X \to \mathbb{R}$ with positive values, and a probability distribution function $g(y \mid x)$. This Markov chain has the property that $\Pr(x_k = x) \to q(x) / \sum_{x \in X} q(x)$ as $k \to \infty$, provided the Markov chain is *ergodic*. A discrete Markov chain is ergodic if the probability of any simulation $X_t$ of the Markov chain has $\lim_{t \to \infty} \Pr[X_t = x] > 0$ and this probability is independent of the simulation. This algorithm is shown in Algorithm 71. The original Metropolis algorithm assumed that $g$ is symmetric:

$$g(x \mid y) = g(y \mid x) \qquad \text{for all } x,\ y \in X. \tag{7.4.2}$$

Note that $q$ defines the limiting probability distribution for the iterates. However, the iterates $x_j$ and $x_k$ for $j \neq k$ are not independent. They are, in a sense, asymptotically independent in that $\mathbb{E}\left[\varphi(x_j)\,\psi(x_k)\right] - \mathbb{E}\left[\varphi(x_j)\right] \mathbb{E}\left[\psi(x_k)\right] \to 0$ as $|j - k| \to \infty$ for any $\varphi$ and $\psi: X \to \mathbb{R}$.

By suitably re-interpeting $q$ and $g$ and the formulas involving them, the algorithm can be extended to continuous as well as discrete probability distributions.

**Algorithm 71** Metropolis–Hastings algorithm; $U$ is a generator of independent random numbers with distribution Uniform(0, 1).

```
1  generator metropolishastings(q, g, x_0, n, U)
2    for k = 0, 1, 2, ..., n − 1
3      sample x' from g(x' | x_k)
4      if next(U) < min(1, (q(x')/q(x_k)) (g(x_k | x')/g(x' | x_k)))
5        x_{k+1} ← x'
6      else
8        x_{k+1} ← x_k
9      end if
10     yield x_{k+1}
11   end for
12 end generator
```

**Exercises.**

(1) Carry out the Buffon needle experiment with pseudo-random numbers with $\ell = s$. Use $N = 10^m$, $m = 2, 3, 4, 5$, "tosses" of the needle to estimate $\pi$. Plot the error against $N$ using a log-log plot.

(2) Rather than simulate the entire quicksort algorithm [59, Ch. 8] in this Exercise, we generate the number of comparisons needed to perform quicksort given random input. For each input list of length $n$ we choose a pivot $p$, which is randomly chosen uniformly from $\{1, 2, \ldots, n\}$. The number of comparisons needed to split the input list is $n - 1$. The method then recursively calls itself with sublists of lengths $p - 1$ and $n - p - 1$. Pseudo-code for this function is given below ($g$ is random generator of positive integers):

```
function qssim(n, q)
  if n = 0 or n = 1: return 0; end if
  p ← (next(g) (mod n)) + 1
  return qssim(p − 1, g) + qssim(n − p − 1, g) + (n − 1)
end
```

Implement this function and plot the results over multiple runs for $n = 10^m$, $m = 1, 2, 3, 4, 5$. Perform 100 runs for each value of $n$. How wide is the spread of the values returned? The average number of comparisons is usually given as $\mathcal{O}(n \log n)$. Do your empirical results confirm this theoretical result? How far does the number of comparisons diverge from this average?

(3) A simple continuous time birth–death Markov chain model for a population has the differential equations for $p_n(t) = \Pr[P(t) = n]$ given by

$$\frac{dp_n}{dt} = (n-1)\beta(n-1)\,p_{n-1} - n[\beta(n) + \delta(n)]p_n + (n+1)\delta(n+1)\,p_{n+1}, \qquad \text{for } n > 0,$$
$$\frac{dp_0}{dt} = \delta(1)\,p_1,$$

where $\beta(k)$ is the birth rate for population $k$ and $\delta(k)$ is the death rate for population $k$. These rates are per individual in the population per unit time. Show that the total probability is constant: $(d/dt)\sum_{n=0}^{\infty} p_n(t) = 0$. If $\beta(k) = \widehat{\beta}$ and $\delta(k) = \widehat{\delta}$ for all $k$, show that the average population $\sum_{n=0}^{\infty} n\, p_n(t)$ grows or decays exponentially as $P_0 \exp((\widehat{\beta} - \widehat{\delta})t)$.

(4) The example of the previous Exercise has a peculiar property: once the population reaches zero, it stays at zero. That is, $p_0(t)$ is an increasing function of $t$. This makes the Markov chain an *absorbing Markov chain*: there is a subset $S_0$ of the states $S$ where the probability of transferring from any state in $S_0$ to any state in $S_1 := S \backslash S_0$ is zero. For a continuous Markov chain, the differential equation for the probabilities has the form

$$\frac{d}{dt}\begin{bmatrix} \boldsymbol{p}_0 \\ \boldsymbol{p}_1 \end{bmatrix} = \begin{bmatrix} A_{0,0} & A_{0,1} \\ & A_{1,1} \end{bmatrix}\begin{bmatrix} \boldsymbol{p}_0 \\ \boldsymbol{p}_1 \end{bmatrix}.$$

Show that $\boldsymbol{e}^T \boldsymbol{p}_0(t)$ never decreases ($(d/dt)(\boldsymbol{e}^T \boldsymbol{p}_0(t)) \geq 0$) and $\boldsymbol{e}^T \boldsymbol{p}_1(t)$ never increases. Also show that there is an eigenvalue $\mu > 0$ and an eigenvector $\widehat{\boldsymbol{p}}_1$ with non-negative components where $-A_{1,1}\widehat{\boldsymbol{p}}_1 = \mu\widehat{\boldsymbol{p}}_1$. [**Hint:** For the last part, $A_{11}$ is a matrix with negative diagonal entries and non-negative off-diagonal entries. Writing $-A_{1,1} = D - N$, note that $d_{ii} \geq \sum_j n_{ij}$ so for any $\epsilon > 0$, $\epsilon I - A_{1,1}$ is a diagonally dominant matrix with positive diagonal entries and non-negative off-diagonal entries, so $(\epsilon I - A_{1,1})^{-1}$ exists and has only non-negative entries. The dominant eigenvalue of this inverse gives approximations $\mu_\epsilon$ that converge to $\mu$ as $\epsilon \downarrow 0$.]

(5) Suppose that $G = (V, E)$ is a connected undirected graph with transitions between vertices $x \to y$ having probability $p_{y,x} = 1/\deg_G(x)$ at each step for each $y$ directly connected to $x$ by an edge ($y \sim x$). Show that the equilibrium probability distribution over the vertices of $G$ is given by $\pi_x = \deg_G(x)/\sum_{z:z\in V} \deg_G(z) = \deg_G(x)/(2\,|E|)$.

(6) Considering the previous Exercise, what should be the transition probabilities to ensure that the equilibrium probability distribution is uniform ($\pi_x = 1/\,|V|$)? [**Hint:** Transitions $x \mapsto x$ should be allowed and given a suitable transition probability.]

(7) Simulate a continuous-time Markov chain model of an infection in a population: suppose everyone is either susceptible ($S$), infected ($I$), or recovered ($R$). Let $s$ be the number of susceptible individuals, $i$ the number of infected individuals, and $r$ the number of recovered individuals. Assuming either population gain or loss, $s + i + r = N$, the total population so $r = N - s - i$. The transitions are: (1) infected person recovers $(s, i, r) \mapsto (s, i - 1, r + 1)$; (2) susceptible person becomes infected $(s, i, r) \mapsto (s - 1, i + 1, r)$; and recovered person becomes susceptible $(s, i, r) \mapsto (s + 1, i, r - 1)$. Transition (1) has a rate $\alpha\, i$; transition (2) has a rate $\beta\, s\, i$; transition (3) has rate $\gamma\, r = \gamma(N - s - i)$. Use $\alpha = 10^{-1}$, $\beta = 10^{-2}$, and $\gamma = 10^{-3}$, with $N = 100$. Initially, assume all but five individuals are susceptible, and five are infected. Note that this is an absorbing

Markov chain as once the number of infected people becomes zero, no more infections can occur. The time step needed for numerically solving this Markov chain may be quite small, especially for an explicit method: $\Delta t \approx 10^{-2}$ perhaps. Try repeating this example with $N = 1000$.

(8) ⚠The *Google PageRank algorithm* is based on the idea of a Markov chain arising from the network of web pages. Each state of the PageRank Markov chain is a web page. Every web page is linked to zero or more other web pages. Provided there is at least one outgoing link on a given web page, the transitions from this web page are to all web pages linked from the current web page. Each link is equally likely to be chosen; this gives the transition probabilities. If a given web page has no outgoing links, then the transitions from this web page are to *every* web page, with each web page having equal probability. These rules define a transition matrix $P$, which is a sparse matrix plus a rank-1 matrix (for the web pages with no outgoing links). The actual transition matrix used in the PageRank algorithm is $P_\alpha = (1-\alpha)P + \alpha \boldsymbol{e}\boldsymbol{e}^T/N$ where $\boldsymbol{e} = [1, 1, 1, \ldots, 1]^T$ and $N$ is the total number of web pages. Show that $P_\alpha$ is a transition matrix for a Markov chain provided $0 < \alpha < 1$. The actual PageRank algorithm is to obtain the equilibrium probability distribution $\boldsymbol{\pi}$ for the Markov chain represented by $P_\alpha$ (usually with $\alpha \approx 0.15$) and to rank order any set of web pages found in a search in decreasing order of $\pi_k$ for web page $k$. The equilibrium distribution is found by means of the power method: $\boldsymbol{\pi}^{(t+1)} \leftarrow P_\alpha \boldsymbol{\pi}^{(t)}$, $t = 0, 1, 2, \ldots$. Implement the PageRank algorithm and apply it to a network of your own devising or to a network obtained by scraping the World Wide Web.

## 7.5 Stochastic Differential Equations

*Stochastic differential equations* combine randomness with differential equations, and if they are autonomous (in the sense that time does not appear explicitly in the equation) then these form a continuous time and continuous state Markov chain. A stochastic differential equation, in general, has the form

$$d\boldsymbol{X}_t = \boldsymbol{f}(t, \boldsymbol{X}_t)\, dt + \sigma(t, \boldsymbol{X}_t)\, d\boldsymbol{W}_t. \tag{7.5.1}$$

Here $\boldsymbol{W}_t$ represents a *Brownian motion*, named after Robert Brown who in 1827 observed pollen particles[1] under a microscope, moving in water in a random and jittery way without apparent cause. The cause was in fact the collisions of water molecules with the pollen particles. The frequent collisions with water molecules push the pollen particles along a random walk with frequent small steps.

[1] Brown was actually observing organelles of the pollen moving, not the pollen grains themselves moving.

Stochastic differential equations are the subject of Øksendal's book [191]; numerical methods for stochastic differential equations are the subject of the book by Kloeden and Platen [144].

### 7.5.1 Wiener Processes

Our understanding of Brownian motion was developed by Einstein, Smoluchowski, and Norbert Wiener. It was Wiener who put it all on a rigorous foundation. For this reason we often refer to *Wiener processes*. The first property of Wiener processes is that it is an *independent increments* process: that is, for $a < b < c$ the random variables $\boldsymbol{W}_c - \boldsymbol{W}_b$ and $\boldsymbol{W}_b - \boldsymbol{W}_a$ are independent. The second property is translation invariance: $\boldsymbol{W}_b - \boldsymbol{W}_a$ has the same distribution as $\boldsymbol{W}_{b-a} - \boldsymbol{W}_0$. The third property is that $\boldsymbol{W}_b - \boldsymbol{W}_a$ has finite variance. Because of the independent increments property and finite variance,

$$\begin{aligned}\mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_c - \boldsymbol{W}_a\right] &= \mathbb{V}\mathrm{ar}\left[(\boldsymbol{W}_c - \boldsymbol{W}_b) + (\boldsymbol{W}_b - \boldsymbol{W}_a)\right] \\ &= \mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_c - \boldsymbol{W}_b\right] + \mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_b - \boldsymbol{W}_a\right].\end{aligned}$$

Combined with translation invariance, we see that $\mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_c - \boldsymbol{W}_a\right]$ must be proportional to $c - a$. The final property is that $\mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_1 - \boldsymbol{W}_0\right] = I$. Then $\mathbb{V}\mathrm{ar}\left[\boldsymbol{W}_c - \boldsymbol{W}_a\right] = (c-a)I$. It is a standard Wiener process if, in addition, $\boldsymbol{W}_0 = \boldsymbol{0}$ with probability one.

The Central Limit Theorem (Theorem 7.4) implies that each $\boldsymbol{W}_t$ and difference $\boldsymbol{W}_b - \boldsymbol{W}_a$ must be normally distributed.

We can create approximate Wiener processes by selecting a time-step $h > 0$, and then take $\boldsymbol{W}_{h,t}$ to be the linear interpolant over $t$ of $\boldsymbol{W}_{h,kh}$, $k = 0, 1, 2, \ldots$ with $\boldsymbol{W}_{h,0} = \boldsymbol{0}$ and $\boldsymbol{W}_{h,(k+1)h} = \boldsymbol{W}_{h,kh} + \sqrt{h}\,\boldsymbol{Z}_k^{(h)}$ where $\boldsymbol{Z}_k^{(h)} \sim \mathrm{Normal}(\boldsymbol{0}, I)$ is an independent increment. The trouble with this approach is that we cannot take meaningful limits as $h \downarrow 0$: unless we have some relationship between $\boldsymbol{Z}_k^{(h)}$ and $\boldsymbol{Z}_{2k}^{(h/2)}$, for example, we cannot expect the $\boldsymbol{W}_{h,t}$ to converge as $h \downarrow 0$.

To see how to create a convergent sequence $\boldsymbol{W}_{h,kh}$, we focus on the scalar case. The vector case can be dealt with by treating each component independently.

Start with the values $W_{1,0} = 0$ and $W_{1,1} \sim \mathrm{Normal}(0, 1)$ and "fill in" the values between $t = 0$ and $t = 1$. For $h = 1/2$ we set $W_{1/2,0} = W_{1,0}$ and $W_{1/2,1} = W_{1,1}$. However, we need to determine a value $W_{1/2,1/2}$ so that $W_{1/2,1/2} - W_{1/2,0}$ and $W_{1/2,1} - W_{1/2,1/2}$ are independent and are both distributed as $\mathrm{Normal}(0,\ 1/2)$. We generate an independent normally distributed sample $U_1^{(1/2)} \sim \mathrm{Normal}(0,\ 1/4)$ and set $W_{1/2,1/2} = \frac{1}{2}(W_{0,0} + W_{1,1}) + U_1^{(1/2)}$. We chose $U_1^{(1/2)}$ to have variance $1/4$ so that the variances add properly:

$$\frac{1}{2} = \mathbb{V}\mathrm{ar}\left[W_{1/2,1/2}\right] = \mathbb{V}\mathrm{ar}\left[\frac{1}{2}(W_{0,0} + W_{1,1})\right] + \mathbb{V}\mathrm{ar}\left[U_1^{(1/2)}\right] = \frac{1}{4} + \mathbb{V}\mathrm{ar}\left[U_1^{(1/2)}\right].$$

Since $(W_{1/2,0},\ W_{1/2,1/2},\ W_{1,1/2})$ are jointly normally distributed, to show independence of the increments it suffices to show that they have zero correlation. Since all means are zero, this reduces to showing that $\mathbb{E}\left[(W_{1/2,1/2} - W_{1/2,0})(W_{1/2,1} - W_{1/2,1/2})\right] = 0$:

$$\begin{aligned}
&\mathbb{E}\left[\left(W_{1/2,1/2} - W_{1/2,0}\right)\left(W_{1/2,1} - W_{1/2,1/2}\right)\right] \\
&\quad = \mathbb{E}\left[\left(\frac{1}{2}(W_{1,0} - W_{1,1}) + U_1^{(1/2)}\right)\left(\frac{1}{2}(W_{1,0} - W_{1,1}) - U_1^{(1/2)}\right)\right] \\
&\quad = \mathbb{E}\left[\frac{1}{4}(W_{1,0} - W_{1,1})^2 - \left(U_1^{(1/2)}\right)^2\right] = \frac{1}{4} - \mathbb{V}\mathrm{ar}\left[U_1^{(1/2)}\right] = 0.
\end{aligned}$$

Also note that the variances $\mathbb{V}\mathrm{ar}\left[W_{1/2,1/2} - W_{1/2,0}\right] = \mathbb{V}\mathrm{ar}\left[W_{1/2,1} - W_{1/2,1/2}\right] = 1/2$.

In general, suppose that we have constructed $W_{H,Hk}$ for $k = 0, 1, 2, \ldots, 2^m$ and $H = 2^{-m}$. Now for $h = 2^{-m-1}$ we wish to construct $W_{h,h\ell}$ for $\ell = 0, 1, 2, \ldots, 2^{m+1}$. For even $\ell = 2j$ we set $W_{h,h2j} = W_{H,j}$. For odd $\ell = 2j + 1$ we set $W_{h,h(2j+1)} = \frac{1}{2}(W_{H,Hj} + W_{H,H(j+1)}) + U_j^{(h)}$ where $U_j^{(h)}$ as an independent sample of the Normal$(0,\ \frac{1}{2}h)$ distribution. The arguments used above for the case $H = 1$ and $h = \frac{1}{2}$ can be applied here: the random variables $(W_{h,h2j},\ W_{h,h(2j+1)},\ W_{h,h(2j+2)})$ are jointly normally distributed, and $\mathbb{E}\left[\left(W_{h,h(2j+1)} - W_{h,h2j}\right)\left(W_{h,h(2j+2)} - W_{h,h(2j+1)}\right)\right] = 0$. This shows that the increments $W_{h,h(2j+1)} - W_{h,h2j}$ and $W_{h,h(2j+2)} - W_{h,h(2j+1)}$ are independent. Standard calculations show that $\mathbb{V}\mathrm{ar}\left[W_{h,h(2j+1)} - W_{h,2hj}\right] = \mathbb{V}\mathrm{ar}\left[W_{h,h(2j+2)} - W_{h,h(2j+1)}\right] = h$.

Assuming that for integers $0 \le p \le q \le r \le 2^m$ the increments $W_{H,Hq} - W_{H,Hp}$ and $W_{H,Hr} - W_{H,Hq}$ are independent with variances $H \cdot (q - p)$ and $H \cdot (r - q)$, respectively, then we can show the corresponding result for $W_{h,h\ell}$, $\ell = 0, 1, 2, \ldots, 2^{m+1}$, using the independence and variance results of the previous paragraph.

We now have a sequence of functions $t \mapsto W_{h,t}$ for $h = 2^{-m}$, $m = 0, 1, 2, \ldots$, each of which has the property that for $h = 2^{-m}$ and $0 \le p \le q \le r$ the increments $W_{h,hq} - W_{h,hp}$ and $W_{h,hr} - W_{h,hq}$ are independent with variances $h(q - p)$ and $h(r - q)$, respectively. Furthermore, if $H = 2^{-s}$ for some positive integer $s$, then for $h = 2^{-m}$ with $m > s$ we have $W_{H,Hj} = W_{h,Hj}$ as $Hj = h\,2^{(m-s)}\,j$ and the construction used.

It can also be shown that the limit $t \mapsto W_t$ is a continuous function; in fact, with probability one, $W_t$ is Hölder continuous with exponent $\alpha$ for any $0 < \alpha < 1/2$. In fact, we do a little better. Lévy [161] was able to show that for any $T > 0$, with probability one,

$$\limsup_{\epsilon \downarrow 0} \sup_{|s-t| \le \epsilon,\ s,t \in [0,1]} \frac{|W_t - W_s|}{\sqrt{2\epsilon\ \ln(1/\epsilon)}} = 1. \tag{7.5.2}$$

A modern proof of this result can be found in [25, Thm. 10.2, p. 70]. However, with probability one $W_t$ is not differentiable anywhere, nor does it have bounded

variation[2]. This means we cannot interpret integrals $\int_a^b g(t)\,dW_t$ as Lebesgue–Stieltjes integrals. This has an impact on how we can interpret and numerically solve stochastic differential equations.

### 7.5.2 Itô Stochastic Differential Equations

Interpreting an ordinary differential equation (ODE)

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}(t, \boldsymbol{x}(t)), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0$$

can be done in terms of integrals:

$$\boldsymbol{x}(t) = \boldsymbol{x}_0 + \int_{t_0}^{t} \boldsymbol{f}(s, \boldsymbol{x}(s))\,ds \qquad \text{for all } t \geq t_0.$$

Interpreting the stochastic differential equation (SDE)

$$d\boldsymbol{X}_t = \boldsymbol{f}(t, \boldsymbol{X}_t)\,dt + \sigma(t, \boldsymbol{X}_t)\,d\boldsymbol{W}_t$$

in the sense of Itô in terms of integrals

$$\boldsymbol{X}_t = \boldsymbol{X}_{t_0} + \int_{t_0}^{t} \left[\boldsymbol{f}(s, \boldsymbol{X}_s)\,ds + \sigma(s, \boldsymbol{X}_s)\,d\boldsymbol{W}_s\right]$$

is a more difficult task as we need to interpret the integral $\int_{t_0}^{t} \sigma(s, \boldsymbol{X}_s)\,d\boldsymbol{W}_s$ which involves products of random quantities. A summary of Itô calculus is [191].

A specific example we can consider is $\int_0^t W_s\,dW_s$. If we used standard methods from calculus, a change of variable $u(s) = \frac{1}{2}W_s^2$ would give $du = W_s\,dW_s$ and so $\int_0^t W_s\,dW_s = \frac{1}{2}W_s^2\big|_{s=0}^{s=t} = \frac{1}{2}W_t^2$. Note that its expectation is $\frac{1}{2}t$. On the other hand, if we approximate the integral by the sum

$$\sum_{k=0}^{n-1} W_{hk}(W_{h(k+1)} - W_{hk})$$

where $t = n\,h$ we get a random quantity whose expectation is zero: $W_{h(k+1)} - W_{hk}$ is independent of $W_{hk} = W_{hk} - W_0$ by the independent increments property, so $\mathbb{E}\left[W_{hk}(W_{h(k+1)} - W_{hk})\right] = \mathbb{E}\left[W_{hk}\right]\mathbb{E}\left[W_{h(k+1)} - W_{hk}\right] = 0$. Taking the limit as $h \to 0$ gives $\mathbb{E}\left[\int_0^t W_s\,dW_s\right] = 0$, not $\frac{1}{2}t$.

[2] Having bounded variation $V$ over $[a, b]$ would mean that $\sum_{i=0}^{n-1} \left|W_{t_{i+1}} - W_{t_i}\right| \leq V$ for any sequence $a \leq t_0 < t_1 < \cdots < t_{n-1} < t_n \leq b$ with any integer $n \geq 1$.

This difference can be explained in terms of the Itô formula [191, Lemma 4.2.1]:

**Theorem 7.7** *If $d\boldsymbol{X}_t = \boldsymbol{u}(t)\,dt + \sigma(t)\,d\boldsymbol{W}_t$ in the Itô sense and $Y_t = g(t, \boldsymbol{X}_t)$ where $\boldsymbol{u}$ and $\sigma$ are continuous and $g$ has continuous second derivatives, then*

$$\begin{aligned} dY_t &= \frac{\partial g}{\partial t}(t, \boldsymbol{X}_t)\,dt + \nabla g(t, \boldsymbol{X}_t)^T d\boldsymbol{X}_t + \frac{1}{2} d\boldsymbol{X}_t^T \mathit{Hess}\, g(t, \boldsymbol{X}_t)\, d\boldsymbol{X}_t \\ &= \left[\frac{\partial g}{\partial t}(t, \boldsymbol{X}_t) + \nabla g(t, \boldsymbol{X}_t)^T \boldsymbol{u}(t) + \frac{1}{2}\mathit{trace}\left(\sigma(t)^T \mathit{Hess}\, g(t, \boldsymbol{X}_t)\,\sigma(t)\right)\right] dt \\ &\qquad + \nabla g(t, \boldsymbol{X}_t)^T \sigma(t)\, d\boldsymbol{W}_t. \end{aligned} \tag{7.5.3}$$

The Itô formula applied to $\frac{1}{2}W_t^2$ yields $d\left(\frac{1}{2}W_t^2\right) = W_t\,dW_t - \frac{1}{2}dt$; integrating then gives $\int_0^t W_s\,dW_s = \frac{1}{2}W_t^2 - \frac{1}{2}t$ which has expectation zero.

The integral $\int_{t_0}^t \sigma(s, \boldsymbol{X}_s)\,d\boldsymbol{W}_s$ in the Itô sense is

$$\lim_{n\to\infty} \sum_{k=0}^{n-1} \sigma(hk,\ \boldsymbol{X}_{hk})\left[\boldsymbol{W}_{h(k+1)} - \boldsymbol{W}_{hk}\right] \qquad \text{where } h = t/n. \tag{7.5.4}$$

Solutions exist and are unique provided both $\boldsymbol{f}(t, \boldsymbol{x})$ and $\sigma(t, \boldsymbol{x})$ are Lipschitz in $\boldsymbol{x}$, uniformly in $t$, given the initial value $\boldsymbol{X}_{t_0} = \boldsymbol{x}_0$ [191, Thm. 5.2.1], which can be shown via Picard iteration and the Itô isometry [191, Lemma 3.1.5]: if $\varphi(t)$ is a random variable that is continuous in $t$,

$$\mathbb{E}\left[\left(\int_a^b \varphi(s)\,dW_s\right)^2\right] = \mathbb{E}\left[\int_a^b \varphi(s)^2\,ds\right]. \tag{7.5.5}$$

Given the underlying Wiener process $\boldsymbol{W}_t$, we have existence and uniqueness of solutions of $d\boldsymbol{X}_t = \boldsymbol{f}(t, \boldsymbol{X}_t)\,dt + \sigma(t, \boldsymbol{X}_t)\,d\boldsymbol{W}_t$ with $\boldsymbol{X}_{t_0} = \boldsymbol{x}_0$. This is what we need for simulating a stochastic differential equation. However, for determining the probability distribution of $\boldsymbol{X}_t$, we need a different equation. If $p(t, \boldsymbol{x})$ is the probability density function of $\boldsymbol{X}_t$, we can obtain a differential equation for $p(t, \boldsymbol{x})$. We can do that by using the Itô formula (7.5.3): for any smooth $g(\boldsymbol{x})$, if $d\boldsymbol{X}_t = \boldsymbol{f}(t, \boldsymbol{X}_t)\,dt + \sigma(t, \boldsymbol{X}_t)\,d\boldsymbol{W}_t$ then

$$\begin{aligned} \frac{d}{dt}\mathbb{E}\left[g(\boldsymbol{X}_t)\right] &= \mathbb{E}\left[\nabla g(\boldsymbol{X}_t)^T \boldsymbol{f}(t, \boldsymbol{X}_t) + \frac{1}{2}\mathrm{trace}\left(\sigma(t, \boldsymbol{X}_t)^T \mathrm{Hess}\, g(\boldsymbol{X}_t)\,\sigma(t, \boldsymbol{X}_t)\right)\right] \\ &= \int_{\mathbb{R}^n}\left[\nabla g(\boldsymbol{x})^T \boldsymbol{f}(t, \boldsymbol{x}) + \frac{1}{2}\mathrm{trace}\left(\sigma(t, \boldsymbol{x})^T \mathrm{Hess}\, g(\boldsymbol{x})\,\sigma(t, \boldsymbol{x})\right)\right] p(t, \boldsymbol{x})\,d\boldsymbol{x}. \end{aligned}$$

Integrating by parts assuming that $p(t, \boldsymbol{x}) \to 0$ rapidly as $\|\boldsymbol{x}\| \to \infty$ gives

$$\frac{d}{dt}\mathbb{E}\left[g(\boldsymbol{X}_t)\right] = \int_{\mathbb{R}^n} \left[ -\operatorname{div}\left(p(t,\boldsymbol{x})\boldsymbol{f}(t,\boldsymbol{x})\right) + \frac{1}{2}\sum_{i,j} \frac{\partial^2}{\partial x_i\,\partial x_j}\left(p(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})^T\right)_{ij} \right] g(\boldsymbol{x})\,d\boldsymbol{x}.$$

On the other hand,

$$\frac{d}{dt}\mathbb{E}\left[g(\boldsymbol{X}_t)\right] = \frac{d}{dt}\int_{\mathbb{R}^n} p(t,\boldsymbol{x})\,g(\boldsymbol{x})\,d\boldsymbol{x} = \int_{\mathbb{R}^n} \frac{\partial p}{\partial t}(t,\boldsymbol{x})\,g(\boldsymbol{x})\,d\boldsymbol{x}.$$

Equating the two gives

$$\int_{\mathbb{R}^n} \frac{\partial p}{\partial t}(t,\boldsymbol{x})\,g(\boldsymbol{x})\,d\boldsymbol{x} = \int_{\mathbb{R}^n} \left[ -\operatorname{div}\left(p(t,\boldsymbol{x})\boldsymbol{f}(t,\boldsymbol{x})\right) + \frac{1}{2}\sum_{i,j} \frac{\partial^2}{\partial x_i\,\partial x_j}\left(p(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})^T\right)_{ij} \right] g(\boldsymbol{x})\,d\boldsymbol{x}$$

for all smooth $g$. Then we can conclude that

(7.5.6)
$$\frac{\partial p}{\partial t}(t,\boldsymbol{x}) = -\operatorname{div}\left(p(t,\boldsymbol{x})\boldsymbol{f}(t,\boldsymbol{x})\right) + \frac{1}{2}\sum_{i,j} \frac{\partial^2}{\partial x_i\,\partial x_j}\left(p(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})\sigma(t,\boldsymbol{x})^T\right)_{ij},$$

which is the *Fokker–Planck equation* for the stochastic differential equation.

### 7.5.3 Stratonovich Integrals and Differential Equations

The Itô integral is based on approximations by finite sums of the form

$$\int_a^b f(X_t)\,dY_t \approx \sum_{k=0}^{n-1} f(X_{a+kh})(Y_{a+(k+1)h} - Y_{a+kh}).$$

As was noted in Section 7.5.2, the stochastic integral $\int_0^t W_s\,dW_s$ has expectation zero when interpreted in the sense of Itô. There is another interpretation called the *Stratonovich integral*, which is based on the trapezoidal rule:

(7.5.7)
$$\int_a^b f(X_t)\circ dY_t \approx \sum_{k=0}^{n-1} \frac{1}{2}\left[f(X_{a+kh}) + f(X_{a+(k+1)h})\right](Y_{a+(k+1)h} - Y_{a+kh}).$$

Because of the nature of Wiener processes, the limits as $n \to \infty$ are different for integrals like

$$\int_0^t W_s \circ dW_s \approx \sum_{k=0}^{n-1} \frac{1}{2}(W_{h(k+1)} + W_{hk})(W_{h(k+1)} - W_{hk})$$
$$= \sum_{k=0}^{n-1} \frac{1}{2}(W_{h(k+1)}^2 - W_{hk}^2) = \frac{1}{2}(W_t^2 - W_0^2),$$

matching the naive application of standard calculus rules.

Stratonovich stochastic differential equations have the form

$$d\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t)\,dt + \sigma(\boldsymbol{X}_t) \circ d\boldsymbol{W}_t, \qquad \boldsymbol{X}_0 = \boldsymbol{x}_0. \tag{7.5.8}$$

If integrals are interpreted in the Stratonovich sense,

$$\boldsymbol{X}_t = \boldsymbol{x}_0 + \int_0^t \boldsymbol{f}(\boldsymbol{X}_s)\,ds + \int_0^t \sigma(\boldsymbol{X}_t) \circ d\boldsymbol{W}_t,$$

then the solutions are solutions in the sense of Stratonovich.

For scalar stochastic differential equations, the Stratonovich equation

$$dX_t = f(X_t)\,dt + \sigma(X_t) \circ dW_t$$

is equivalent to the Itô equation

$$dX_t = \left(f(X_t) + \frac{1}{2}\sigma(X_t)\sigma'(X_t)\right)dt + \sigma(X_t)\,dW_t.$$

For the vector Stratonovich equation

$$d\boldsymbol{X}_t = \boldsymbol{f}(\boldsymbol{X}_t)\,dt + \sigma(\boldsymbol{X}_t) \circ d\boldsymbol{W}_t, \qquad \boldsymbol{X}_0 = \boldsymbol{x}_0,$$

the equivalent Itô equation is

$$d\boldsymbol{X}_t = \left(\boldsymbol{f}(\boldsymbol{X}_t) + \frac{1}{2}\boldsymbol{c}(\boldsymbol{X}_t)\right)dt + \sigma(\boldsymbol{X}_t)\,d\boldsymbol{W}_t \qquad \text{where}$$
$$c_i(\boldsymbol{x}) = \sum_{k=1}^{n}\sum_{j=1}^{n} \frac{\partial \sigma_{ik}}{\partial x_j}(\boldsymbol{x})\,\sigma_{jk}(\boldsymbol{x}).$$

Itô equations can also be represented as Stratonovich equations by replacing $\boldsymbol{f}(\boldsymbol{X}_t)$ with $\boldsymbol{f}(\boldsymbol{X}_t) - \frac{1}{2}\boldsymbol{c}(\boldsymbol{X}_t)$ and replacing $\sigma(\boldsymbol{X}_t)\,d\boldsymbol{W}_t$ by $\sigma(\boldsymbol{X}_t) \circ d\boldsymbol{W}_t$.

### 7.5.4 Euler–Maruyama Method

The *Euler–Maruyama method* is essentially the Euler method applied to the stochastic differential equation

$$d\boldsymbol{X}_t = \boldsymbol{f}(t, \boldsymbol{X}_t)\, dt + \sigma(t, \boldsymbol{X}_t)\, d\boldsymbol{W}_t, \qquad \boldsymbol{X}_0 = \boldsymbol{x}_0.$$

For a step size $h > 0$, the method consists of the iteration

$$\widehat{\boldsymbol{X}}^h_{h(k+1)} = \widehat{\boldsymbol{X}}^h_{hk} + h\, \boldsymbol{f}(t, \widehat{\boldsymbol{X}}^h_{hk}) + \sigma(hk,\ \widehat{\boldsymbol{X}}^h_{hk})(\boldsymbol{W}_{h(k+1)} - \boldsymbol{W}_{hk}) \tag{7.5.9}$$

where the numerical solution is $\widehat{\boldsymbol{X}}^h_{hk} \approx \boldsymbol{X}_{hk}$, $k = 0, 1, 2, \ldots$, with initial value $\widehat{\boldsymbol{X}}_0 = \boldsymbol{x}_0$.

An example of the result of the Euler–Maruyama method is shown in Figure 7.5.1. This shows solutions for

$$dX_t = r\, X_t\, dt + s\, X_t\, dW_t, \qquad X_0 = 1,$$

with $r = s = 1$. This is a model for price evolution with a natural interest or inflation rate of $r$ and volatility $s$. The trajectories in Figure 7.5.1(a) use the same underlying Wiener process. The details, in case you want to reconstruct this solution, are as follows: the approximate Wiener process was created using Matlab's `randn` function based on the Mersenne Twister generator with seed 95324965 to generate $2^{20}$ pseudo-random distributed according to the Normal$(0, 2^{-20})$ distribution; then cumulative sums were used to give the values of $W_{hk}$ with $h = 2^{-20}$ and $k = 0, 1, 2, \ldots, 2^{20}$.

The convergence of the trajectories, rather than just the statistical properties of the trajectories, illustrates *strong convergence* of the approximate trajectories for this method. That is, there are positive constants $C$ and $h_0$ where

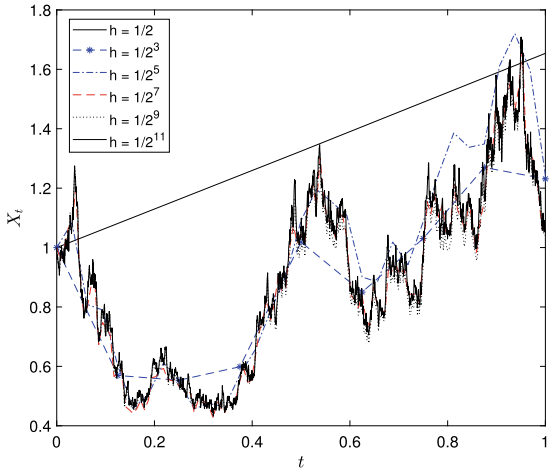
$$\max_{0 \le hk \le T} \mathbb{E}\left[\left\| \widehat{\boldsymbol{X}}^h_{hk} - \boldsymbol{X}_{hk} \right\|\right] \le C\, h^\alpha \qquad \text{for all } 0 < h \le h_0. \tag{7.5.10}$$

Weak convergence means that for any smooth function $g(\boldsymbol{x})$,

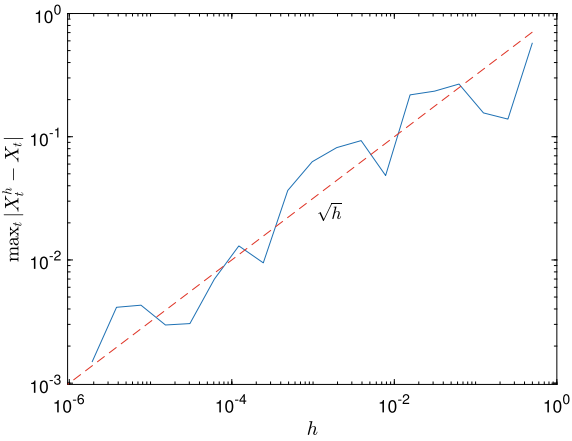
$$\max_{0 \le hk \le T} \mathbb{E}\left[g(\widehat{\boldsymbol{X}}^h_{hk}) - g(\boldsymbol{X}_{hk})\right] = \mathcal{O}(h^\alpha) \qquad \text{as } h \downarrow 0. \tag{7.5.11}$$

Figure 7.5.1(b) shows the corresponding maximum errors. Note that the dashed line in Figure 7.5.1(b) is the plot of $\sqrt{h}$.

Convergence theory for the Euler–Maruyama method is developed in, for example, Kloeden and Platen [144, Thm. 10.2.2]. The error is on average $\mathcal{O}(\sqrt{h})$, and $\mathcal{O}(\sqrt{h \ln h})$ with probability one.




(A) Trajectories




(B) Error behavior

**Fig. 7.5.1** Results of the Euler–Maruyama method

## *7.5.5 Higher Order Methods for Stochastic Differential Equations*

Creating higher order methods for stochastic differential equations is harder than creating higher order methods for ordinary differential equations, or even partial differential equations. Part of the problem is that multiple integrals of Wiener processes arise, which require special treatment [143]. However, higher order methods can be created using Taylor series and Runge–Kutta approaches [42, 144]. The Taylor series approach involves further random quantities obtained from a given Wiener process.

The Milstein method [143, (4.2) on p. 291] for a scalar SDE in Itô sense is

$$X^h_{h(k+1)} = X^h_{hk} + f(X^h_{hk})\,h + \sigma(X^h_{hk})(W_{h(k+1)} - W_{hk}) + \\ + \frac{1}{2}\sigma(X^h_{hk})\sigma'(X^h_{hk})\left[(W_{h(k+1)} - W_{hk})^2 - h\right]. \tag{7.5.12}$$

This method is strongly convergent with order 1, so that the expected global error is $\mathcal{O}(h)$. There is also an implicit version of the Milstein method

$$X^h_{h(k+1)} = X^h_{hk} + f(X^h_{h(k+1)})\,h + \sigma(X^h_{hk})(W_{h(k+1)} - W_{hk}) + \\ + \frac{1}{2}\sigma(X^h_{hk})\sigma'(X^h_{hk})\left[(W_{h(k+1)} - W_{hk})^2 - h\right], \tag{7.5.13}$$

which will also strongly convergent with order 1. Higher order Taylor series methods involve higher order derivatives of $f$ and $\sigma$.

Runge–Kutta methods for stochastic differential equations are given in Burrage and Burrage [40]. Burrage and Burrage first mention explicit $s$-stage stochastic Runge–Kutta methods of the form

$$\boldsymbol{Y}_{k,i} = \widehat{\boldsymbol{X}}^h_{hk} + h\sum_{j=1}^{s} a_{ij}\boldsymbol{f}(\boldsymbol{Y}_{k,j}) + \sum_{j=1}^{s} b_{ij}\sigma(\boldsymbol{Y}_{k,j})\boldsymbol{J}_1, \qquad i = 1, 2, \ldots, s, \tag{7.5.14}$$

$$\widehat{\boldsymbol{X}}^h_{h(k+1)} = \widehat{\boldsymbol{X}}^h_{hk} + h\sum_{j=1}^{s} \alpha_j\boldsymbol{f}(\boldsymbol{Y}_{k,j}) + \sum_{j=1}^{s} \beta_j\sigma(\boldsymbol{Y}_{k,j})\boldsymbol{J}_1. \tag{7.5.15}$$

Here $\boldsymbol{J}_1 = \boldsymbol{W}_{h(k+1)} - \boldsymbol{W}_{hk}$, which is the first stochastic integral: $\boldsymbol{J}_1 = \int_{hk}^{h(k+1)} d\boldsymbol{W}_s$. As noted in [40], methods of the form (7.5.14, 7.5.15) cannot have strong order greater than 1.5. That is, (7.5.10) cannot hold with $\alpha > 1.5$, which was shown by Rümelin [221]. This order is, in fact, obtained by the stochastic version of Heun's method:

$$\boldsymbol{Y}_k = \widehat{\boldsymbol{X}}^h_{hk} + h\,\boldsymbol{f}(\widehat{\boldsymbol{X}}^h_{hk}) + \sigma(\widehat{\boldsymbol{X}}^h_{hk})\boldsymbol{J}_1,$$
$$\widehat{\boldsymbol{X}}^h_{h(k+1)} = \widehat{\boldsymbol{X}}^h_{hk} + \frac{1}{2}h\left[\boldsymbol{f}(\widehat{\boldsymbol{X}}^h_{hk}) + \boldsymbol{f}(\boldsymbol{Y}_k)\right] \\ + \frac{1}{2}h\left[\sigma(\widehat{\boldsymbol{X}}^h_{hk}) + \sigma(\boldsymbol{Y}_k)\right]\boldsymbol{J}_1.$$

This method can also be represented by the extended Butcher tableau

$$\begin{array}{c|c} A & B \\ \hline \boldsymbol{\alpha}^T & \boldsymbol{\beta}^T \end{array} = \begin{array}{cc|cc} 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ \hline \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{array}.$$

The key to higher order than 1.5 is to incorporate nested stochastic integrals of Wiener processes. The Milstein method (7.5.12) achieves strong order one instead of $\frac{1}{2}$ by using the nested Itô integral $\int_{hk}^{h(k+1)} \int_{hk}^{s} dW_r\, dW_s = (W_{h(k+1)} - W_{hk})^2 - h$. Burrage and Burrage [40] use nested Stratonovich and Riemann integrals, such as

$$
\begin{aligned}
J_{11,k} &= \int_{hk}^{h(k+1)} \int_{hk}^{s} 1 \circ dW_r \circ dW_s = (W_{h(k+1)} - W_{hk})^2, \\
J_{10,k} &= \int_{hk}^{h(k+1)} \int_{hk}^{s} 1 \circ dW_r \; ds, \\
J_{01,k} &= \int_{hk}^{h(k+1)} \int_{hk}^{s} 1 \; dr \circ dW_s, \\
J_{100,k} &= \int_{hk}^{h(k+1)} \int_{hk}^{s} \int_{hk}^{r} 1 \circ dW_q \; dr \; ds, \qquad \text{etc.}
\end{aligned}
$$

The latter nested integrals cannot be written in terms of $W_{hk}$ and $W_{h(k+1)}$. Instead, they depend on the path taken between these values. If the underlying Wiener process can be computed to higher resolution, then these nested integrals can be computed. Alternatively, samples of appropriate probability distributions can be computed for the values of the nested integrals. Then the resulting numerical solution is an accurate approximation to the true solution for *some* underlying Wiener process with the given values of $W_{hk}$, $k = 0, 1, 2, \ldots$.

**Exercises.**

(1) Solve the stochastic differential equation for a pendulum

$$
\begin{aligned}
d\theta &= \omega\, dt \\
d\omega &= -\frac{g}{\ell} \sin\theta\, dt + \sigma\, dW_t
\end{aligned}
$$

with $g = \ell = 1$ and $\sigma = 10^{-2}$ using the Euler–Maruyama method with step size $\Delta t = 10^{-2}$ over the time interval $[0, 100]$. Use initial conditions $\theta(0) = \pi/3$ and $\omega(0) = 0$. Compare your results with the deterministic equation ($\sigma = 0$) with the same initial conditions, step size, with the Euler method.

(2) For the previous Exercise, write out the Fokker–Planck equation (7.5.6) for the stochastic differential equation. Remember the Fokker–Planck equation is for the probability density function $p(t, \theta, \omega)$.

(3) Repeat Exercise 1 for the van der Pol equation (6.2.14) with the same parameter values except $\Delta t = 10^{-3}$, and with $\mu = 2$. Since the deterministic solution has a strongly stable limit cycle, the solution of the stochastic version with small $\sigma$ should remain close to the deterministic trajectory, although the stochastic solution may be delayed or advanced in time. Check if this is true for your solution to the stochastic equation.

(4) Compare numerical solutions of the stochastic differential equation $dX_t = X_t\,dt + X_t\,dW_t$, $X_0 = 1$ over the time interval $t \in [0, 1]$ using the Euler–Maruyama (7.5.9) and the Milstein methods (6.2.14) with step sizes $\Delta t = 2^{-j}$, $j = 1, 2, \ldots, 10$. Be careful to use the same sample $W_t$ of the standard Wiener process. For the "exact" solution, use the Milstein method with $\Delta t = 2^{-16}$. Plot the maximum error in $X_t$ over $t \in [0, 1]$ against $\Delta t$ for each of the two methods. Use a log-log plot, at least at first, to see the order of convergence.

(5) ⚠In Exercise 7.4.7, a Markov chain model of an infectious disease is given. The state space consists of points $(s, i, r) \in \{0, 1, 2, \ldots\}^3$ with $s + i + r = N$ (the total population) and three different kinds of transitions. If we take the expectation of the change in $(s, i, r)$ and ignore the fact that $(s, i, r)$ are integers, we obtain the "mass action" differential equations

$$\begin{aligned}\frac{ds}{dt} &= -\beta\, s\, i + \gamma\, r,\\ \frac{di}{dt} &= +\beta\, s\, i - \alpha\, i,\\ \frac{dr}{dt} &= -\gamma\, r \quad + \alpha\, i.\end{aligned}$$

However, this ignores the stochastic aspect of the process. In time $\Delta t$, for small $\Delta t$, the transitions are $(s, i, r) \mapsto (s, i-1, r+1)$ with probability $\alpha\, i\, \Delta t + \mathcal{O}(\Delta t)^2$, $(s, i, r) \mapsto (s-1, i+1, r)$ with probability $\beta\, s\, i\, \Delta t + \mathcal{O}(\Delta t)^2$, and $(s, i, r) \mapsto (s+1, i, r-1)$ with probability $\gamma\, r\, \Delta t + \mathcal{O}(\Delta t)^2$. The expected value of the changes is

$$\mathbb{E}\left[((s(t+\Delta t), i(t+\Delta t), r(t+\Delta t)) - (s(t), i(t), r(t)))/\Delta t\right]$$

goes to the right-hand side of the above differential equations as $\Delta t \to 0$. Show that the variance–covariance matrix of the change

$$\mathbb{V}\mathrm{ar}\left[((s(t+\Delta t), i(t+\Delta t), r(t+\Delta t)) - (s(t), i(t), r(t))\right] \sim \Delta t\, V \text{ as } \Delta t \downarrow 0 \quad \text{where}$$

$$V(s,i,r) = \alpha\, i \begin{bmatrix} 0 \\ -1 \\ +1 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ +1 \end{bmatrix}^T + \beta\, i\, s \begin{bmatrix} -1 \\ +1 \\ 0 \end{bmatrix} \begin{bmatrix} -1 \\ +1 \\ 0 \end{bmatrix}^T + \gamma\, r \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}^T.$$

Set $\sigma(t, \boldsymbol{x})$ to be a matrix where $\sigma(t, \boldsymbol{x})\sigma(t, \boldsymbol{x})^T = V(\boldsymbol{x})$ with $\boldsymbol{x} = [s, i, r]^T$. The stochastic differential equation that results is

$$d\begin{bmatrix} S_t \\ I_t \\ R_t \end{bmatrix} = \begin{bmatrix} -\beta\, S_t\, I_t + \gamma\, R_t \\ +\beta\, S_t\, I_t - \alpha\, I_t \\ -\gamma\, R_t \quad + \alpha\, I_t \end{bmatrix} dt + \sigma(t, \boldsymbol{X}_t)\, d\boldsymbol{W}_t, \qquad \boldsymbol{X}_t = \begin{bmatrix} S_t \\ I_t \\ R_t \end{bmatrix}.$$

Solve this stochastic differential equation using the Euler–Maruyama method using the parameter and initial values in Exercise 7.4.7. Compare the results to simulating the Markov chain of Exercise 7.4.7 directly. Note, however, that we can only expect *statistical* similarity over many runs.

(6) Since chaotic differential equations are persistently unstable, any stochastic perturbation (even if very small) will result in large changes in the solution. Consider, for example, the Lorenz equations modeling weather (6.1.13–6.1.15), but convert them to a stochastic differential equation by adding a term "$\sigma\, d\boldsymbol{W}_t$" with $\sigma = 10^{-4}$. Solve with the same initial conditions as for Figure 6.1.2 for a time interval of length 100. Compare the results with the deterministic solution (for $\sigma = 0$).

## Project

A way of generating text is to use a Markov chain. For text that sounds more like English, given a body of text we want to resemble, split the text into words. Choose a positive integer $r$, and for each $r$-tuple $(w_{k-r+1}, \ldots, w_{k-1}, w_k)$ of words, assign a given word $w$ a transition probability of the number of occurrences of word $w$ immediately after words $w_{k-r+1}, \ldots, w_{k-1}, w_k$ in the body of text, divided by the total number of occurrences of $w_{k-r+1}, \ldots, w_{k-1}, w_k$ in the text. This gives the transition probability $(w_{k-r+1}, \ldots, w_{k-1}, w_k) \mapsto (w_{k-r+2}, \ldots, w_k, w)$ in a Markov chain. If $(w_{k-r+1}, \ldots, w_{k-1}, w_k)$ does not occur in the body of text then assign a transition $(w_{k-r+1}, \ldots, w_{k-1}, w_k) \mapsto (w_{k-r+2}, \ldots, w_k, w)$ where $w$ indicates the end of a sentence. Implement this in a suitable programming language (hash tables are good for storing transitions). Pick a suitable value of $r$ (say, 2 or 3). Use some body of text to generate the transition probabilities (Shakespeare is in the public domain at http://shakespeare.mit.edu/, for example). Run the Markov chain to create "text".

# Chapter 8
# Optimization


Optimization is the task of making the most of what you have. Mathematically, this is turned into finding either the maximum or minimum of a function $f: A \to \mathbb{R}$ where $A \subseteq \mathbb{R}^n$ is the *feasible set*, the set of allowed choices. Since maximizing $f(\boldsymbol{x})$ over $\boldsymbol{x} \in A$ is equivalent to minimizing $-f(\boldsymbol{x})$ over $\boldsymbol{x} \in A$, by convention we usually consider *minimizing* $f(\boldsymbol{x})$ over a set $\boldsymbol{x} \in A$. After reviewing the necessary and sufficient conditions for optimization, we turn to numerical methods. Of particular importance is the distinction between convex and non-convex optimization problems.

This chapter can only scratch the surface of the wide range of optimization methods. Nocedal and Wright [190], for example, go into most of the topics covered here in much more depth. Kochenderfer and Wheeler [146], on the other hand, delve into many more algorithms than here but in less depth.

## 8.1 Basics of Optimization

### *8.1.1 Existence of Minimizers*

Of course, there might not be a minimum. The function $f(x) = x$ over $A = \mathbb{R}$ does not have a minimum; this function is *unbounded below*: for any $L \in \mathbb{R}$ there is an $x \in \mathbb{R}$ where $f(x) < L$. Other functions, such as $f(x) = e^x$ over $A = \mathbb{R}$ is bounded below ($f(x) > 0$ for all $x \in \mathbb{R}$), but there is no minimum. Instead, we can make $f(x)$ approach zero as closely as we please, but there is no $x$ where $f(x) = 0$. So the *infimum* of $f(x) = e^x$ over $x \in \mathbb{R}$ exists and is zero, but since the infimum cannot be attained, there is no minimum.

A variation of this idea is that if $A = (0, 1)$, the open interval of $x \in \mathbb{R}$ where $0 < x < 1$, the minimum of $f(x) = x$ over $x \in A$ does not exist either. Again, we have a situation where the infimum of $f(x)$ over $x \in A$ exists (it is zero) but cannot



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7_8

be attained as this would require using $x = 0$, which is not in $A = (0, 1)$. Similarly, if we have a function that is not continuous, say $f(x) = x$ for $x > 0$ but $f(x) = 1$ for $x \leq 0$, then again we have an infimum of $f(x)$ over $x \in \mathbb{R}$ which is finite (again, it is zero), but we cannot attain this value, because $1 = f(0) > \lim_{x \downarrow 0} f(x) = 0$.

We can show existence of a minimizer if $f : A \to \mathbb{R}$ is continuous and $A \subset \mathbb{R}^n$ is both closed and bounded. Since $A$ is *closed*: if $\boldsymbol{x}_k \in A$ for $k = 1, 2, \ldots$ and $\boldsymbol{x}_k \to \boldsymbol{x}$ as $k \to \infty$, then $\boldsymbol{x} \in A$ as well. Also, since $A$ is *bounded*: there is a bound $B$ where $\|\boldsymbol{x}\| \leq B$ for all $\boldsymbol{x} \in A$. Since $A$ is a subset of $\mathbb{R}^n$, the combination of being closed and bounded implies that $A$ is *compact*, meaning that for any sequence $\boldsymbol{x}_k \in A$ for $k = 1, 2, \ldots$, there is subsequence $\boldsymbol{x}_{k_j}$, $j = 1, 2, \ldots$ with $k_{j+1} > k_j$ for all $j$, which converges: $\boldsymbol{x}_{k_j} \to \widehat{\boldsymbol{x}}$ as $j \to \infty$ for some $\widehat{\boldsymbol{x}} \in A$ [1].

**Theorem 8.1** (Heine–Borel theorem) *If $f : A \to \mathbb{R}$ is continuous and $A$ compact, then $f$ has both a minimum and a maximum over $A$.*

***Proof*** If $f$ is unbounded below over $A$ then there is a sequence $\boldsymbol{x}_k \in A$ where $f(\boldsymbol{x}_k) \to -\infty$ as $k \to \infty$. By the compactness of $A$, there is a subsequence $\boldsymbol{x}_{k_j}$ with $k_{j+1} > k_j$ where $\boldsymbol{x}_{k_j} \to \widehat{\boldsymbol{x}}$ as $j \to \infty$. By continuity of $f$, $f(\widehat{\boldsymbol{x}}) = \lim_{j \to \infty} f(\boldsymbol{x}_{k_j}) = -\infty$, which is impossible.

So there must be a lower bound $L$ where $f(\boldsymbol{x}) \geq L$ for all $\boldsymbol{x} \in A$. By the greatest lower bound principle of real numbers, there must be a greatest lower bound $\widehat{L}$ where $L \leq \widehat{L} \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in A$. Since for any positive integer $n$, $\widehat{L} + 1/n$ is *not* a lower bound, there must be a point $\boldsymbol{x}_n \in A$ where $\widehat{L} \leq f(\boldsymbol{x}_n) < \widehat{L} + 1/n$. By compactness, there is a subsequence $\boldsymbol{x}_{n_j}$ where $\boldsymbol{x}_{n_j} \to \widehat{\boldsymbol{x}} \in A$ as $j \to \infty$. Continuity of $f$ then implies that $f(\boldsymbol{x}_{n_j}) \to f(\widehat{\boldsymbol{x}})$ as $j \to \infty$; the Squeeze Theorem then implies that $\widehat{L} = f(\widehat{\boldsymbol{x}}) \leq f(\boldsymbol{x})$ for any $\boldsymbol{x} \in A$. That is, $\widehat{\boldsymbol{x}} \in A$ minimizes $f(\boldsymbol{x})$ over $\boldsymbol{x} \in A$.

To show that $f$ has a maximum over $A$, apply the above argument to $-f$. □

Often we have to deal with the situation where $A = \mathbb{R}^n$, which is *unconstrained optimization*. In this case, Theorem 8.1 cannot be applied because $A = \mathbb{R}^n$ is *not* bounded. Instead we need a suitable condition on $f$. We say $f$ is *coercive* over $A$ if $\boldsymbol{x}_k \in A$ for all $k$ and $\|\boldsymbol{x}_k\| \to \infty$ as $k \to \infty$ implies that $f(\boldsymbol{x}_k) \to \infty$ as $k \to \infty$. If $A = \mathbb{R}^n$ we simply say that $f$ is coercive.

**Theorem 8.2** *If $f : A \to \mathbb{R}$ is continuous and coercive, and $A \subseteq \mathbb{R}^n$ is closed and non-empty, then $f$ has a minimizer over $A$.*

***Proof*** Pick a point $\boldsymbol{x}_0 \in A$. Let $A_0 = \{\, \boldsymbol{x} \in A \mid f(\boldsymbol{x}) \leq f(\boldsymbol{x}_0) \,\}$. The set $A_0$ is called a *level set*. We will show that $A_0$ is both closed and bounded in $\mathbb{R}^n$. Suppose that $\boldsymbol{x}_k \to \boldsymbol{x}$ as $k \to \infty$ and $\boldsymbol{x}_k \in A_0$ for every $k$. Then $\boldsymbol{x}_k \in A$ and $f(\boldsymbol{x}_k) \leq f(\boldsymbol{x}_0)$ for every $k$. Since $\boldsymbol{x}_k \to \boldsymbol{x}$ as $k \to \infty$ and $A$ is closed, $\boldsymbol{x} \in A$. Since $f$ is continuous, $f(\boldsymbol{x}) = \lim_{k \to \infty} f(\boldsymbol{x}_k) \leq f(\boldsymbol{x}_0)$. That is, $\boldsymbol{x} \in A_0$, and so $A_0$ is closed.

---

[1] Technically, what is described here is *sequential compactness*. The proper definition of compactness is best defined in terms of topologies. See, for example, [184, Chap. 3]. A set being sequentially compact is equivalent to being compact in any space with a norm or metric.

To see that $A_0$ is bounded, suppose otherwise. We will show this leads to a contradiction. If $A_0$ is unbounded, then for every $n$ there is an $\boldsymbol{x}_n \in A_0$ where $\|\boldsymbol{x}_n\| > n$. Clearly, $\|\boldsymbol{x}_n\| \to \infty$ as $n \to \infty$. As $f$ is coercive, $f(\boldsymbol{x}_n) \to \infty$. Therefore, there is an $N$ where $n \geq N$ implies $f(\boldsymbol{x}_n) > f(\boldsymbol{x}_0)$. This violates the definition of $A_0$, which is the contradiction we seek.

Thus $A_0$ is both closed and bounded in $\mathbb{R}^n$. It is therefore compact, and so there must be a minimizer $\widehat{\boldsymbol{x}}$ of $f$ over $A_0$. This minimizer in fact minimizes $f$ over all of $A$: $\boldsymbol{x}_0 \in A_0$ by definition of $A_0$. Since $\widehat{\boldsymbol{x}}$ minimizes $f$ over $A_0$, $f(\widehat{\boldsymbol{x}}) \leq f(\boldsymbol{x}_0)$. If $\boldsymbol{x} \in A$ but $\boldsymbol{x} \notin A_0$, then $f(\boldsymbol{x}) > f(\boldsymbol{x}_0) \geq f(\widehat{\boldsymbol{x}})$. Thus for any $\boldsymbol{x} \in A$ whether $\boldsymbol{x} \in A_0$ or not, $f(\boldsymbol{x}) \geq f(\widehat{\boldsymbol{x}})$ and $\widehat{\boldsymbol{x}}$ minimizes $f$ over $A$. □

Telling if a function is coercive can be easy in some cases, like $f(x, y) = x^2 + 2y^2$. Or it can be more challenging, such as for $f(x, y) = e^x - x + y^4 - xy^2$. We can use comparison principles to tell if a function is coercive: if $g$ is coercive and $f(\boldsymbol{x}) \geq g(\boldsymbol{x})$ for all $\boldsymbol{x}$, then $f$ is also coercive. If $f(\boldsymbol{x}, \boldsymbol{y}) = g(\boldsymbol{x}) + h(\boldsymbol{y})$ with $g$ and $h$ both continuous and coercive, then so is $f$. Sums of coercive functions are also coercive. Products of coercive functions are coercive. Products of coercive functions and positive constants are also coercive. Some simple rules like $|ab| \leq \frac{1}{2}(a^2 + b^2)$ can also be really helpful.

**Example 8.3** We can show that $f(x, y) = e^x - x + y^4 - xy^2$ is coercive as follows (start by applying the rule $|ab| \leq \frac{1}{2}(a^2 + b^2)$ to $xy^2$): for $x \geq 0$,

$$\begin{aligned} f(x, y) &= e^x - x + y^4 - xy^2 \\ &\geq e^x - x + y^4 - \frac{1}{2}(x^2 + y^4) \\ &= e^x - x - \frac{1}{2}x^2 + \frac{1}{2}y^4. \end{aligned}$$

For $x \leq 0$,

$$\begin{aligned} f(x, y) &= e^x - x + y^4 - xy^2 \\ &\geq e^x - x + y^4 \\ &\geq -x + \frac{1}{2}y^4. \end{aligned}$$

Now $h(y) = \frac{1}{2}y^4$ is clearly coercive in $y$; $g(x) = \max(e^x - x - \frac{1}{2}x^2, -x)$ is coercive in $x$. Note that $f(x, y) \geq g(x) + h(y)$ for all $(x, y)$ so $f$ is also coercive.

### *8.1.2 Necessary Conditions for Local Minimizers*

What we have been calling the minimizer is also called the *global minimizer* or *absolute minimizer*. This distinguishes it from a *local minimizer*: $\widehat{\boldsymbol{x}}$ is a local minimizer

of $f$ if there is a $\delta > 0$ where $\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\| < \delta$ implies $f(\widehat{\boldsymbol{x}}) \leq f(\boldsymbol{x})$. We also say that $f(\widehat{\boldsymbol{x}})$ is a local minimum. We say $\widehat{\boldsymbol{x}}$ is a *strict local minimizer* if there is a $\delta > 0$ where $0 < \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\| < \delta$ implies $f(\widehat{\boldsymbol{x}}) < f(\boldsymbol{x})$. In this case, we say $f(\widehat{\boldsymbol{x}})$ is a strict local minimum.

We use tools from calculus to find local minimizers. But telling if a local minimizer is also a global minimizer takes extra information. We can look for *all* local minimizers. As long as a global minimizer exists, one of the local minimizers will also be a global minimizer. We simply need to look at the function values at each local minimizer: the minimum of these local minima is the global minimum, again, provided a global minimizer exists.

The usual rule from calculus is that if $x^*$ is a local minimizer of a function $f : \mathbb{R} \to \mathbb{R}$, then $f'(x^*) = 0$. We can extend this to the multivariate case.

**Theorem 8.4** (Fermat's principle) *If $f : \mathbb{R}^n \to \mathbb{R}$ has continuous first derivatives and a local mimimzer $\boldsymbol{x}^*$ then $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$.*

To be clear, Fermat enunciated his principle for a single variable in 1636 [73], before either Newton or Leibniz had developed calculus.

***Proof*** Let $\boldsymbol{d} \in \mathbb{R}^n$ and take $s > 0$. Assume that $\delta > 0$ where $\|\boldsymbol{x} - \boldsymbol{x}^*\| < \delta$ implies that $f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*)$. Then provided $|s|\,\|\boldsymbol{d}\| < \delta$ we have $f(\boldsymbol{x}^* + s\boldsymbol{d}) \geq f(\boldsymbol{x}^*)$. For $s > 0$, subtracting $f(\boldsymbol{x}^*)$ and dividing by $s$ gives

$$\frac{f(\boldsymbol{x}^* + s\boldsymbol{d}) - f(\boldsymbol{x}^*)}{s} \geq 0.$$

Taking the limit as $s \downarrow 0$ gives $\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \geq 0$. For $s < 0$, subtracting $f(\boldsymbol{x}^*)$ and dividing by $s$ gives

$$\frac{f(\boldsymbol{x}^* + s\boldsymbol{d}) - f(\boldsymbol{x}^*)}{s} \leq 0.$$

Taking the limit as $s \uparrow 0$ gives $\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \leq 0$. So $0 \leq \nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \leq 0$; that is, $\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} = 0$. Since this is true for any $\boldsymbol{d}$, $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$. □

Any point $\boldsymbol{x}$ where $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ is called a *critical point*. If we can find all critical points, we can determine which gives the smallest function value; that critical point is the global minimizer, provided a global minimizer exists. But the condition $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ does not imply that $\boldsymbol{x}$ is even a local minimizer. In calculus, we look for second derivatives: $f''(x) \geq 0$ is a necessary condition for a local minimizer. Generalizing to the multivariate situation, we have the following theorem.

**Theorem 8.5** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ has continuous second order derivatives. Suppose also that $\boldsymbol{x}^*$ is a local minimizer. Then $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$ and Hess $f(\boldsymbol{x}^*)$ is positive semi-definite. If, however, $\nabla f(\widehat{\boldsymbol{x}}) = \boldsymbol{0}$ and Hess $f(\widehat{\boldsymbol{x}})$ is positive definite, then $\widehat{\boldsymbol{x}}$ is a strict local minimizer.*

***Proof*** Suppose that $f$ has continuous second derivatives and $\boldsymbol{x}^*$ is a local minimizer. Then using Taylor series with second-order remainder we have for any $\boldsymbol{d}$,

$$f(\boldsymbol{x}^* + s\boldsymbol{d}) = f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}^*)(s\boldsymbol{d}) + \frac{1}{2}(s\boldsymbol{d})^T \text{Hess}\, f(\boldsymbol{x}^* + s'\boldsymbol{d})(s\boldsymbol{d})$$

for some $s'$ between zero and $s$. By Theorem 8.4, $\nabla f(\boldsymbol{x}^*) = \boldsymbol{0}$. For any $s \neq 0$ sufficiently small, $f(\boldsymbol{x}^* + s\boldsymbol{d}) \geq f(\boldsymbol{x}^*)$ as $\boldsymbol{x}^*$ is a local minimizer. Then subtracting $f(\boldsymbol{x}^*)$ and dividing by $s^2$ gives

$$0 \leq \frac{f(\boldsymbol{x}^* + s\boldsymbol{d}) - f(\boldsymbol{x}^*)}{s^2} = \frac{1}{2}\boldsymbol{d}^T \text{Hess}\, f(\boldsymbol{x}^* + s'\boldsymbol{d})\boldsymbol{d}.$$

Taking $s \to 0$ we get

$$0 \leq \frac{1}{2}\boldsymbol{d}^T \text{Hess}\, f(\boldsymbol{x}^*)\boldsymbol{d}$$

by continuity of the second derivatives. Since this is true for all $\boldsymbol{d}$, Hess $f(\boldsymbol{x}^*)$ is positive semi-definite.

To show that Hess $f(\widehat{\boldsymbol{x}})$ positive definite and $\nabla f(\widehat{\boldsymbol{x}}) = \boldsymbol{0}$ is sufficient to make $\widehat{\boldsymbol{x}}$ a strict local minimizer, we use Taylor series with second order remainder:

$$\begin{aligned} f(\boldsymbol{x}) &= f(\widehat{\boldsymbol{x}}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{x} - \widehat{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x} - \widehat{\boldsymbol{x}})^T \text{Hess}\, f(\widehat{\boldsymbol{x}} + s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}}))(\boldsymbol{x} - \widehat{\boldsymbol{x}}) \\ &= f(\widehat{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x} - \widehat{\boldsymbol{x}})^T \text{Hess}\, f(\widehat{\boldsymbol{x}} + s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}}))(\boldsymbol{x} - \widehat{\boldsymbol{x}}) \end{aligned}$$

for some $s_{\boldsymbol{x}}$ between zero and one. We just need to show now that Hess $f(\widehat{\boldsymbol{x}} + s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}}))$ is positive definite provided $\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\|$ is small enough.

We can use the Sylvester criterion (2.1.12): a real symmetric $n \times n$ matrix $A$ is positive definite if and only if the determinants

$$\det \begin{bmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kk} \end{bmatrix} > 0 \qquad \text{for } k = 1, 2, \ldots, n.$$

Since Hess $f(\widehat{\boldsymbol{x}})$ is symmetric and positive definite, this condition holds. The determinants in Sylvester's criterion are polynomials in the second derivatives of $f(\boldsymbol{x})$, and therefore continuous functions of $\boldsymbol{x}$. Thus for some $\delta > 0$, $\|z - \widehat{\boldsymbol{x}}\| < \delta$ implies Hess $f(z)$ is positive definite. Provided $\|\boldsymbol{x} - \widehat{\boldsymbol{x}}\| < \delta$, $\|s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}})\| < \delta$ as $s_{\boldsymbol{x}}$ is between zero and one, and hence Hess $f(\widehat{\boldsymbol{x}} + s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}}))$ is positive definite as we wanted. Then

$$f(\boldsymbol{x}) = f(\widehat{\boldsymbol{x}}) + \frac{1}{2}(\boldsymbol{x} - \widehat{\boldsymbol{x}})^T \text{Hess}\, f(\widehat{\boldsymbol{x}} + s_{\boldsymbol{x}}(\boldsymbol{x} - \widehat{\boldsymbol{x}}))(\boldsymbol{x} - \widehat{\boldsymbol{x}}) > f(\widehat{\boldsymbol{x}})$$

provided $0 < \|\boldsymbol{x} - \widehat{\boldsymbol{x}}\| < \delta$. Thus $\widehat{\boldsymbol{x}}$ is a local strict minimizer. □

Calculus-based methods using derivatives at a point give us useful information about how a function behaves near a point. However, it cannot tell us how the function

behaves far away. We need additional information about the function to tell if a local minimizer is a global minimizer. One class of functions for which this is easy is the class of convex functions. This is the focus of Section 8.2.

### 8.1.3 *Lagrange Multipliers and Equality-Constrained Optimization*

In equality constrained optimization, we look to minimize a function $f(\boldsymbol{x})$ subject to equality constraints $g_j(\boldsymbol{x}) = 0$ for $j = 1, 2, \ldots, m$. That is, the feasible set is

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid g_j(\boldsymbol{x}) = 0 \text{ for } j = 1, 2, \ldots, m \right\} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \right\}.$$

We will show how we can use *Lagrange multipliers* to give necessary conditions for constrained local minimizers. Lagrange multipliers were developed by Lagrange in his *Mécanique Analytique* (1788-89) [43, 151] in dealing with a mechanical system with constraints.

To be precise, $\boldsymbol{x}^*$ is a constrained local minimizer means that there is a $\delta > 0$ where

$$\boldsymbol{x}^* \in \Omega \quad \text{and} \quad \left[ \left\| \boldsymbol{x} - \boldsymbol{x}^* \right\| < \delta \text{ and } \boldsymbol{x} \in \Omega \right] \text{ implies } f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*). \tag{8.1.1}$$

We make an assumption about the functions $g_j(\boldsymbol{x})$:

$$\text{if } \boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \text{ then } \left\{ \nabla g_j(\boldsymbol{x}) \mid j = 1, 2, \ldots, m \right\} \text{ is linearly independent.} \tag{8.1.2}$$

The condition (8.1.2) is known as the *linearly independent constraint qualification* (LICQ). The LICQ condition is sufficient to ensure that the feasible set $\mathcal{A}$ is a manifold of dimension $n - m$. In general, the solution set of even a single equation $\{ \boldsymbol{x} \mid g(\boldsymbol{x}) = 0 \}$ can be extremely complex; in fact, every closed subset of $\mathbb{R}^n$ is $\{ \boldsymbol{x} \mid g(\boldsymbol{x}) = 0 \}$ for some $C^\infty$ function $g$ (this is an easy consequence of the results in [260]). The LICQ ensures that the feasible set has a suitable structure.

The Lagrange multiplier theorem we prove uses some results involving orthogonal complements (2.2.1). Recall that the orthogonal complement of a vector space $V \subseteq \mathbb{R}^n$ is

$$V^\perp = \left\{ \boldsymbol{u} \in \mathbb{R}^n \mid \boldsymbol{u}^T \boldsymbol{v} = 0 \text{ for all } \boldsymbol{v} \in V \right\}.$$

The orthogonal complement of a vector subspace is another vector subspace, and the dimension of $V^\perp$ is $n - \dim V$. We use this for a useful piece of linear algebra:

**Theorem 8.6** *If $B$ is an $m \times n$ real matrix, then $\text{range}(B) = \{ B\boldsymbol{x} \mid \boldsymbol{x} \in \mathbb{R}^n \}$ and $\text{null}(B) = \{ \boldsymbol{u} \in \mathbb{R}^n \mid B\boldsymbol{u} = \boldsymbol{0} \}$ are related through*

$$null(B) = range(B^T)^\perp \quad and, \tag{8.1.3}$$

$$range(B) = null(B^T)^\perp. \tag{8.1.4}$$

***Proof*** We show (8.1.3) first.

$$\begin{aligned}
\text{null}(B) &= \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid B\boldsymbol{x} = \boldsymbol{0} \right\} = \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{y}^T B\boldsymbol{x} = 0 \text{ for all } \boldsymbol{y} \right\} \\
&= \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid (B^T \boldsymbol{y})^T \boldsymbol{x} = 0 \text{ for all } \boldsymbol{y} \right\} \\
&= \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid \boldsymbol{z}^T \boldsymbol{x} = 0 \text{ for all } \boldsymbol{z} \in \text{range}(B^T) \right\} \\
&= \text{range}(B^T)^\perp.
\end{aligned}$$

For (8.1.4), we replace $B$ with $B^T$ so that $\text{null}(B^T) = \text{range}(B^{TT})^\perp = \text{range}(B)^\perp$. If we take the orthogonal complement of this, we get $\text{null}(B^T)^\perp = \text{range}(B)^{\perp\perp}$. To complete the proof we just need to show that $\text{range}(B)^{\perp\perp} = \text{range}(B)$.

We show that $V^{\perp\perp} = V$ for any vector space $V \subseteq \mathbb{R}^n$. First, $V \subseteq V^{\perp\perp}$ as

$$V^{\perp\perp} = \left\{ \boldsymbol{y} \mid \boldsymbol{z}^T \boldsymbol{y} = 0 \text{ for all } \boldsymbol{z} \in V^\perp \right\}.$$

If $\boldsymbol{v} \in V$, then $\boldsymbol{v}^T \boldsymbol{z} = 0$ for any $\boldsymbol{z} \in V^\perp$ by definition of $V^\perp$. On the other hand, $\dim V^{\perp\perp} = n - \dim V^\perp = n - (n - \dim V) = \dim V$. For there to be any vector $\boldsymbol{w} \in V^{\perp\perp}$ but $\boldsymbol{w} \notin V$ we would need $\dim V^{\perp\perp} > \dim V$. As this is not the case, $V = V^{\perp\perp}$ for any subspace $V$ of $\mathbb{R}^n$.

Thus, $\text{null}(B^T)^\perp = \text{range}(B)$, which shows (8.1.4). □

Given $\widehat{\boldsymbol{x}} \in \mathcal{A}$, we can apply the QR factorization (2.2.8) to $\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T = Q\,R$ with $Q$ orthogonal $n \times n$ and $R$ upper triangular $n \times m$ with $n \geq m$. Since the vectors $\nabla g_j(\widehat{\boldsymbol{x}})$ for $j = 1, 2, \ldots, m$ are linearly independent by the LICQ, the rank of $R$ is $m$. Then we can split $Q = [\, Q_1 \mid Q_2 \,]$ with $Q_1$ $n \times m$ and $Q_2$ $n \times (n - m)$. Both $Q_1$ and $Q_2$ have orthonormal columns; also $\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T = Q_1 R_1$ where $R_1$ is invertible. Then $\text{range}(\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T) = \text{range}(Q_1)$, so by Theorem 8.6 $\text{null}(\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})) = \text{range}(Q_1)^\perp = \text{range}(Q_2)$.

Consider the function $(\boldsymbol{y}, \boldsymbol{z}) \mapsto \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z})$. Now

$$\begin{aligned}
\nabla_y \left[ \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z}) \right] &= \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z})\, Q_1 \qquad \text{so} \\
\nabla_y \left[ \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z}) \right]\big|_{(\boldsymbol{y},\boldsymbol{z})=(\boldsymbol{0},\boldsymbol{0})} &= \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})\, Q_1 = (Q_1 R_1)^T Q_1 = R_1^T,
\end{aligned}$$

which is invertible. Then by the Implicit Function Theorem of multivariate calculus, there is a smooth implicit function $\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{z})$, and $\delta > 0$, where $\boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(\boldsymbol{z}) + Q_2 \boldsymbol{z}) = \boldsymbol{0}$ for all $\boldsymbol{z}$ where $\|\boldsymbol{z}\| < \delta$. Since $\nabla_z \left[ \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z}) \right] = \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{y} + Q_2 \boldsymbol{z})\, Q_2$, so at $(\boldsymbol{y}, \boldsymbol{z}) = (\boldsymbol{0}, \boldsymbol{0})$ we have the Jacobian matrix with respect to $\boldsymbol{z}$ is $\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})\, Q_2 = (Q_1 R_1)^T Q_2 = R_1^T Q_1^T Q_2 = 0$. So

$$\begin{aligned} 0 &= \nabla_z \left[ \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(z) + Q_2 z) \right] \\ &= \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(z) + Q_2 z) \, [Q_1 \nabla \boldsymbol{h}(z) + Q_2] \, ; \\ &\quad \text{and at } (\boldsymbol{y}, z) = (\boldsymbol{0}, \boldsymbol{0}) \text{ the Jacobian matrix is} \\ &= \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}}) \, [Q_1 \nabla \boldsymbol{h}(\boldsymbol{0}) + Q_2] = R_1^T \nabla \boldsymbol{h}(\boldsymbol{0}). \end{aligned}$$

As $R_1$ is invertible, $\nabla \boldsymbol{h}(\boldsymbol{0}) = 0$.

Now we can get back to optimization!

We can write the constrained $\boldsymbol{x} = \widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(z) + Q_2 z$ satisfying $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$ for any $z$ with $\|z\| < \delta$. So we do not have to consider constraints on $z$ around $z = \boldsymbol{0}$. Then we can apply the conditions for unconstrained optimization to $z \mapsto f(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(z) + Q_2 z)$. Applying Fermat's principle to this function of $z$, if $\widehat{\boldsymbol{x}}$ is a local constrained minimum,

$$\boldsymbol{0} = \nabla f(\widehat{\boldsymbol{x}})^T \, [Q_1 \nabla \boldsymbol{h}(\boldsymbol{0}) + Q_2] = \nabla f(\widehat{\boldsymbol{x}})^T \, Q_2.$$

This means that $\nabla f(\widehat{\boldsymbol{x}})$ is orthogonal to every column of $Q_2$. That is,

$$\nabla f(\widehat{\boldsymbol{x}}) \in \text{range}(Q_2)^{\perp} = \text{range}(Q_1) = \text{range}(\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T).$$

That means $\nabla f(\widehat{\boldsymbol{x}}) = \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T \boldsymbol{\lambda}$ for some $\boldsymbol{\lambda} \in \mathbb{R}^m$. That is,

$$\begin{aligned} \boldsymbol{0} &= \nabla f(\widehat{\boldsymbol{x}}) - \nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})^T \boldsymbol{\lambda} \\ &= \nabla f(\widehat{\boldsymbol{x}}) - \sum_{j=1}^{m} \lambda_j \nabla g_j(\widehat{\boldsymbol{x}}). \end{aligned} \tag{8.1.5}$$

The vector $\boldsymbol{\lambda}$ is the vector of Lagrange multipliers for $\widehat{\boldsymbol{x}}$.

The *Lagrangian function* is

$$L(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) - \sum_{j=1}^{m} \lambda_j \, g_j(\boldsymbol{x}). \tag{8.1.6}$$

The Lagrange multiplier condition (8.1.5) is that $\nabla_{\boldsymbol{x}} L(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{0}$. The constraint equations can be recovered from the condition that $\partial L / \partial \lambda_i(\boldsymbol{x}, \boldsymbol{\lambda}) = 0$ for all $i$.

**Example 8.7** To see how this works, consider the problem: $\min_{(x,y)} x$ subject to $x^2 + y^2 = 1$. Then we take $f(x, y) = x$ and $g(x, y) = x^2 + y^2 - 1$. There is only one constraint, so there is only one Lagrange multiplier $\lambda$. The Lagrangian function is $L(x, y, \lambda) = x - \lambda(x^2 + y^2 - 1)$. The Lagrange multiplier conditions for a constrained local minimizer or maximizer are

$$\begin{aligned} 0 &= 1 - 2x\,\lambda \qquad (\partial L/\partial x = 0), \\ 0 &= 0 - 2y\,\lambda \qquad (\partial L/\partial y = 0), \qquad \text{and} \\ 0 &= x^2 + y^2 - 1 \quad (\partial L/\partial \lambda = 0). \end{aligned}$$

There are exactly two possible solutions of this system of equations: $(x, y, \lambda) = (\pm 1, 0, \pm\frac{1}{2})$. The point $(x, y) = (+1, 0)$ is actually a constrained maximizer, while $(x, y) = (-1, 0)$ is a constrained minimizer.

#### 8.1.3.1 Shadow Costs and Lagrange Multipliers

Many ask what the Lagrange multipliers mean. While it can be tempting to think of them as just a mathematical device, they do have an important meaning: they are *shadow costs*. These measure the rate of change of the optimal value with respect to changes in the constraint functions. If the constraints represent resource limits, then the Lagrange multipliers represent the marginal cost (or value) of each additional unit of the resources represented. In mechanical systems, the Lagrange multipliers represent generalized forces.

To see how this works, we modify the constraints to $\boldsymbol{g}(\boldsymbol{x}) - s\boldsymbol{\gamma} = \mathbf{0}$. The optimal solution for these modified constraints is $\widehat{\boldsymbol{x}}(s)$. For the problem with modified constraints,

$$\nabla_{\boldsymbol{x}} \left[ f(\boldsymbol{x}) - \sum_{j=1}^{m} \lambda_j \left( g_j(\boldsymbol{x}) - s\gamma_j \right) \right] = \mathbf{0} \qquad \text{at } \boldsymbol{x} = \widehat{\boldsymbol{x}}(s).$$

Since $\widehat{\boldsymbol{x}}(s)$ satisfied $g_j(\widehat{\boldsymbol{x}}(s)) - \gamma_j s = 0$ for $j = 1, 2, \ldots, m$,

$$\begin{aligned} f(\widehat{\boldsymbol{x}}(s)) &= f(\widehat{\boldsymbol{x}}(s)) - \sum_{j=1}^{m} \lambda_j \left[ g_j(\widehat{\boldsymbol{x}}(s)) - \gamma_j s \right] \\ &= f(\widehat{\boldsymbol{x}}(s)) - \sum_{j=1}^{m} \lambda_j g_j(\widehat{\boldsymbol{x}}(s)) + s \sum_{j=1}^{m} \lambda_j \gamma_j \\ &= L(\widehat{\boldsymbol{x}}(s), \boldsymbol{\lambda}) + s\,\boldsymbol{\lambda}^T \boldsymbol{\gamma} \qquad \text{so} \\ \frac{d}{ds} f(\widehat{\boldsymbol{x}}(s)) &= \nabla_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}(s), \boldsymbol{\lambda})^T \frac{d\widehat{\boldsymbol{x}}}{ds} + \boldsymbol{\lambda}^T \boldsymbol{\gamma} \\ &= \boldsymbol{\lambda}^T \boldsymbol{\gamma} \qquad \text{since } \nabla_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}(s), \boldsymbol{\lambda}) = \mathbf{0}. \end{aligned}$$

This means that changing the constraints by $s\boldsymbol{\gamma}$ changes the function value at the constrained optimum $f(\widehat{\boldsymbol{x}}(s))$ by $\boldsymbol{\lambda}^T \boldsymbol{\gamma}\, s + \mathcal{O}(s^2)$ provided all functions are smooth. The value of $\lambda_i$ is the "price" per unit change in constraint $i$, which is why $\lambda_i$ is called a shadow price.

### 8.1.3.2 Second-order Conditions

We can follow the theory of unconstrained optimization to obtain second- order necessary conditions for equality constrained local minimizers. These conditions can be developed in terms of the Lagrangian function: if $\widehat{\boldsymbol{x}} + \boldsymbol{w} \in \Omega$ so that $\boldsymbol{g}(\widehat{\boldsymbol{x}} + \boldsymbol{w}) = \boldsymbol{0}$ then

$$\begin{aligned} f(\widehat{\boldsymbol{x}} + \boldsymbol{w}) &= f(\widehat{\boldsymbol{x}} + \boldsymbol{w}) - \sum_{j=1}^{m} \lambda_j g_j(\widehat{\boldsymbol{x}} + \boldsymbol{w}) \\ &= L(\widehat{\boldsymbol{x}} + \boldsymbol{w}, \boldsymbol{\lambda}) \\ &= L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) + \nabla_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda})^T \boldsymbol{w} + \frac{1}{2} \boldsymbol{w}^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) \boldsymbol{w} + \mathcal{O}(\|\boldsymbol{w}\|^3) \\ &= L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) + \frac{1}{2} \boldsymbol{w}^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) \boldsymbol{w} + \mathcal{O}(\|\boldsymbol{w}\|^3) \qquad (8.1.7) \\ &= f(\widehat{\boldsymbol{x}}) + \frac{1}{2} \boldsymbol{w}^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) \boldsymbol{w} + \mathcal{O}(\|\boldsymbol{w}\|^3). \end{aligned}$$

Assuming the LICQ, let us set $\boldsymbol{w} = Q_1 \boldsymbol{h}(z) + Q_2 z$, since $\boldsymbol{g}(\widehat{\boldsymbol{x}} + \boldsymbol{w}) = \boldsymbol{0}$. As $\nabla \boldsymbol{h}(\boldsymbol{0}) = 0$, and $\boldsymbol{h}$ is continuously differentiable, $\|\boldsymbol{h}(z)\| = \mathcal{O}(\|z\|^2)$ as $\|z\| \to 0$. This gives $\|\boldsymbol{w}\| = \mathcal{O}(\|z\|)$. Substituting this into (8.1.7) gives

$$f(\widehat{\boldsymbol{x}} + Q_1 \boldsymbol{h}(z) + Q_2 z) = f(\widehat{\boldsymbol{x}}) + \frac{1}{2} z^T Q_2^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) Q_2 z + \mathcal{O}(\|z\|^3).$$

Thus if $\widehat{\boldsymbol{x}}$ is a constrained local minimizer, $Q_2^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) Q_2$ must be positive semi-definite; further if $Q_2^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) Q_2$ is positive definite then $\widehat{\boldsymbol{x}}$ is a strict constrained local minimizer.

The matrix $Q_2^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) Q_2$ is the reduced Hessian matrix of the Lagrangian. These conditions are equivalent to

$$\begin{aligned} &\text{necessary conditions :} \quad \nabla_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) = \boldsymbol{0} \text{ and} \qquad (8.1.8) \\ &\boldsymbol{d}^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) \boldsymbol{d} \geq 0 \quad \text{for all} \quad \boldsymbol{d} \in \mathrm{null}(\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})) \\ &\text{sufficient conditions :} \quad \nabla_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) = \boldsymbol{0} \text{ and} \qquad (8.1.9) \\ &\boldsymbol{d}^T \mathrm{Hess}_{\boldsymbol{x}} L(\widehat{\boldsymbol{x}}, \boldsymbol{\lambda}) \boldsymbol{d} > 0 \quad \text{for all } \boldsymbol{0} \neq \boldsymbol{d} \in \mathrm{null}(\nabla \boldsymbol{g}(\widehat{\boldsymbol{x}})). \end{aligned}$$

**Example 8.8** If we apply this to $\min_{(x,y)} x$ subject to $x^2 + y^2 = 1$ with the Lagrangian function $L(x, y, \lambda) = x - \lambda(x^2 + y^2 - 1)$ we can determine which solution of the Lagrange multiplier conditions is a local constrained minimum. The solutions of the Lagrange multiplier conditions are $(x, y, \lambda) = (\pm 1, 0, \pm \frac{1}{2})$. First,

$$\mathrm{Hess}_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda) = \begin{bmatrix} -2\lambda & 0 \\ 0 & -2\lambda \end{bmatrix}.$$

Also, $\nabla g(x, y) = [2x,\ 2y]^T$ so $\text{null}(\nabla g(x, y)) = \left\{ [d_x, d_y]^T \mid 2x\, d_x + 2y\, d_y = 0 \right\}$. In particular, $\text{null}(\nabla g(x, y)) = \text{span}\left\{ [y,\ -x]^T \right\}$. If $\boldsymbol{d} \in \text{null}(\nabla g(x, y))$ then writing $\boldsymbol{d} = s[y,\ -x]^T$ we have

$$\boldsymbol{d}^T \text{Hess}_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda)\boldsymbol{d} = s^2(-2\lambda)(x^2 + y^2) = -2\lambda s^2.$$

The sign of this quantity is the negative of the sign of $\lambda$: taking $(x, y, \lambda) = (-1, 0, -\frac{1}{2})$ gives $\boldsymbol{d}^T \text{Hess}_{\boldsymbol{x}} L(\boldsymbol{x}, \lambda)\boldsymbol{d} = +s^2 > 0$ for $s \neq 0$, and so $(x, y) = (-1, 0)$ is a local constrained minimizer. This is actually easy to see geometrically if you draw the feasible set (a unit circle) and look for the point that minimizes $x$ on this circle. But we see here how to handle the situation with these more general tools.

**Exercises.**

(1) Show that if $g(x)$ and $h(y)$ are continuous and coercive ($\lim_{x\to\pm\infty} g(x) = \lim_{y\to\pm\infty} h(y) = +\infty$), and if $f(x, y) \geq g(x) + h(y)$, then $f(x, y)$ is also coercive. Use this and $uv \geq -\frac{1}{2}(\alpha^2 u^2 + \alpha^{-2} v^2)$ for any $\alpha \neq 0$, to show that $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ is coercive.

(2) Show that $(x^*, y^*) = (2, 1)$ is a critical point of $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$. Using this, or otherwise, find all critical points of $f$. Which are local minimizer? Which is the global minimizer?

(3) Show that $f(x, y) = x^3 + e^{-x} + xy + y^2$ is coercive. Find all critical points of $f$. [**Hint:** Reduce $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ to a problem in one variable, and use a one-variable equation solver.] Find the Hessian matrices of $f$ at each of the critical points. Identify all local minimizers and the global minimizer. What is the global minimum value?

(4) Consider the function $f : \mathbb{R}^{20} \to \mathbb{R}$ given by $f(\boldsymbol{x}) = \sum_{i=1}^{20} g(x_i)$ where $g(z) = z^2 - \cos(3z) + \sin(6z)$. Check that $g$ is coercive and has six local minimizers, and five local maximizers. Show that $f$ is coercive. How many local minimizers does $f$ have? How many critical points does $f$ have?

(5) Find $\max_x xy - |x|^p$ for $p > 1$ in terms of $y$.

(6) Compute the gradient and Hessian matrix of $f(\boldsymbol{x}) = (\boldsymbol{x}^T\boldsymbol{x})^2$.

(7) Let $f(\boldsymbol{x}) = \boldsymbol{x}^T A\boldsymbol{x}$ with $A$ symmetric. Consider the problem $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ subject to the condition that $\boldsymbol{x}^T\boldsymbol{x} = 1$. Show that the LICQ (8.1.2) holds for this problem. From the Lagrangian $L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) - \lambda\,(\boldsymbol{x}^T\boldsymbol{x} - 1)$, show that the constrained minimizer must satisfy $A\boldsymbol{x} = \lambda\boldsymbol{x}$ and $\boldsymbol{x}^T\boldsymbol{x} = 1$ where $\lambda$ is the Lagrange multiplier. Thus, $\lambda$ is an eigenvalue of $A$. Which eigenvalue gives the minimizer?

(8) Show that the LICQ (8.1.2) holds for the constraint $\boldsymbol{x}^T\boldsymbol{x} = a$ for $a > 0$, but not for $a = 0$.

(9) Consider the problem of minimizing $e^x + xy$ subject to $x^2 + y^2 = 1$. Reduce the Lagrange multiplier conditions and the constraint to a single equation in one variable. Check that the gradient of the Lagrangian is zero. Check the reduced Hessian matrix at every point you identify as a potential local minimum or maximum.

## 8.2 Convex and Non-convex

### 8.2.1 Convex Functions

A function $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ is *convex* if for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $0 \le \theta \le 1$, then

$$\varphi(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \le \theta\,\varphi(\boldsymbol{x}) + (1-\theta)\,\varphi(\boldsymbol{y}). \tag{8.2.1}$$

Convex functions are very important in optimization theory. Convex functions can be smooth or not smooth. But they cannot be discontinuous unless we allow "$+\infty$" as a value they can have. We say that $\varphi$ is *strictly convex* if for any $\boldsymbol{x} \ne \boldsymbol{y} \in \mathbb{R}^n$ and $0 < \theta < 1$,

$$\varphi(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) < \theta\,\varphi(\boldsymbol{x}) + (1-\theta)\,\varphi(\boldsymbol{y}). \tag{8.2.2}$$

If $f$ is smooth there are equivalent ways of telling if a function is convex.

**Theorem 8.9** *If $f\colon \mathbb{R}^n \to \mathbb{R}$ has continuous first derivatives, then $f$ is convex if and only if*

$$f(\boldsymbol{y}) \ge f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x}) \quad \textit{for all } \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n. \tag{8.2.3}$$

***Proof*** Suppose that $f$ is convex. Then for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and $0 \le \theta \le 1$,

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \le \theta\, f(\boldsymbol{x}) + (1-\theta)\, f(\boldsymbol{y}).$$

Put $\rho = 1 - \theta$, which is also between zero and one:

$$f((1-\rho)\boldsymbol{x} + \rho\boldsymbol{y}) \le (1-\rho)\, f(\boldsymbol{x}) + \rho\, f(\boldsymbol{y}).$$

Subtracting $f(\boldsymbol{x})$ from both sides and dividing by $\rho > 0$:

$$\frac{f(\boldsymbol{x} + \rho(\boldsymbol{y}-\boldsymbol{x})) - f(\boldsymbol{x})}{\rho} \le f(\boldsymbol{y}) - f(\boldsymbol{x}).$$

Taking the limit as $\rho \downarrow 0$ gives

$$\nabla f(\boldsymbol{x})^T(\boldsymbol{y}-\boldsymbol{x}) \le f(\boldsymbol{y}) - f(\boldsymbol{x}).$$

Re-arranging gives (8.2.3).

Now suppose that (8.2.3) holds. Put $\boldsymbol{z} = \theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}$. Then

$$f(\boldsymbol{x}) \ge f(\boldsymbol{z}) + \nabla f(\boldsymbol{z})^T(\boldsymbol{x}-\boldsymbol{z}),$$
$$f(\boldsymbol{y}) \ge f(\boldsymbol{z}) + \nabla f(\boldsymbol{z})^T(\boldsymbol{y}-\boldsymbol{z}).$$

Note that $\boldsymbol{x} - \boldsymbol{z} = (1-\theta)(\boldsymbol{x} - \boldsymbol{y})$ and $\boldsymbol{y} - \boldsymbol{z} = \theta(\boldsymbol{y} - \boldsymbol{x})$. Then

$$\begin{aligned}
&\theta\, f(\boldsymbol{x}) + (1-\theta)\, f(\boldsymbol{y}) \\
&\quad \geq \theta\, f(\boldsymbol{z}) + \theta(1-\theta)\nabla f(\boldsymbol{z})^T(\boldsymbol{x} - \boldsymbol{y}) \\
&\qquad + (1-\theta)\, f(\boldsymbol{z}) + (1-\theta)\theta\, \nabla f(\boldsymbol{z})^T(\boldsymbol{y} - \boldsymbol{x}) \\
&\quad = f(\boldsymbol{z}) = f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}).
\end{aligned}$$

That is, $f$ is convex. □

The condition (8.2.3) is a little easier than the original definition (8.2.1) in that we do not need to consider the parameter $\theta$ in (8.2.3). We can take this one step further, by looking at second derivatives.

**Theorem 8.10** *If $f : \mathbb{R}^n \to \mathbb{R}$ has continuous second derivatives, then $f$ is convex if and only if Hess $f(\boldsymbol{x})$ is positive semi-definite for all $\boldsymbol{x}$.*

***Proof*** Suppose $f$ is convex and has continuous second derivatives. By (8.2.3), setting $\boldsymbol{y} = \boldsymbol{x} + s\,\boldsymbol{d}$, with $s > 0$,

$$\begin{aligned}
f(\boldsymbol{x} + s\,\boldsymbol{d}) &\geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T s\boldsymbol{d}, \\
f(\boldsymbol{x}) &\geq f(\boldsymbol{x} + s\,\boldsymbol{d}) + \nabla f(\boldsymbol{x} + s\boldsymbol{d})^T(-s\boldsymbol{d}).
\end{aligned}$$

Adding both inequalities and subtracting $f(\boldsymbol{x} + s\,\boldsymbol{d}) + f(\boldsymbol{x})$ gives

$$0 \geq s\,[\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x} + s\boldsymbol{d})]^T\,\boldsymbol{d}.$$

Dividing by $s^2$ gives

$$0 \geq \left[\frac{\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{x} + s\boldsymbol{d})}{s}\right]^T \boldsymbol{d}.$$

Taking the limit $s \to 0$ we get

$$0 \geq -\boldsymbol{d}^T \text{Hess}\, f(\boldsymbol{x})\, \boldsymbol{d}.$$

Since this is true for all $\boldsymbol{d}$, we see that Hess $f(\boldsymbol{x})$ is positive semi-definite for any choice of $\boldsymbol{x}$.

For the converse, suppose Hess $f(\boldsymbol{z})$ is positive semi-definite for all $\boldsymbol{z}$. Then using Taylor series with second order remainder, we have

$$\begin{aligned}
f(\boldsymbol{y}) &= f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}) + \frac{1}{2}(\boldsymbol{y} - \boldsymbol{x})^T \text{Hess}\, f(\boldsymbol{x} + s(\boldsymbol{y} - \boldsymbol{x}))(\boldsymbol{y} - \boldsymbol{x}) \\
&\qquad \text{for some } 0 < s < 1 \\
&\geq f(\boldsymbol{x}) + \nabla f(\boldsymbol{x})^T(\boldsymbol{y} - \boldsymbol{x}).
\end{aligned}$$

Since this holds for any choice of $\boldsymbol{x}$ and $\boldsymbol{y}$, (8.2.3) holds, and by Theorem 8.9, $f$ is convex. □

Theorem 8.10 means that we do not need to look at pairs of points $(\boldsymbol{x}, \boldsymbol{y})$ if we can tell if the Hessian matrix of $f$ is positive semi-definite.

Beyond this, it should be noted that convex functions have a good "arithmetic": sums of convex functions are convex; the product of a convex function with a positive constant is convex; linear, and constant functions are convex. However, products of convex functions are generally not convex. Two other operations on convex functions give convex functions that are especially worth noting here: the point-wise maximum of convex functions $f(\boldsymbol{x}) = \max(g(\boldsymbol{x}), h(\boldsymbol{x}))$ are convex, and the composition of convex functions $f(\boldsymbol{x}) = g(h(\boldsymbol{x}))$ *provided* $g$ *is increasing as well as convex*. The proofs of these results are left to the Exercises.

While convex functions are not necessarily smooth, provided the value of a convex function is always finite, we can at least prove the existence of *directional derivatives*.

**Lemma 8.11** *Any convex function* $\varphi\colon \mathbb{R}^m \to \mathbb{R}$ *must have directional derivatives: for any* $\boldsymbol{x}, \boldsymbol{d} \in \mathbb{R}^m$,

$$\varphi'(\boldsymbol{x}; \boldsymbol{d}) = \lim_{h \downarrow 0} \frac{\varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x})}{h} \quad \textit{exists}. \tag{8.2.4}$$

***Proof*** We first show that $h \mapsto (\varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x}))/h$ is a non-decreasing function for $h > 0$. Suppose that $0 < h' < h$. Let $\theta = h'/h$ so that $0 \le \theta \le 1$. Then by convexity,

$$\begin{aligned} \varphi(\theta(\boldsymbol{x} + h\boldsymbol{d}) + (1-\theta)\boldsymbol{x}) &\le \theta\,\varphi(\boldsymbol{x} + h\boldsymbol{d}) + (1-\theta)\,\varphi(\boldsymbol{x}) \qquad \text{so} \\ \varphi(\boldsymbol{x} + \theta h\boldsymbol{d}) &\le \varphi(\boldsymbol{x}) + \theta\,[\varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x})]\,. \end{aligned}$$

Subtracting $\varphi(\boldsymbol{x})$ from both sides and dividing by $h' = \theta h > 0$ gives

$$\frac{\varphi(\boldsymbol{x} + \theta h\boldsymbol{d}) - \varphi(\boldsymbol{x})}{\theta h} \le \frac{\varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x})}{h},$$

as desired. Since this difference quotient is non-decreasing, either the limit exists, or the difference quotient is unbounded below giving a limit of $-\infty$. To show that the limit of $-\infty$ is impossible, we first note that convexity of $\varphi$ implies that $\varphi(\boldsymbol{x}) \le \frac{1}{2}\varphi(\boldsymbol{x} + h\boldsymbol{d}) + \frac{1}{2}\varphi(\boldsymbol{x} + h(-\boldsymbol{d}))$ giving $-\varphi(\boldsymbol{x} + h(-\boldsymbol{d})) + \varphi(\boldsymbol{x}) \le \varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x})$, and therefore

$$-\lim_{h \downarrow 0} \frac{\varphi(\boldsymbol{x} + h(-\boldsymbol{d})) - \varphi(\boldsymbol{x})}{h} \le \lim_{h \downarrow 0} \frac{\varphi(\boldsymbol{x} + h\boldsymbol{d}) - \varphi(\boldsymbol{x})}{h}.$$

If the right-hand side has the limit $-\infty$, then $\lim_{h \downarrow 0}(\varphi(\boldsymbol{x} + h(-\boldsymbol{d})) - \varphi(\boldsymbol{x}))/h = +\infty$, which is not possible as $h \mapsto (\varphi(\boldsymbol{x} + h(-\boldsymbol{d})) - \varphi(\boldsymbol{x}))/h$ is also a non-decreasing function. Thus

$$\varphi'(\boldsymbol{x};\boldsymbol{d}) = \lim_{h\downarrow 0} \frac{\varphi(\boldsymbol{x}+h\boldsymbol{d}) - \varphi(\boldsymbol{x})}{h}$$

exists and is a finite real number. □

The condition for a point $\widehat{\boldsymbol{x}}$ to be a global minimizer of a convex function $\varphi$ can be determined just from its directional derivatives.

**Theorem 8.12** *If $\varphi\colon \mathbb{R}^m \to \mathbb{R}$ is convex, then $\widehat{\boldsymbol{x}}$ minimizes $\varphi$ if and only if $\varphi'(\widehat{\boldsymbol{x}};\boldsymbol{d}) \geq 0$ for all $\boldsymbol{d}$.*

***Proof*** If $\widehat{\boldsymbol{x}}$ minimizes $\varphi$, then for any $\boldsymbol{d}$ and $h > 0$, $(\varphi(\widehat{\boldsymbol{x}} + h\boldsymbol{d}) - \varphi(\widehat{\boldsymbol{x}}))/h \geq 0$. Taking the limit as $h \downarrow 0$, we see that $\varphi'(\widehat{\boldsymbol{x}};\boldsymbol{d}) \geq 0$ for any $\boldsymbol{d}$.

Conversely, suppose that $\widehat{\boldsymbol{x}}$ does *not* minimize $\varphi$; then there must be $\boldsymbol{y}$ where $\varphi(\boldsymbol{y}) < \varphi(\widehat{\boldsymbol{x}})$. Put $\boldsymbol{d} = \boldsymbol{y} - \widehat{\boldsymbol{x}}$. Since $h \mapsto (\varphi(\widehat{\boldsymbol{x}} + h\boldsymbol{d}) - \varphi(\widehat{\boldsymbol{x}}))/h$ is a non-decreasing function, if $0 < h < 1$,

$$\frac{\varphi(\widehat{\boldsymbol{x}} + h\boldsymbol{d}) - \varphi(\widehat{\boldsymbol{x}})}{h} \leq \frac{\varphi(\widehat{\boldsymbol{x}} + 1\,\boldsymbol{d}) - \varphi(\widehat{\boldsymbol{x}})}{1} = \varphi(\boldsymbol{y}) - \varphi(\widehat{\boldsymbol{x}}) < 0.$$

Taking the limit as $h \downarrow 0$ gives $\varphi'(\widehat{\boldsymbol{x}};\boldsymbol{d}) < 0$ for $\boldsymbol{d} = \boldsymbol{y} - \widehat{\boldsymbol{x}}$.

Thus $\widehat{\boldsymbol{x}}$ minimizes $\varphi$ if and only if $\varphi'(\widehat{\boldsymbol{x}};\boldsymbol{d}) \geq 0$ for all $\boldsymbol{d}$. □

If $f$ is differentiable, then the directional derivative $f'(\boldsymbol{x};\boldsymbol{d}) = \nabla f(\boldsymbol{x})^T\boldsymbol{d}$. The advantage of working with directional derivatives is that directional derivatives always exist for convex functions with finite values. In fact, this result gives us a partial non-smooth version of Theorem 8.9: for convex $f\colon \mathbb{R}^n \to \mathbb{R}$,

$$f(\boldsymbol{y}) \geq f(\boldsymbol{x}) + f'(\boldsymbol{x};\ \boldsymbol{y}-\boldsymbol{x}) \qquad \text{for all } \boldsymbol{x},\ \boldsymbol{y}.$$

Here, we give a quick example of how we can use this to determine optimality for convex but nonsmooth functions. Take $\varphi(x) = |x| + g(x)$ where $g$ is smooth and convex. Since $|x|$ is a convex function of $x$, this $\varphi$ is convex. The directional derivative $\varphi'(x;d) = \big[\text{sign}(x) + g'(x)\big]\,d$ for $x \neq 0$ and $\varphi'(0;d) = |d| + g'(0)\,d$. Zero is a global minimizer if $\big|g'(0)\big| \leq 1$ as then $\varphi'(0;d) \geq 0$ for $d > 0$ and $\varphi'(0;d) \geq 0$ for $d < 0$, so the conditions of Theorem 8.12 are satisfied.

### 8.2.2 Convex Sets

A set $C$ in a real vector space is a *convex set* if

(8.2.5) $\qquad \boldsymbol{x}, \boldsymbol{y} \in C$ and $0 \leq \theta \leq 1$ implies $\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y} \in C$.

If $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ is a convex function and $L \in \mathbb{R}$, then the level sets $\{\,\boldsymbol{x} \in \mathbb{R}^n \mid \varphi(\boldsymbol{x}) \leq L\,\}$ are convex. Since norms are convex functions, the unit ball

$\{ \boldsymbol{x} \in V \mid \|\boldsymbol{x}\|_V < 1 \}$ is also a convex set for each vector space with a norm $\|\cdot\|_V$. If we use a strict inequality "<", then we have an open ball; if we use a non-strict inequality we get $\{ \boldsymbol{x} \in V \mid \|\boldsymbol{x}\|_V \leq 1 \}$, which is a closed ball.

If $C_1$ and $C_2$ are convex sets, then $C_1 \cap C_2$ is either empty or a convex set, but $C_1 \cup C_2$ usually is not convex. Every real vector space is a convex set. If $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ is convex, then the set

$$\operatorname{epi} \varphi = \left\{ (\boldsymbol{x}, y) \in \mathbb{R}^n \times \mathbb{R} \mid y \geq \varphi(\boldsymbol{x}) \right\}, \tag{8.2.6}$$

called the *epigraph* of $\varphi$, is convex.

Given a set $S \subset \mathbb{R}^n$, the smallest convex set $C$ containing $S$ is called the *convex hull* of $S$ and denoted co $S$. To see why we can say "the" convex hull, suppose that $C_1 \neq C_2$ are two different candidates for the convex hull of $S$. Then $C := C_1 \cap C_2$ also contains $S$ and is convex, but $C \subseteq C_1$ and $C_2$. This contradicts the claim that both $C_1$ and $C_2$ are convex hulls of $S$. The convex hull of three points not all on a common line is a triangle; the convex hull of four points not all on a common plane is a tetrahedron.

Many results on convex sets can be obtained through a single theorem:

**Theorem 8.13** (Separating Hyperplane Theorem) *If $C$ is a non-empty closed convex set and $\boldsymbol{y} \notin C$ all in $\mathbb{R}^n$, then there is a vector $\boldsymbol{n} \in \mathbb{R}^n$ and $\beta \in \mathbb{R}$ where*

$$\boldsymbol{n}^T \boldsymbol{x} \leq \beta \quad \text{for all } \boldsymbol{x} \in C, \text{ and} \tag{8.2.7}$$

$$\boldsymbol{n}^T \boldsymbol{y} > \beta. \tag{8.2.8}$$

This theorem can be generalized to any Banach space, so that it can be applied to questions about convex sets in infinite dimensional spaces.

***Proof*** Let $\varphi\colon \mathbb{R}^n \to \mathbb{R}$ be the function $\varphi(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{y}\|_2^2$. Note that $\varphi$ is continuous and coercive, so by Theorem 8.2 there must be a minimizer $\boldsymbol{x}^* \in C$ of $\varphi$. If $\boldsymbol{x} \in C$, then $\theta\boldsymbol{x} + (1-\theta)\boldsymbol{x}^* = \boldsymbol{x}^* + \theta(\boldsymbol{x} - \boldsymbol{x}^*) \in C$ for any $0 \leq \theta \leq 1$. For $0 < \theta \leq 1$,

$$\frac{\varphi(\boldsymbol{x}^* + \theta(\boldsymbol{x} - \boldsymbol{x}^*)) - \varphi(\boldsymbol{x}^*)}{\theta} \geq 0.$$

Taking the limit $\theta \downarrow 0$ we obtain

$$\nabla\varphi(\boldsymbol{x}^*)^T(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0.$$

That is,

$$(\boldsymbol{x}^* - \boldsymbol{y})^T(\boldsymbol{x} - \boldsymbol{x}^*) \geq 0 \qquad \text{for any } \boldsymbol{x} \in C.$$

Setting $\boldsymbol{n} = \boldsymbol{y} - \boldsymbol{x}^*$ and $\beta = \boldsymbol{n}^T\boldsymbol{x}^*$ we see that for $\boldsymbol{x} \in C$, $\boldsymbol{n}^T\boldsymbol{x} \leq \boldsymbol{n}^T\boldsymbol{x}^*$ so

$$\boldsymbol{n}^T\boldsymbol{x} \le \boldsymbol{n}^T\boldsymbol{x}^* = \beta, \qquad \text{while}$$
$$\boldsymbol{n}^T\boldsymbol{y} = (\boldsymbol{y}-\boldsymbol{x}^*)^T\boldsymbol{y} = (\boldsymbol{y}-\boldsymbol{x}^*)^T(\boldsymbol{y}-\boldsymbol{x}^*) + \beta > \beta,$$

with a strict inequality since $C \not\ni \boldsymbol{y} \neq \boldsymbol{x}^* \in C$. □

Many important results about convex functions can be derived from the Separating Hyperplane Theorem. This theorem is essential for establishing the Karush–Kuhn–Tucker conditions.

**Exercises.**

(1) Show that if $f$ and $g$ are convex functions $\mathbb{R}^n \to \mathbb{R}$ and $\alpha > 0$ is a real number, then $f + g$ and $\alpha f$ are also convex.
(2) Show that if $f$ and $g$ are convex functions $\mathbb{R}^n \to \mathbb{R}$ then the function $h(\boldsymbol{x}) = \max(f(\boldsymbol{x}),\ g(\boldsymbol{x}))$ is also convex.
(3) Show that if $f\colon \mathbb{R} \to \mathbb{R}$ and $g\colon \mathbb{R}^n \to \mathbb{R}$ are convex functions and $f$ is also a non-decreasing function ($u \ge v$ implies $f(u) \ge f(v)$) then $h(\boldsymbol{x}) = f(g(\boldsymbol{x}))$ is also convex.
(4) Show that the non-decreasing condition on $f$ in the previous Exercise is necessary, by means of the counter-example $f(u) = \exp(-u)$ and $g(x) = x^2$.
(5) Show that the function $f(\boldsymbol{x}) = \sqrt{1 + \boldsymbol{x}^T\boldsymbol{x}}$ is convex.
(6) Show that the functions $g(u) = -\ln u$ and $h(u) = u\ \ln u$ are both convex functions of $u$ for $u > 0$.
(7) Show that linear functions $f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} + b$ are convex.
(8) An *extreme point* of a convex set $C$ is a point $\boldsymbol{x} \in C$ such that there are no points $\boldsymbol{y},\ \boldsymbol{z} \in C$ where $\boldsymbol{y} \neq \boldsymbol{z}$ and $\boldsymbol{x} = \frac{1}{2}(\boldsymbol{y} + \boldsymbol{z})$. Show that any convex function $f$ that has a *maximizer* over a convex set $C \subset \mathbb{R}^n$, has a maximizer that is an extreme point of $C$. Show that if $f$ is strictly convex then *every maximizer* over $C$ is an extreme point of $C$.
(9) Show that the function $f(\boldsymbol{x}) = -\ln(\boldsymbol{a}^T\boldsymbol{x} + b) + \sum_{j=1}^m r_j \exp(\boldsymbol{c}_j^T\boldsymbol{x} + d_j) + \sqrt{1 + \boldsymbol{x}^T B\boldsymbol{x}}$ is convex provided each $r_j \ge 0$ and $B$ is positive definite.
(10) Show that the set of symmetric positive-definite matrices is a convex set. Also show that the function $f(A) = -\ln(\det A)$ is convex over the set of symmetric positive-definite matrices. [**Hint:** Compute $(d^2/ds^2)\,[-\ln(\det(A + sE))]$ using the fact that $(d/ds)\det(A + sE) = \text{trace}((A + sE)^{-1}E)\ \det(A + sE)$, that symmetric positive-definite matrices have symmetric positive-definite square roots, and that the trace of a positive-definite matrix is positive.]

## 8.3 Gradient Descent and Variants

For $s > 0$ and small, $f(\boldsymbol{x} + s\boldsymbol{d}) \approx f(\boldsymbol{x}) + s\,\boldsymbol{d}^T\nabla f(\boldsymbol{x})$. If $\boldsymbol{d}^T\nabla f(\boldsymbol{x}) < 0$ we say $\boldsymbol{d}$ is a *descent direction* of $f$ at $\boldsymbol{x}$. If we minimize $\boldsymbol{d}^T\nabla f(\boldsymbol{x})$ over $\|\boldsymbol{d}\|_2 = c$ then we choose $\boldsymbol{d} = -(c/\,\|\nabla f(\boldsymbol{x})\|_2)\,\nabla f(\boldsymbol{x})$. Scaling this vector, we can use $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$

as the "most efficient" direction in which to reduce the objective function. Stepping in the negative gradient direction is the basis of the gradient descent method. Gradient descent is the basis, or a fall-back, for many other algorithms.

Computing gradients are often considered a drawback for gradient and other derivative-based optimization methods. This is because computing gradients are considered either expensive or inconvenient or both. But computing gradients need not be either excessively expensive or inconvenient if automatic or computational differentiation is used (see Section 5.5.2).

### 8.3.1 Gradient Descent

The simplest version of gradient descent is to simply step a small, but fixed amount, in the negative gradient direction. This is shown in Algorithm 72.

To analyze gradient descent algorithms, we make the assumption that $\nabla f$ is a Lipschitz function with Lipschitz constant $L$:

$$\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L \|\boldsymbol{x} - \boldsymbol{y}\|_2 \qquad \text{for all } \boldsymbol{x},\ \boldsymbol{y}. \tag{8.3.1}$$

For $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$,

$$\begin{aligned} f(\boldsymbol{x} + s\boldsymbol{d}) &= f(\boldsymbol{x}) + s\boldsymbol{d}^T \nabla f(\boldsymbol{x}) + \int_0^s \boldsymbol{d}^T \left[\nabla f(\boldsymbol{x} + t\boldsymbol{d}) - \nabla f(\boldsymbol{x})\right] dt, \qquad \text{so} \\ &\leq f(\boldsymbol{x}) - s \|\nabla f(\boldsymbol{x})\|_2^2 + \int_0^s \|\boldsymbol{d}\|_2 \|\nabla f(\boldsymbol{x} + t\boldsymbol{d}) - \nabla f(\boldsymbol{x})\|_2 \, dt \\ &\qquad \text{(by the Cauchy-Schwartz inequality)} \\ &\leq f(\boldsymbol{x}) - s \|\nabla f(\boldsymbol{x})\|_2^2 + \int_0^s \|\nabla f(\boldsymbol{x})\|_2 \ t\, L \ \|\boldsymbol{d}\|_2 \ dt \\ &= f(\boldsymbol{x}) - s \|\nabla f(\boldsymbol{x})\|_2^2 + \frac{1}{2} s^2 L \|\nabla f(\boldsymbol{x})\|_2^2 \\ &= f(\boldsymbol{x}) - s \|\nabla f(\boldsymbol{x})\|_2^2 \left[1 - \frac{1}{2} sL\right]. \end{aligned} \tag{8.3.2}$$

**Algorithm 72** Simple gradient descent

```
1  function simplegraddescent(f, ∇f, x_0, s, ϵ)
2    k ← 0
3    while ‖∇f(x_k)‖ > ϵ
4      x_{k+1} ← x_k − s ∇f(x_k)
5      k ← k + 1
6    end while
7    return x_k
8  end function
```

In order to ensure a decrease in the function values so $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ provided $\nabla f(\boldsymbol{x}_k) \neq \mathbf{0}$, we should have $0 < s < 2/L$. As we often do not have good estimates for $L$, the Lipschitz constant for $\nabla f$, we usually make $s > 0$ small. But the smaller we make $s$, the more iterations are needed to achieve a target reduction in the function values. Larger $s > 0$ should mean larger steps, and hopefully, fewer steps to come close to a minimizer.

We should also note that we do not guarantee that the $\boldsymbol{x}_k$ converge to a global minimizer. At best, we can only expect to approach a local minimizer. Even this is not always true. For $0 < s < 2/L$, all we can guarantee is that either $f(\boldsymbol{x}_k) \to -\infty$ or $\nabla f(\boldsymbol{x}_k) \to 0$ as $k \to \infty$. To see why, we start with

$$f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - s \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2 \left[1 - \frac{1}{2} sL\right]; \qquad \text{that is}$$

$$f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) \leq -s \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2 \left[1 - \frac{1}{2} sL\right].$$

Summing both sides from $k = 0$ to $k = M$ gives

$$f(\boldsymbol{x}_{M+1}) - f(\boldsymbol{x}_0) \leq -s \left[1 - \frac{1}{2} sL\right] \sum_{k=0}^{M} \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2 .$$

Flipping the signs, and the inequality, gives

$$f(\boldsymbol{x}_0) - f(\boldsymbol{x}_{M+1}) \geq s \left[1 - \frac{1}{2} sL\right] \sum_{k=0}^{M} \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2 ,$$

so taking $M \to \infty$ gives

$$f(\boldsymbol{x}_0) - \inf_k f(\boldsymbol{x}_k) \geq s \left[1 - \frac{1}{2} sL\right] \sum_{k=0}^{\infty} \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2 .$$

Either the left side is infinite ($f(\boldsymbol{x}_k) \to -\infty$) or $\sum_{k=0}^{\infty} \left\|\nabla f(\boldsymbol{x}_k)\right\|_2^2$ is finite. Thus if $f$ is bounded below, then $\nabla f(\boldsymbol{x}_k) \to \mathbf{0}$ as $k \to \infty$.

To see how quickly the method converges, it helps to see how it works for a simple problem. Consider, for example, the quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$ with $A$ symmetric and positive definite. A step of the simple gradient descent method gives

$$\begin{aligned} \boldsymbol{x}_{k+1} - A^{-1}\boldsymbol{b} &= \boldsymbol{x}_k - s(A\boldsymbol{x}_k + \boldsymbol{b}) - A^{-1}\boldsymbol{b} \\ &= (I - sA)(\boldsymbol{x}_k - A^{-1}\boldsymbol{b}). \end{aligned}$$

The rate of convergence of this fixed point iteration is the spectral radius (2.4.5) $\rho(I - sA) = \max(|1 - s\lambda_{\min}|\,,\ |1 - s\lambda_{\max}|)$ where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum

and maximum eigenvalues of $A$ respectively. So, in this case, the optimal $s = s^* = 2/(\lambda_{\min} + \lambda_{\max})$ and the optimal linear convergence rate is given by

$$\begin{aligned}\rho(I - s^*A) &= \frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} = \frac{1 - (\lambda_{\min}/\lambda_{\max})}{1 + (\lambda_{\min}/\lambda_{\max})} \\ &\approx 1 - 2\frac{\lambda_{\min}}{\lambda_{\max}} = 1 - \frac{2}{\kappa_2(A)}\end{aligned}$$

where $\kappa_2(A) = \left\|A^{-1}\right\|_2 \|A\|_2 = \lambda_{\max}/\lambda_{\min}$ is the 2-norm condition number of $A$. Note that $\lambda_{\min} > 0$ as $A$ is assumed positive definite.

More generally, for convex functions we have an asymptotically slower bound:

**Theorem 8.14** *Suppose $f : \mathbb{R}^n \to \mathbb{R}$ is convex and has continuous first derivatives where $\nabla f$ is Lipschitz with constant $L$, and the minimizer of $f$ is $\boldsymbol{x}^*$. Then provided $0 < sL \leq 1$ in Algorithm 72,*

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}^*) \leq \frac{\|\boldsymbol{x}_0 - \boldsymbol{x}^*\|_2^2}{2\,s\,k}. \tag{8.3.3}$$

***Proof*** From (8.3.2), $f(\boldsymbol{x}_{k+1}) \leq f(\boldsymbol{x}_k) - \frac{1}{2}s\,\|\nabla f(\boldsymbol{x}_k)\|_2^2$ as $0 < sL \leq 1$. Since $f$ is convex,

$$\begin{aligned}f(\boldsymbol{x}^*) &\geq f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}^* - \boldsymbol{x}_k), \qquad \text{so} \\ f(\boldsymbol{x}_k) &\leq f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}_k - \boldsymbol{x}^*).\end{aligned}$$

Thus

$$\begin{aligned}f(\boldsymbol{x}_{k+1}) &\leq f(\boldsymbol{x}_k) - \frac{1}{2}s\,\|\nabla f(\boldsymbol{x}_k)\|_2^2 \\ &\leq f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}_k - \boldsymbol{x}^*) - \frac{1}{2}s\,\|\nabla f(\boldsymbol{x}_k)\|_2^2.\end{aligned}$$

Then

$$\begin{aligned}f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^*) &\leq \frac{1}{2s}\left[2s\nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}_k - \boldsymbol{x}^*) - s^2\,\|\nabla f(\boldsymbol{x}_k)\|_2^2\right] \\ &= \frac{1}{2s}\left[\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2 - \|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2 + 2s\nabla f(\boldsymbol{x}_k)^T(\boldsymbol{x}_k - \boldsymbol{x}^*) - s^2\,\|\nabla f(\boldsymbol{x}_k)\|_2^2\right] \\ &= \frac{1}{2s}\left[\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2 - \|\boldsymbol{x}_k - \boldsymbol{x}^* - s\nabla f(\boldsymbol{x}_k)\|_2^2\right] \\ &= \frac{1}{2s}\left[\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2^2 - \|\boldsymbol{x}_{k+1} - \boldsymbol{x}^*\|_2^2\right].\end{aligned}$$

Summing over $k$ gives

**Algorithm 73** Gradient descent with line search

```
1   function graddescent_ls(f, ∇f, x_0, linesearch, params, ϵ)
2     k ← 0
3     while ‖∇f(x_k)‖ > ϵ
4       d_k ← −∇f(x_k)
5       s_k ← linesearch(f, ∇f, x_k, d_k, params, s_{k−1})
6       x_{k+1} ← x_k + s_k d_k
7       k ← k + 1
8     end while
9     return x_k
10  end function
```

$$\sum_{k=0}^{p-1} \left(f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}^*)\right) \leq \frac{1}{2s}\left[\left\|\boldsymbol{x}_0 - \boldsymbol{x}^*\right\|_2^2 - \left\|\boldsymbol{x}_p - \boldsymbol{x}^*\right\|_2^2\right]$$
$$\leq \frac{1}{2s}\left\|\boldsymbol{x}_0 - \boldsymbol{x}^*\right\|_2^2.$$

Since $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ for all $k$,

$$p\left(f(\boldsymbol{x}_p) - f(\boldsymbol{x}^*)\right) \leq \frac{1}{2s}\left\|\boldsymbol{x}_0 - \boldsymbol{x}^*\right\|_2^2.$$

Dividing by $p$ gives the result. □

The value of Theorem 8.14 is that the bounds are not asymptotic and do not depend on the condition number $\kappa_2(\text{Hess } f(\boldsymbol{x}^*))$. But whatever method of analysis is used, the step length $s$ must be "small enough" for the method to work. Simply fixing its value does not guarantee it. We need a more adaptive method. We need line searches.

### *8.3.2 Line Searches*

Knowing the negative gradient direction does not give any indication of how far in that direction the algorithm should step. Line searches are needed in all but the most basic gradient descent algorithms. Line search algorithms make gradient descent algorithms much more robust. The basic idea of a line search algorithm for gradient descent is shown in Algorithm 73.

The ideal line search computes the minimum of $f(\boldsymbol{x} + s\boldsymbol{d})$ over $s \geq 0$. We want to ensure that $s = 0$ is not the minimizer as this would mean that no progress would be made. Usually, this is prevented by ensuring that $\boldsymbol{d}$ is a descent direction for $f$ at $\boldsymbol{x}$. Setting $\boldsymbol{d} = -\nabla f(\boldsymbol{x})$ ensures that $\boldsymbol{d}$ is a descent direction.

**Algorithm 74** Armijo/backtracking line search

```
1   function armijo(f, ∇f, s_0, x, d, c_1)
2     g ← ∇f(x);  s ← s_0
3     while f(x + sd) > f(x) + c_1 s g^T d
4       s ← (1/2) s
5     end while
6     return s
7   end function
```

#### 8.3.2.1 Armijo/Backtracking Line Search

The simplest widely used line search algorithm is the *Armijo/backtracking* algorithm shown as Algorithm 74 [8]. For the method to terminate, we need $s_0 > 0$ and $0 < c_1 < 1$. The result of Algorithm 74 is $s$ where

$$s > 0 \quad \text{and} \quad f(\boldsymbol{x} + s\boldsymbol{d}) \leq f(\boldsymbol{x}) + c_1 s\, \boldsymbol{d}^T \nabla f(\boldsymbol{x}). \tag{8.3.4}$$

Condition (8.3.4) is called the *sufficient decrease condition*.

The Armijo/backtracking algorithm terminates in finite time (assuming no round-off error) because an infinite loop would imply that

$$\lim_{k \to \infty} \frac{f(\boldsymbol{x} + s_0 2^{-k} \boldsymbol{d}) - f(\boldsymbol{x})}{s_0 2^{-k}} = \nabla f(\boldsymbol{x})^T \boldsymbol{d} \geq c_1 \nabla f(\boldsymbol{x})^T \boldsymbol{d}$$

and thus $\nabla f(\boldsymbol{x})^T \boldsymbol{d} \geq 0$ contradicting the assumption that $\boldsymbol{d}$ is a descent direction.

#### 8.3.2.2 Goldstein Condition-based Line Search

A related method is the Goldstein line search algorithm [103]. This method is based on satisfying two conditions, the first of which is the sufficient decrease criterion:

$$f(\boldsymbol{x} + s\boldsymbol{d}) \leq f(\boldsymbol{x}) + \quad c_1\ s\, \boldsymbol{d}^T \nabla f(\boldsymbol{x}), \tag{8.3.5}$$

$$f(\boldsymbol{x} + s\boldsymbol{d}) \geq f(\boldsymbol{x}) + (1 - c_1) s\, \boldsymbol{d}^T \nabla f(\boldsymbol{x}). \tag{8.3.6}$$

In order to satisfy both conditions, it is necessary that $0 < c_1 < 1/2$. An algorithm to implement the Goldstein search is shown in Algorithm 75.

The Goldstein line search algorithm terminates provided $f$ is continuously differentiable, $\boldsymbol{d}$ is a descent direction, and $f$ is bounded below, since

- the loop on lines 6–8 terminates for the same reason that the Armijo/backtracking algorithm terminates;
- the loop on lines 9–11 terminates since otherwise $s_{hi} \to \infty$ and $f(\boldsymbol{x} + s_{hi}\boldsymbol{d}) < f(\boldsymbol{x}) + (1 - c_1) s_{hi} \nabla f(\boldsymbol{x})^T \boldsymbol{d} \to -\infty$ violating the assumption that $f$ is bounded below;

**Algorithm 75** Goldstein line search

```
1   function goldstein(f, ∇f, x, d, s, c1)
2     v ← ∇f(x)^T d  // slope of f(x + sd) at s = 0
2     GC1 ← [f(x + sd) ≤ f(x) + c1 s v]
3     GC2 ← [f(x + sd) ≥ f(x) + (1 − c1)s v]
4     if GC1 and GC2: return s; end if
5     s_lo ← s; s_hi ← s
6     while not GC1
7       s_lo ← (1/2) s_lo; GC1 ← [f(x + s_lo d) ≤ f(x) + c1 s_lo v]
8     end while
9     while not GC2
10      s_hi ← 2 s_hi; GC2 ← [f(x + s_hi d) ≥ f(x) + (1 − c1)s_hi v]
11    end while
12    while true
13      s ← (s_lo + s_hi)/2
14      GC1 ← [f(x + sd) ≤ f(x) + c1 s v]
15      GC2 ← [f(x + sd) ≥ f(x) + (1 − c1)s v]
16      if GC1
17        if GC2
18          return s
19        else
20          s_lo ← s
21        end if
22        s_hi ← s
23      end if
24    end while
25  end function
```

- the loop on lines 12–24 maintains the properties that (8.3.5) holds at $s = s_{lo}$ while (8.3.6) holds at $s = s_{hi}$. If the loop on lines 12–24 were infinite, then $|s_{hi} - s_{lo}| \to 0$ as $k \to \infty$ leading to a point $s = s^*$ being the common limit of $s_{lo}$ and $s_{hi}$. Both (8.3.5) and (8.3.6) hold at $s = s^*$. Since $0 < c_1 < 1/2$, both (8.3.5) and (8.3.6) hold for $s$ near $s^*$, so that the algorithm terminates once $s_{lo}$ and $s_{hi}$ are sufficiently close to $s^*$. Thus the assumption of an infinite loop is false.

It should be noted that the choice of $s \leftarrow \frac{1}{2}(s_{lo} + s_{hi})$ in line 13 is not the only choice. It should be noted that hybrid methods discussed in Section 3.4.3 can be used to identify other formulations that are faster in general. Unlike the methods in Section 3.4.3, we can terminate the Goldstein line search algorithm as soon as Goldstein conditions hold, rather than attempting to solve a nonlinear equation.

#### 8.3.2.3 Wolfe Condition-based Line Search

The *Wolfe conditions* [262, 263] impose conditions on $\nabla f(\boldsymbol{x} + s\boldsymbol{d})$ as well as on the value $f(\boldsymbol{x} + s\boldsymbol{d})$. The strong Wolfe conditions are the sufficient decrease criterion (8.3.4)

$$f(\boldsymbol{x}+s\boldsymbol{d}) \le f(\boldsymbol{x}) + c_1 s\, \boldsymbol{d}^T \nabla f(\boldsymbol{x}) \tag{8.3.7}$$

and the "curvature" condition

$$\left|\boldsymbol{d}^T \nabla f(\boldsymbol{x}+s\boldsymbol{d})\right| \le c_2 \left|\boldsymbol{d}^T \nabla f(\boldsymbol{x})\right|. \tag{8.3.8}$$

Note that to guarantee that both of the Wolfe conditions can be satisfied, we have to assume that $f$ is continuously differentiable, $f$ is bounded below, and $0 < c_1 < c_2 < 1$. There are also the weak Wolfe conditions, where the curvature condition (8.3.8) is replaced by

$$\boldsymbol{d}^T \nabla f(\boldsymbol{x}+s\boldsymbol{d}) \ge c_2 \boldsymbol{d}^T \nabla f(\boldsymbol{x}). \tag{8.3.9}$$

**Lemma 8.15** (Wolfe conditions) *If $f : \mathbb{R}^n \to \mathbb{R}$ is bounded below and continuously differentiable with $\boldsymbol{d}^T \nabla f(\boldsymbol{x}) < 0$, then there is a solution $s > 0$ of (8.3.7, 8.3.8) provided $0 < c_1 < c_2 < 1$.*

***Proof*** Suppose that $f(\boldsymbol{z}) \ge f_{\inf}$ for all $\boldsymbol{z} \in \mathbb{R}^n$. Then (8.3.7) implies that there is a finite

$$s_{hi} = \inf \left\{ s > 0 \mid f(\boldsymbol{x}+s\boldsymbol{d}) \ge f(\boldsymbol{x}) + c_1 s \boldsymbol{d}^T \nabla f(\boldsymbol{x}) \right\}.$$

The set defining $s_{hi}$ is non-empty because $f$ is differentiable, bounded below, and $\boldsymbol{d}^T \nabla f(\boldsymbol{x}) < 0$. Also $s_{hi} > 0$ as for sufficiently small $s > 0$ we have $f(\boldsymbol{x}+s\boldsymbol{d}) < f(\boldsymbol{x}) + c_1 s \boldsymbol{d}^T \nabla f(\boldsymbol{x})$. Note that $f(\boldsymbol{x}+s_{hi}\boldsymbol{d}) = f(\boldsymbol{x}) + c_1 s_{hi} \boldsymbol{d}^T \nabla f(\boldsymbol{x})$. Note that if $0 \le s \le s_{hi}$, the condition $f(\boldsymbol{x}+s\boldsymbol{d}) \le f(\boldsymbol{x}) + c_1 s \boldsymbol{d}^T \nabla f(\boldsymbol{x})$ holds.

By the Mean Value Theorem, there must be a point $\widehat{s}$ strictly between zero and $s_{hi}$ where

$$0 > c_1 s_{hi} \boldsymbol{d}^T \nabla f(\boldsymbol{x}) = f(\boldsymbol{x}+s_{hi}\boldsymbol{d}) - f(\boldsymbol{x}) = s_{hi} \boldsymbol{d}^T \nabla f(\boldsymbol{x}+\widehat{s}\boldsymbol{d}).$$

Dividing by $s_{hi}$ gives $\boldsymbol{d}^T \nabla f(\boldsymbol{x}+\widehat{s}\boldsymbol{d}) = c_1 \boldsymbol{d}^T \nabla f(\boldsymbol{x})$ and so

$$\left|\boldsymbol{d}^T \nabla f(\boldsymbol{x}+\widehat{s}\boldsymbol{d})\right| = c_1 \left|\boldsymbol{d}^T \nabla f(\boldsymbol{x})\right| < c_2 \left|\boldsymbol{d}^T \nabla f(\boldsymbol{x})\right|.$$

Furthermore, there is an interval of values of $s$ where both (8.3.7) and (8.3.8) hold. □

An algorithm to find a solution of the strong Wolfe conditions (8.3.7, 8.3.8) is given in two parts as Algorithms 76 and 77 [190, pp. 60-61]. Algorithm 77 shows the inner "zoom" function, while Algorithm 76 shows the outer function that calls *zoom*(). These algorithms are written in terms of the function $\phi(s) = f(\boldsymbol{x}+s\boldsymbol{d})$ and $\phi'(s) = \boldsymbol{d}^T \nabla f(\boldsymbol{x}+s\boldsymbol{d})$.

There are two competing objectives in selecting the step length $s > 0$: we want the step to give a reduction in the objective function which can be achieved by choosing a small value of $s$. On the other hand, making a "safe" choice of $s$ by making it small, means that more steps will be needed to obtain the same reduction in the objective

**Algorithm 76** Wolfe condition based line search — outer function

```
1   function wolfe(φ, φ′, s_0, s_max)
2     choose s_1 ∈ (0, s_max)
3     k ← 1
4     while true
5       if φ(s_k) > φ(0) + c_1 s_k φ′(0) or [φ(s_k) ≥ φ(s_{k−1}) and k > 1]
6         return zoom(φ, φ′, s_{k−1}, s_k)
7       else if |φ′(s_k)| ≤ c_2 |φ′(0)|
8         return s_k
9       else if φ′(s_k) ≥ 0
10        return zoom(φ, φ′, s_k, s_{k−1})
11      end if
12      choose s_{k+1} ∈ (s_k, s_max)
13      k ← k + 1
14    end while
15  end function
```

**Algorithm 77** Wolfe condition based line search — zoom

```
1   function zoom(φ, φ′, s_lo, s_hi)
2     while true
3       obtain new estimate s between s_lo and s_hi
4       if φ(s) > φ(0) + c_1 s φ′(0) or φ(s) ≥ φ(s_lo)
5         s_hi ← s
6       else
7         if |φ′(s)| ≤ c_2 |φ′(0)|
8           return s
9         else if φ′(s)(s_hi − s_lo) ≥ 0
10          s_hi ← s_lo
11        end if
12        s_lo ← s
13      end if
14    end while
15  end function
```

function value. The Wolfe conditions (8.3.7, 8.3.8) ensure both that the step length $s > 0$ is both "not too large" and "not too small".

#### 8.3.2.4 Choice of Line Search Parameters

The line search parameters are $c_1$ for the Armijo/backtracking method and the Goldstein condition-based search, and both $c_1$ and $c_2$ for the Wolfe condition-based line search. There are some essential conditions that must be satisfied by these parameters:

- Armijo/backtracking: $0 < c_1 < 1$.
- Goldstein: $0 < c_1 < 1/2$.
- Wolfe: $0 < c_1 < c_2 < 1$.

The smaller $c_2$ is, the tighter the line search is. The exact or ideal line search method can be approximated by a Wolfe condition based line search with $c_2$ small.

However, looser line searches are preferred in practice. Looser line search conditions mean that fewer function evaluations are needed between updating the line search direction $\boldsymbol{d}_k$. For Newton and quasi-Newton methods (see Sections 8.4 and 8.5), the initial choice of step length is known ($s_0 = 1$), and this choice will often work. In this case, looser line search criteria are definitely preferred.

### *8.3.3 Convergence*

Gradient descent algorithms cannot be guaranteed to give convergence to a global minimizer. Generally, we might expect that gradient descent algorithms would converge to a local minimizer. In fact, the best that can be guaranteed for gradient descent type algorithms is that they converge to a stationary point: $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. Consider, for example, the function $f(x, y) = x^2 - y^2$. This function is unbounded below, but if we start from $(x_0, 0)$, $x_0 \neq 0$, then any gradient descent algorithm or even Newton method would converge to $(0, 0)$. Any perturbation $(x_0, y_0)$ with $y_0 \neq 0$ would result in the iterates $(x_k, y_k)$ where $y_k \to \pm\infty$ as $k \to \infty$. So we cannot guarantee convergence to a local minimizer, just to a stationary point.

We also assume that our objective function $f$ is bounded below, and its gradient $\nabla f$ is Lipschitz continuous. Zoutendijk's theorem was originally for Wolfe condition-based line search methods but we prove that the conclusions also hold for Armijo/backtracking and Goldstein line search methods. The search directions $\boldsymbol{d}_k$ do not need to be negative gradient vectors, but can be any descent direction ($\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) < 0$).

**Theorem 8.16** *Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is bounded below with Lipschitz continuous gradient: $\|\nabla f(\boldsymbol{x}) - \nabla f(\boldsymbol{y})\|_2 \leq L \|\boldsymbol{x} - \boldsymbol{y}\|_2$ for all $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$. Also suppose that every search vector $\boldsymbol{d}_k$ is a descent direction (that is, $\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) < 0$). Then provided either the weak Wolfe conditions (8.3.7, 8.3.9), the Goldstein conditions (8.3.5, 8.3.6), or the Armijo/backtracking algorithm (Algorithm 74) is used with $s_0 \|\boldsymbol{d}_k\|_2 \geq c_3 \|\nabla f(\boldsymbol{x}_k)\|_2 > 0$ for all $k$, we have a constant $C$ depending only on $f(\boldsymbol{x}_0) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$, $L$, $c_1$, $c_2$, and $c_3$, where*

$$\sum_{k=0}^{\infty} \cos^2(\angle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)) \ \|\nabla f(\boldsymbol{x}_k)\|_2^2 \leq C. \tag{8.3.10}$$

The essential point of this theorem is that the angle $\angle \boldsymbol{d}_k, -\nabla f(\boldsymbol{x}_k)$ between the search direction $\boldsymbol{d}_k$ and the negative gradient direction $-\nabla f(\boldsymbol{x}_k)$ should not become and stay too close to $\pi/2$.

The central part of the proof is finding a constant $C'$ where

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq C' \cos^2(\angle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)) \ \|\nabla f(\boldsymbol{x}_k)\|_2^2 .$$

***Proof*** Let $\theta_k = \angle \boldsymbol{d}_k, \nabla f(\boldsymbol{x}_k)$, so $\cos\theta_k = \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)/(\|\boldsymbol{d}_k\|_2 \, \|\nabla f(\boldsymbol{x}_k)\|_2)$.

Whichever line search method is used, the sufficient decrease criterion must be satisfied. That is,

$$f(\boldsymbol{x}_{k+1}) \le f(\boldsymbol{x}_k) + c_1 s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) \qquad \text{where } \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + s_k \boldsymbol{d}_k .$$

Using the Lipschitz continuity of $\nabla f$,

$$\begin{aligned} f(\boldsymbol{x}_k + s\boldsymbol{d}_k) &= f(\boldsymbol{x}_k) + \int_0^s \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k + t\boldsymbol{d}_k)\, dt \\ &\le f(\boldsymbol{x}_k) + s\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \int_0^s \|\boldsymbol{d}_k\|_2 \; L \; \|t\, \boldsymbol{d}_k\|_2 \; dt \qquad \text{(by (8.3.2))} \\ &= f(\boldsymbol{x}_k) + s\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2} L s^2 \, \|\boldsymbol{d}_k\|_2^2 . \end{aligned} \tag{8.3.11}$$

We now need to use the different conditions for the different methods to obtain a *lower* bound on $s_k$ of the form constant $\times \left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right| / \|\boldsymbol{d}_k\|_2^2$.

(i) Wolfe conditions: From (8.3.9), $\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_{k+1}) \ge c_2 \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)$. Since $\nabla f$ is Lipschitz, $\|\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)\|_2 \le L\, s_k \; \|\boldsymbol{d}_k\|_2$. Thus

$$\begin{aligned} \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_{k+1}) &= \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \boldsymbol{d}_k^T (\nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k)) \\ &\ge \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \|\boldsymbol{d}_k\|_2 \; L \; \|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2 \\ &\ge \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + L\, s_k \; \|\boldsymbol{d}_k\|_2^2 . \end{aligned}$$

Then $c_2 \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) \le \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_{k+1}) \le \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + L\, s_k \; \|\boldsymbol{d}_k\|_2^2$. This gives

$$s_k \ge \frac{(1 - c_2) \left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|}{L \; \|\boldsymbol{d}_k\|_2^2} .$$

(ii) Goldstein conditions: From (8.3.6), $f(\boldsymbol{x}_{k+1}) \ge f(\boldsymbol{x}_k) + (1 - c_1) s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)$. Then using (8.3.11) with $s = s_k$ we have

$$\begin{aligned} f(\boldsymbol{x}_k) + s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2} L s_k^2 \, \|\boldsymbol{d}_k\|_2^2 &\ge f(\boldsymbol{x}_k) + (1 - c_1) s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k), \quad \text{and so} \\ \frac{1}{2} L s_k^2 \, \|\boldsymbol{d}_k\|_2^2 &\ge -c_1 \, s_k \, \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) = c_1 \, s_k \, \left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right| . \end{aligned}$$

Dividing by $s_k > 0$ gives

$$s_k \ge \frac{2c_1}{L} \frac{\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|}{\|\boldsymbol{d}_k\|_2^2} .$$

(iii) Armijo/backtracking: **Case 1:** $s_k = s_0$.

$$s_k = s_0 \geq c_3 \frac{\|\nabla f(\boldsymbol{x}_k)\|_2}{\|\boldsymbol{d}_k\|_2} \geq c_3 \frac{\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|}{\|\boldsymbol{d}_k\|_2^2}.$$

**Case 2:** $s_k < s_0$. Then the sufficient decrease criterion must be false for $s = 2s_k$. That is,

$$f(\boldsymbol{x}_k + 2s_k \boldsymbol{d}_k) > f(\boldsymbol{x}_k) + c_1\, 2s_k\, \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k).$$

Using (8.3.11) with $s = 2s_k$ gives

$$f(\boldsymbol{x}_k) + 2s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) + \frac{1}{2} L (2s_k)^2 \|\boldsymbol{d}_k\|_2^2 > f(\boldsymbol{x}_k) + c_1\, 2s_k\, \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k).$$

Therefore

$$2L\, s_k^2 \|\boldsymbol{d}_k\|_2^2 > 2(c_1 - 1) s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) = 2(1 - c_1) s_k \left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|.$$

Dividing by $2Ls_k \|\boldsymbol{d}_k\|_2^2 > 0$ gives

$$s_k > \frac{(1 - c_1)\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|}{\|\boldsymbol{d}_k\|_2^2}.$$

Given a lower bound $s_k \geq c_4 \left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right| / \|\boldsymbol{d}_k\|_2^2$, we can substitute for $s_k$ in the sufficient decrease criterion:

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) - f(\boldsymbol{x}_k) &\leq c_1 s_k \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) \leq c_1 c_4 \frac{\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|}{\|\boldsymbol{d}_k\|_2^2} \boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) \\ &= -c_1 c_4 \frac{\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|^2}{\|\boldsymbol{d}_k\|_2^2} = -c_1 c_4 \frac{\left|\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k)\right|^2}{\|\boldsymbol{d}_k\|_2^2 \|\nabla f(\boldsymbol{x}_k)\|_2^2} \|\nabla f(\boldsymbol{x}_k)\|_2^2 \\ &= -c_1 c_4 \cos^2 \theta_k \, \|\nabla f(\boldsymbol{x}_k)\|_2^2 . \end{aligned}$$

Reversing the direction of the inequality,

$$f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1}) \geq c_1 c_4 \cos^2 \theta_k \, \|\nabla f(\boldsymbol{x}_k)\|_2^2 .$$

Summing over $k$ we have a telescoping sum on the left:

$$f(\boldsymbol{x}_0) - \lim_{k \to \infty} f(\boldsymbol{x}_k) = c_1 c_4 \sum_{k=0}^{\infty} \cos^2 \theta_k \, \|\nabla f(\boldsymbol{x}_k)\|_2^2 .$$

The left-hand side is bounded above by $f(\boldsymbol{x}_0) - \inf_{\boldsymbol{x}} f(\boldsymbol{x})$, which is finite as $f$ is bounded below. Thus

$$\sum_{k=0}^{\infty} \cos^2 \theta_k \ \|\nabla f(\boldsymbol{x}_k)\|_2^2$$

is finite, as we wanted to show. □

Zoutendijk's theorem (Theorem 8.16) and its variants show that line search methods converge globally to a stationary point under mild conditions. However, this theorem does not give much information about how quickly these methods converge. We can show that for the special case of exact line searches ($\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k + s_k \boldsymbol{d}_k) = 0$) applied to a strictly convex quadratic function $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$ with $A$ symmetric and positive definite, we can show a linear rate of convergence that depends on the condition number of $A$. Kantorovich's inequality [137, 243] is used to give the upper bound.

**Theorem 8.17** *Suppose $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T\boldsymbol{x} + c$ where $A$ is symmetric and positive definite. Let $f^* = \inf_{\boldsymbol{x}} f(\boldsymbol{x})$. Then if steepest descent ($\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$) is used with exact line search ($\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k + s_k \boldsymbol{d}_k) = 0$) then*

$$f(\boldsymbol{x}_{k+1}) - f^* \leq \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^2 \left[f(\boldsymbol{x}_k) - f^*\right] \qquad \text{for all } k. \tag{8.3.12}$$

***Proof*** First note that $\nabla f(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$. Since $f$ is convex (Hess $f(\boldsymbol{x}) = A$ is positive definite everywhere), the sufficient condition for a global minimizer at $\boldsymbol{x}^*$ is $A\boldsymbol{x}^* = \boldsymbol{b}$. Also, $f^* = f(\boldsymbol{x}^*) = \frac{1}{2}(A^{-1}\boldsymbol{b})^T A(A^{-1}\boldsymbol{b}) - \boldsymbol{b}^T(A^{-1}\boldsymbol{b}) + c = c - \frac{1}{2}\boldsymbol{b}^T A^{-1}\boldsymbol{b}$. Then using $A\boldsymbol{x}^* = \boldsymbol{b}$ and $A = A^T$,

$$\begin{aligned} f(\boldsymbol{x}) - f^* &= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - (A\boldsymbol{x}^*)^T\boldsymbol{x} + c - c + \frac{1}{2}\boldsymbol{b}^T A^{-1}\boldsymbol{b} \\ &= \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - (A\boldsymbol{x}^*)^T\boldsymbol{x} + \frac{1}{2}(\boldsymbol{x}^*)^T A^T A^{-1} A\boldsymbol{x}^* \\ &= \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^*)^T A(\boldsymbol{x} - \boldsymbol{x}^*). \end{aligned}$$

Let $\boldsymbol{g}_k = \nabla f(\boldsymbol{x}_k) = A\boldsymbol{x}_k - \boldsymbol{b}$. Note that $\boldsymbol{g}_k = A(\boldsymbol{x}_k - \boldsymbol{x}^*)$.

The exact line search condition implies that $\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_{k+1}) = 0 = \boldsymbol{d}_k^T(A(\boldsymbol{x}_k + s_k\boldsymbol{d}_k) - \boldsymbol{b})$, so $s_k = -\boldsymbol{d}_k^T\boldsymbol{g}_k / (\boldsymbol{d}_k^T A\boldsymbol{d}_k)$. For steepest descent, $\boldsymbol{d}_k = -\boldsymbol{g}_k$. This means $s_k = \boldsymbol{g}_k^T\boldsymbol{g}_k / (\boldsymbol{g}_k^T A\boldsymbol{g}_k)$. The update is $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - s_k\boldsymbol{g}_k$. So

$$\begin{aligned} f(\boldsymbol{x}_{k+1}) - f^* &= \frac{1}{2}(\boldsymbol{x}_{k+1} - \boldsymbol{x}^*)^T A(\boldsymbol{x}_{k+1} - \boldsymbol{x}^*) \\ &= \frac{1}{2}(\boldsymbol{x}_k - \boldsymbol{x}^* - s_k\boldsymbol{g}_k)^T A(\boldsymbol{x}_k - \boldsymbol{x}^* - s_k\boldsymbol{g}_k) \\ &= \frac{1}{2}(\boldsymbol{x}_k - \boldsymbol{x}^*)^T A(\boldsymbol{x}_k - \boldsymbol{x}^*) - (s_k\boldsymbol{g}_k)^T A(\boldsymbol{x}_k - \boldsymbol{x}^*) + \frac{1}{2}s_k^2\boldsymbol{g}_k^T A\boldsymbol{g}_k \end{aligned}$$

$$
\begin{aligned}
&= \big(f(\boldsymbol{x}_k) - f^*\big) - s_k \boldsymbol{g}_k^T \boldsymbol{g}_k + \frac{1}{2} s_k^2 \boldsymbol{g}_k^T A \boldsymbol{g}_k \\
&= \big(f(\boldsymbol{x}_k) - f^*\big) - \frac{(\boldsymbol{g}_k^T \boldsymbol{g}_k)^2}{\boldsymbol{g}_k^T A \boldsymbol{g}_k} + \frac{1}{2} \left( \frac{\boldsymbol{g}_k^T \boldsymbol{g}_k}{\boldsymbol{g}_k^T A \boldsymbol{g}_k} \right)^2 \boldsymbol{g}_k^T A \boldsymbol{g}_k \\
&= \big(f(\boldsymbol{x}_k) - f^*\big) - \frac{1}{2} \frac{(\boldsymbol{g}_k^T \boldsymbol{g}_k)^2}{\boldsymbol{g}_k^T A \boldsymbol{g}_k}.
\end{aligned}
$$

But $f(\boldsymbol{x}_k) - f^* = \frac{1}{2} \boldsymbol{g}_k^T A^{-1} \boldsymbol{g}_k$. So

$$
f(\boldsymbol{x}_{k+1}) - f^* = \big(f(\boldsymbol{x}_k) - f^*\big) \left\{ 1 - \frac{(\boldsymbol{g}_k^T \boldsymbol{g}_k)^2}{(\boldsymbol{g}_k^T A^{-1} \boldsymbol{g}_k)(\boldsymbol{g}_k^T A \boldsymbol{g}_k)} \right\}.
$$

To bound the expression

$$
1 - \frac{(\boldsymbol{g}^T \boldsymbol{g})^2}{(\boldsymbol{g}^T A \boldsymbol{g})(\boldsymbol{g}^T A^{-1} \boldsymbol{g})}, \tag{8.3.13}
$$

above in terms of $\kappa_2(A)$, we expand $\boldsymbol{g}$ in terms of eigenvectors $\boldsymbol{v}_j$ where $A\boldsymbol{v}_j = \lambda_j \boldsymbol{v}_j$; since $A$ is symmetric we can choose the $\boldsymbol{v}_j$ to be orthonormal: $\boldsymbol{g} = \sum_j \gamma_j \boldsymbol{v}_j$. Since $A$ is positive definite, $\lambda_j > 0$ for all $j$. We can order the eigenvalues $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$. Note that $\kappa_2(A) = \lambda_n / \lambda_1$. By orthonormality of the eigenvectors, $\boldsymbol{g}^T \boldsymbol{g} = \sum_j \gamma_j^2$ while $\boldsymbol{g}^T A \boldsymbol{g} = \sum_j \lambda_j \gamma_j^2$ and $\boldsymbol{g}^T A^{-1} \boldsymbol{g} = \sum_j \lambda_j^{-1} \gamma_j^2$. Then

$$
1 - \frac{(\boldsymbol{g}^T \boldsymbol{g})^2}{(\boldsymbol{g}^T A \boldsymbol{g})(\boldsymbol{g}^T A^{-1} \boldsymbol{g})} = 1 - \frac{(\sum_j \gamma_j^2)^2}{(\sum_j \lambda_j \gamma_j^2)(\sum_j \lambda_j^{-1} \gamma_j^2)}.
$$

Setting $z_j = \gamma_j^2 / \sum_\ell \gamma_\ell^2$, we can write

$$
1 - \frac{(\boldsymbol{g}^T \boldsymbol{g})^2}{(\boldsymbol{g}^T A \boldsymbol{g})(\boldsymbol{g}^T A^{-1} \boldsymbol{g})} = 1 - \frac{1}{(\sum_j \lambda_j z_j)(\sum_j \lambda_j^{-1} z_j)}. \tag{8.3.14}
$$

Note that $z_j \geq 0$ for all $j$, and that $\sum_j z_j = 1$. So maximizing (8.3.13) over $\boldsymbol{g}$ is equivalent to maximizing $(\sum_j \lambda_j z_j)(\sum_j \lambda_j^{-1} z_j)$ over non-negative $z_j$'s that sum to one.

Let $\varphi(\lambda) = \lambda^{-1}$ and $\psi(\lambda)$ the linear interpolant of $\varphi$ at $\lambda = \lambda_1$ and $\lambda = \lambda_n$. Note that $\varphi$ is positive and convex on $[\lambda_1, \lambda_n] \subset (0, \infty)$. Let $\widehat{\lambda} = \sum_j z_j \lambda_j \in [\lambda_1, \lambda_n]$. By convexity,

$$
\varphi(\widehat{\lambda}) = \varphi(\sum_j z_j \lambda_j) \leq \sum_j z_j \, \varphi(\lambda_j) \leq \sum_j z_j \, \psi(\lambda_j) = \psi(\sum_j z_j \lambda_j).
$$

The last equality holds since $\psi$ is affine and $\sum_j z_j = 1$. Since $\varphi(\widehat{\lambda}) = \widehat{\lambda}^{-1}$ is positive,

$$(\sum_j \lambda_j z_j)(\sum_j \lambda_j^{-1} z_j) = \varphi(\widehat{\lambda})^{-1} \sum_j z_j \, \varphi(\lambda_j) \le \varphi(\widehat{\lambda})^{-1} \psi(\widehat{\lambda}) = \widehat{\lambda}\, \psi(\widehat{\lambda}).$$

The task now is to bound $\lambda\,\psi(\lambda)$ for $\lambda \in [\lambda_1,\ \lambda_n]$. The function $\lambda \mapsto \lambda\,\psi(\lambda)$ is concave since $\psi$ is affine and decreasing: $(d/d\lambda)^2[\lambda\,\psi(\lambda)] = 2\,\psi'(\lambda) + \lambda\psi''(\lambda) < 0$. So we seek $\lambda$ where $(d/d\lambda)[\lambda\,\psi(\lambda)] = 0$. This maximum occurs at $\lambda = (\lambda_1 + \lambda_n)/2$. Substituting this into $\lambda\,\psi(\lambda)$ gives $\frac{1}{2}(\lambda_1 + \lambda_n)\frac{1}{2}(\lambda_1^{-1} + \lambda_n^{-1}) = \frac{1}{4}(2 + (\lambda_1/\lambda_n) + (\lambda_n/\lambda_1))$. Using this bound in (8.3.14) gives

$$\begin{aligned} 1 - \frac{(\boldsymbol{g}^T\boldsymbol{g})^2}{(\boldsymbol{g}^T A\boldsymbol{g})(\boldsymbol{g}^T A^{-1}\boldsymbol{g})} &\le 1 - \frac{1}{\frac{1}{2}(\lambda_1 + \lambda_n)\frac{1}{2}(\lambda_1^{-1} + \lambda_n^{-1})} \\ &= 1 - \frac{4\lambda_1\lambda_n}{(\lambda_1 + \lambda_n)^2} = \frac{(\lambda_1 - \lambda_n)^2}{(\lambda_1 + \lambda_n)^2} = \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^2. \end{aligned}$$

Finally, we see that

$$f(\boldsymbol{x}_{k+1}) - f^* \le \left(\frac{\kappa_2(A) - 1}{\kappa_2(A) + 1}\right)^2 \left(f(\boldsymbol{x}_k) - f^*\right),$$

as we wanted. □

While Theorem 8.17 is for an idealized line search and only gives an upper bound, the dependence on $\kappa_2(A)$ is clearly observed in practice. The bound (8.3.12) gives a bound for one step. It is possible to get better performance from gradient descent if, for example, $-\nabla f(\boldsymbol{x})$ happens to point nearly directly to the minimizer. This will happen in the case of quadratic functions if $\boldsymbol{x} - \boldsymbol{x}^*$ is nearly an eigenvector of $A$. But if $\kappa_2(A)$ is large, small deviations from being an eigenvector will result in wildly incorrect directions.

To illustrate the behavior of gradient descent with exact line search, we show a plot of the function values $f(\boldsymbol{x}_k)$ against $k$ in Figure 8.3.1. This is shown for $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x}$ where $A = \mathrm{diag}(100^{k/n} \mid k = 0, 1, 2, \ldots, n)$ with $n = 20$ and a randomly chosen $\boldsymbol{x}_0$. Note that $\kappa_2(A) = 100$. While there is an initially rapid reduction in the function value, this slows to a geometric rate. This geometric rate is estimated to be $(f(\boldsymbol{x}_{k+1}) - f^*)/(f(\boldsymbol{x}_k) - f^*) \approx \exp(-4.9259 \times 10^{-2})$ by using the function values at $k = 50$ and $k = 100$, while $((\kappa_2(A) - 1)/(\kappa_2(A) + 1))^2 \approx \exp(-4.000 \times 10^{-2})$. The number of iterations needed for $f(\boldsymbol{x}_m) - f^* \le \epsilon$ is asymptotically $\log(1/\epsilon)/\log(1/\rho)$, where $\rho$ is the geometric rate of decrease per step. Thus the bound is overestimating the number of iterations needed by about 25% (not including the initial rapid reduction).

Why is there such a rapid initial rate of decrease? This is not a peculiarity of this example. This behavior is often observed. Suppose that the direction of $\boldsymbol{x}_0 - \boldsymbol{x}^*$ is uniformly distributed. Then $\boldsymbol{g}_0 = \nabla f(\boldsymbol{x}_0) = A(\boldsymbol{x}_0 - \boldsymbol{x}^*)$ will be directed strongly towards the eigenvector of $A$ with the largest eigenvalue. This makes $1 - (\boldsymbol{g}^T\boldsymbol{g})^2/((\boldsymbol{g}^T A\boldsymbol{g})(\boldsymbol{g}^T A^{-1}\boldsymbol{g}))$ close to zero, and we see rapid initial convergence.

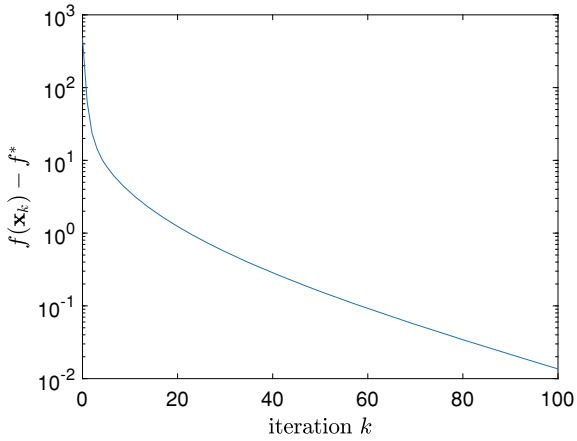



**Fig. 8.3.1** Function value vs iteration for steepest descent with exact line search

#### 8.3.3.1 Modification for Inexact Line Search Methods

Theorem 8.17 assumes exact line searches. If we use Goldstein or Wolfe-condition-based line search methods instead, we can still obtain similar bounds. If $\boldsymbol{x}_{k+1}$ is the result of either line search method and $\widetilde{\boldsymbol{x}}_{k+1}$ the result of an exact line search from $\boldsymbol{x}_k$ in direction $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$, then provided $f$ is convex and quadratic, there is a constant $0 < c^* < 1$ (depending only on $c_1$ and $c_2$) where

$$\frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_{k+1})}{f(\boldsymbol{x}_k) - f(\widetilde{\boldsymbol{x}}_{k+1})} \geq c^*. \tag{8.3.15}$$

For the Goldstein conditions (8.3.5, 8.3.6), we can take $c^* = 4c_1(1 - c_1)$. For the Wolfe conditions (8.3.7, 8.3.9), we can take $c^* = \min(4c_1(1 - c_1),\ 1 - c_2^2)$. If the Armijo/backtracking is modified so that the returned step length $s$ satisfies the sufficient decrease criterion (8.3.4) but $2s$ does not, we can prove (8.3.15) with $c^* = \min(4c_1(1 - c_1),\ 1 - c_1^2)$.

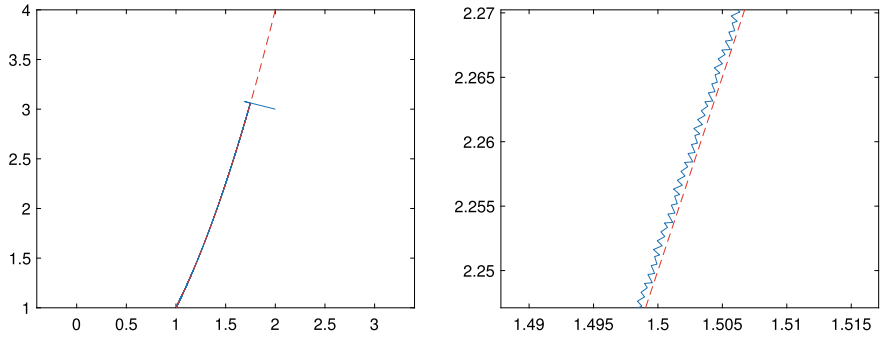
Once we have (8.3.15), we can modify the one-step bound for exact line searches (8.3.12) to

$$f(\boldsymbol{x}_{k+1}) - f^* \leq \left\{ \left(1 - c^*\right) + c^* \left( \frac{\kappa_2(A) - 1}{\kappa_2(A) + 1} \right)^2 \right\} \left[ f(\boldsymbol{x}_k) - f^* \right] \tag{8.3.16}$$

for Goldstein, Wolfe, or modified Armijo line search methods.

Even for non-quadratic functions, these bounds matter: once $\boldsymbol{x}_k$ is close to $\boldsymbol{x}^*$, we can approximate

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^*) + \nabla f(\boldsymbol{x}^*)^T (\boldsymbol{x} - \boldsymbol{x}^*) + \frac{1}{2} (\boldsymbol{x} - \boldsymbol{x}^*)^T \operatorname{Hess} f(\boldsymbol{x}^*) (\boldsymbol{x} - \boldsymbol{x}^*).$$




(A) Overall view (B) Zoom view

**Fig. 8.3.2** Steepest descent with Armijo/backtracking ($c_1 = 0.1$) applied to the Rosenbrock function (8.3.17)

So Theorem 8.17 should give a good indication of the behavior of steepest descent methods with line search close to a local minimizer, provided the Hessian matrix at the local minimizer is positive definite. In practice, we often see zigzag behavior, especially when using looser line search methods. For example, using steepest descent with the Armijo/backtracking line search method ($c_1 = 0.1$, $\boldsymbol{x}_0^T = [2,\ 3]$) for the Rosenbrock function

$$f(x, y) = 100\,(y - x^2)^2 + (x - 1)^2, \tag{8.3.17}$$

gives the zigzag behavior shown in Figure 8.3.2. The dashed curve in Figure 8.3.2 is $y = x^2$. After $k = 10^4$ iterations, the distance between $\boldsymbol{x}_k$ and the global optimizer $\boldsymbol{x}^* = [1,\ 1]^T$ is still approximately $1.08 \times 10^{-2}$.

### *8.3.4 Stochastic Gradient Method*

The *stochastic gradient method* is a stochastic version of gradient descent, specifically adapted for large data problems. This method is often called stochastic gradient descent. However, the method does not guarantee descent of the objective function, so here the word "descent" is replaced by "method".

A typical large data optimization problems is to find, given a set of data points $\left\{\, (\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i = 1, 2, \ldots, N \,\right\}$, the weight vector $\boldsymbol{w}$ that minimizes

$$f(\boldsymbol{w}) = \frac{1}{N} \sum_{i=1}^{N} \ell(\boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{w}) + R(\boldsymbol{w}), \tag{8.3.18}$$

**Algorithm 78** Stochastic gradient method.

```
1   function stochgrad(φ, x_0, (s_0, s_1, s_2, ...), n)
2     for k = 0, 1, 2, ..., n − 1
3       sample ξ_k from distribution of ξ
4       x_{k+1} ← x_k − s_k ∇_x φ(x_k, ξ_k)
5     end for
6     return x_n
7   end function
```

where $\ell$ is the loss function, measuring the error in the estimate of $\boldsymbol{y}_i$ given the "input" data value $\boldsymbol{x}_i$ using the weight vector $\boldsymbol{w}$. The function $R(\boldsymbol{w})$ is a regularizer function, that is used to prevent $\boldsymbol{w}$ from becoming "too large".

A common issue in many big data optimization problems is that the data set is too large to keep in memory, or to process efficiently as a single unit. Instead, the idea is to randomly select an index $i$ in $\{1, 2, \ldots, N\}$ and do a partial gradient descent step with respect to the loss function for the data point $(\boldsymbol{x}_i, y_i)$:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - s\, \nabla_{\boldsymbol{w}} \left[\ell(\boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{w}) + R(\boldsymbol{w})\right]_{\boldsymbol{w}=\boldsymbol{w}_k}. \tag{8.3.19}$$

The value of $s > 0$ is usually chosen to be small and fixed. The parameter $s$ can be referred to as a step length parameter, although machine learning specialists often call $s$ the *learning rate*. If $L$ is a common Lipschitz constant for $\boldsymbol{w} \mapsto \nabla_{\boldsymbol{w}} \ell(\boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{w}) + R(\boldsymbol{w})$, $i = 1, 2, \ldots, N$, then we choose $0 < s < 1/L$. No line search is performed as computing $f(\boldsymbol{w})$ for any $\boldsymbol{w}$ requires accessing the entire data set, and is therefore very expensive. Performing a line search to minimize $\ell(\boldsymbol{x}_i, \boldsymbol{y}_i;\ \boldsymbol{w}) + R(\boldsymbol{w})$ along $\boldsymbol{w} = \boldsymbol{w}_k + s\boldsymbol{d}_k$ can result in a large step in a bad direction.

It should be noted at the outset, that simply using the update (8.3.19) with a random choice of $i$ for each update *will not converge* except in special situations. However, if $s > 0$ is small, then the variance of the $\boldsymbol{w}_k$'s will approach a small but non-zero value as $k \to \infty$.

Richtarik et al. [188] have an analysis of a family of algorithms for minimizing

$$f(\boldsymbol{w}) = \mathbb{E}_{\xi}\left[\varphi(\boldsymbol{w}; \xi)\right],$$

where $\xi$ is a random variable. In this generalization, each iteration chooses a sample $\xi$ from its distribution and performs a gradient descent step in $\boldsymbol{w}$:

$$\boldsymbol{w}_{k+1} \leftarrow \boldsymbol{w}_k - s\, \nabla_{\boldsymbol{w}} \left[\varphi(\boldsymbol{w}; \xi)\right]_{\boldsymbol{w}=\boldsymbol{w}_k}.$$

The previous form of stochastic gradient descent can be recovered by making $\xi = i$ the random variable which takes each value in $\{1, 2, \ldots, N\}$ with equal probability. This generalized algorithm is shown in Algorithm 78.

Algorithm 78 does not include any line search. As noted above, performing a line search on $\varphi(\boldsymbol{x} + s\boldsymbol{d}, \xi)$ is unlikely to reduce $\mathbb{E}\left[\varphi(\boldsymbol{x}, \xi)\right]$, and accurately evaluating

$\mathbb{E}[\varphi(\boldsymbol{x}, \xi)]$ is expensive. This means that the step lengths $s_k$ should be chosen judiciously. These step lengths should not be constant but should decrease to zero as $k \to \infty$.

#### 8.3.4.1 Analysis of Stochastic Gradient Method

Before we continue, we need to note that iterates $\boldsymbol{x}_k$ are themselves random variables, along with $\xi_k$ and $\xi$. Also, $\xi_k$ is chosen independently of $\boldsymbol{x}_k$, but $\boldsymbol{x}_{k+1}$ is *not* independent of $\xi_k$. Here $\nabla\varphi(\boldsymbol{x}, \xi)$ is the gradient of $\varphi(\boldsymbol{x}, \xi)$ with respect to $\boldsymbol{x}$.

The analysis here is inspired by [188] but is simplified to present the essential aspects of the analysis.

We make the following assumptions:

(8.3.20) $\nabla\varphi(\boldsymbol{x}, \xi)$ is Lipschitz continuous in $\boldsymbol{x}$ with constant $L$ for all $\xi$;

(8.3.21) $\mathbb{V}\text{ar}[\varphi(\boldsymbol{x}, \xi)]$and $\mathbb{V}\text{ar}[\nabla\varphi(\boldsymbol{x}, \xi)]$ are bounded, indpendently of $\boldsymbol{x}$.

Since each $\boldsymbol{x}_k$ is a random variable, except possibly $\boldsymbol{x}_0$, we must be somewhat careful in analyzing the method. Starting from

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - s_k \nabla\varphi(\boldsymbol{x}_k, \xi_k),$$

we use the standard methods of analysis to show that

$$\begin{aligned}\varphi(\boldsymbol{x}_{k+1}, \xi) &\le \varphi(\boldsymbol{x}_k, \xi) + \nabla\varphi(\boldsymbol{x}_k, \xi)^T(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) + \frac{1}{2}L\,\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\|_2^2 \\ &= \varphi(\boldsymbol{x}_k, \xi) - s_k\,\nabla\varphi(\boldsymbol{x}_k, \xi)^T\nabla\varphi(\boldsymbol{x}_k, \xi_k) + \frac{1}{2}L\,s_k^2\,\|\nabla\varphi(\boldsymbol{x}_k, \xi_k)\|_2^2\,.\end{aligned}$$

Given the value of $\boldsymbol{x}_k$ we then have the independent random variable $\xi_k$ used to compute $\boldsymbol{x}_{k+1}$, and another independent random variable $\xi$ used for obtaining $f(\boldsymbol{x}) = \mathbb{E}[\varphi(\boldsymbol{x}, \xi)]$. Both $\xi_k$ and $\xi$ have the same probability distribution. Taking expectations conditional on the given value of $\boldsymbol{x}_k$ we get

$$\begin{aligned}\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) \mid \boldsymbol{x}_k\right] &= \mathbb{E}\left[\varphi(\boldsymbol{x}_{k+1}, \xi) \mid \boldsymbol{x}_k\right] \\ &\le \mathbb{E}[\varphi(\boldsymbol{x}_k, \xi) \mid \boldsymbol{x}_k] - s_k\,\mathbb{E}\left[\nabla\varphi(\boldsymbol{x}_k, \xi)^T\nabla\varphi(\boldsymbol{x}_k, \xi_k) \mid \boldsymbol{x}_k\right] \\ &\quad + \frac{1}{2}L\,s_k^2\,\mathbb{E}\left[\|\nabla\varphi(\boldsymbol{x}_k, \xi_k)\|_2^2 \mid \boldsymbol{x}_k\right].\end{aligned}$$

Note that by independence of $\xi$ and $\xi_k$

$$\begin{aligned}&\mathbb{E}\left[\nabla\varphi(\boldsymbol{x}_k, \xi)^T\nabla\varphi(\boldsymbol{x}_k, \xi_k) \mid \boldsymbol{x}_k\right] \\ &\quad = \mathbb{E}[\nabla\varphi(\boldsymbol{x}_k, \xi) \mid \boldsymbol{x}_k]^T\,\mathbb{E}[\nabla\varphi(\boldsymbol{x}_k, \xi_k) \mid \boldsymbol{x}_k] \\ &\quad = \|\nabla f(\boldsymbol{x}_k)\|_2^2\,,\end{aligned}$$

and that

$$\begin{aligned}\mathbb{E}\left[\|\nabla\varphi(\boldsymbol{x}_k,\xi_k)\|_2^2 \mid \boldsymbol{x}_k\right] &= \|\mathbb{E}\left[\nabla\varphi(\boldsymbol{x}_k,\xi_k) \mid \boldsymbol{x}_k\right]\|_2^2 + \mathbb{V}\mathrm{ar}\left[\nabla\varphi(\boldsymbol{x}_k,\xi_k)\right] \\ &= \|\nabla f(\boldsymbol{x}_k)\|_2^2 + \mathbb{V}\mathrm{ar}\left[\nabla\varphi(\boldsymbol{x}_k,\xi_k)\right].\end{aligned}$$

Then we see that

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) \mid \boldsymbol{x}_k\right] \le f(\boldsymbol{x}_k) - s_k(1-\frac{1}{2}Ls_k)\,\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \frac{1}{2}Ls_k^2\mathbb{V}\mathrm{ar}\left[\nabla\varphi(\boldsymbol{x}_k,\xi_k) \mid \boldsymbol{x}_k\right].$$

If we bound $\mathbb{V}\mathrm{ar}\left[\nabla\varphi(\boldsymbol{x}_k,\xi_k) \mid \boldsymbol{x}_k\right] \le M$ by (8.3.21) and assume $0 < s_k \le 1/L$, then

$$\mathbb{E}\left[f(\boldsymbol{x}_{k+1}) \mid \boldsymbol{x}_k\right] \le f(\boldsymbol{x}_k) - \frac{1}{2}s_k\,\|\nabla f(\boldsymbol{x}_k)\|_2^2 + \frac{1}{2}LMs_k^2.$$

Note that we cannot guarantee that $f(\boldsymbol{x}_{k+1}) < f(\boldsymbol{x}_k)$ no matter how small $s_k > 0$ is.

Summing over the iterations $k$ and taking expectations over all possible sequences of iterates, we have

$$\mathbb{E}\left[f(\boldsymbol{x}_n) - f(\boldsymbol{x}_0)\right] \le -\sum_{k=0}^{n-1} s_k\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|_2^2\right] + \frac{1}{2}LM\sum_{k=0}^{n-1}s_k^2 \qquad \text{so that}$$

$$\sum_{k=0}^{n-1} s_k\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|_2^2\right] \le \mathbb{E}\left[f(\boldsymbol{x}_0) - f(\boldsymbol{x}_n)\right] + \frac{1}{2}LM\sum_{k=0}^{n-1}s_k^2.$$

To say anything useful about the gradients as $n \to \infty$, we need $\sum_{k=0}^{n-1} s_k^2$ to be bounded; so we assume that $\sum_{k=0}^{\infty} s_k^2$ is finite, but $\sum_{k=0}^{\infty} s_k$ is infinite. In that case we can show that

$$\liminf_{k\to\infty} \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|_2\right] = 0.$$

That is, for any $\epsilon,\ K > 0$ there is a $k \ge K$ where $\mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|_2\right] < \epsilon$. If we keep $s_k = s$ constant for all $k$, then we can show that

$$\liminf_{k\to\infty} \mathbb{E}\left[\|\nabla f(\boldsymbol{x}_k)\|_2\right] \le \frac{1}{2}L\,M\,s = \mathcal{O}(s) \qquad \text{as } s \downarrow 0.$$

In this case we do not expect convergence of $\nabla f(\boldsymbol{x}_k)$, but rather $\nabla f(\boldsymbol{x}_k)$ is usually small while there may be occasional large spikes.

#### 8.3.4.2 Speed of Convergence: The Linear Least Squares Case

Consider a simple linear least squares problem

$$\min_{\boldsymbol{a}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \boldsymbol{x}_i^T \boldsymbol{a})^2.$$

Each term is $\varphi(\boldsymbol{a}, i) = (y_i - \boldsymbol{x}_i^T \boldsymbol{a})^2$. We can use Algorithm 78, sampling $i$ from $\{1, 2, \ldots, N\}$ uniformly with $\nabla\varphi(\boldsymbol{a}, i) = 2(\boldsymbol{x}_i^T \boldsymbol{a} - y_i)\boldsymbol{x}_i$. The minimizing $\boldsymbol{a} = \boldsymbol{a}^*$ is given by $\sum_{i=1}^{N} (\boldsymbol{x}_i^T \boldsymbol{a}^* - y_i)\boldsymbol{x}_i = \boldsymbol{0}$. Letting $\gamma_i = y_i - \boldsymbol{x}_i^T \boldsymbol{a}^*$, the optimality conditions become $\sum_{i=1}^{N} \gamma_i \boldsymbol{x}_i = \boldsymbol{0}$. If $i = i(k)$ is the sample chosen from $\{1, 2, \ldots, N\}$ at step $k$, then

$$\begin{aligned} \boldsymbol{a}_{k+1} &= \boldsymbol{a}_k - 2s_k(\boldsymbol{x}_i^T \boldsymbol{a}_k - y_i)\boldsymbol{x}_i \\ &= (I - 2s_k \boldsymbol{x}_i \boldsymbol{x}_i^T)\boldsymbol{a}_k + 2s_k y_i \boldsymbol{x}_i. \end{aligned}$$

Using $y_i = \gamma_i + \boldsymbol{x}_i^T \boldsymbol{a}^*$,

$$\boldsymbol{a}_{k+1} - \boldsymbol{a}^* = (I - 2s_k \boldsymbol{x}_i \boldsymbol{x}_i^T)(\boldsymbol{a}_k - \boldsymbol{a}^*) + 2s_k \gamma_i \boldsymbol{x}_i.$$

Taking expectations,

$$\mathbb{E}\left[\boldsymbol{a}_{k+1} - \boldsymbol{a}^*\right] = \mathbb{E}\left[(I - 2s_k \boldsymbol{x}_i \boldsymbol{x}_i^T)(\boldsymbol{a}_k - \boldsymbol{a}^*)\right] + 2s_k \mathbb{E}\left[\gamma_i \boldsymbol{x}_i\right].$$

But $\boldsymbol{x}_i$ is independent of $\boldsymbol{a}_k$ as $i$ is chosen independently of $\boldsymbol{a}_k$, and $\mathbb{E}[\gamma_i \boldsymbol{x}_i] = N^{-1} \sum_{j=1}^{N} \gamma_j \boldsymbol{x}_i = \boldsymbol{0}$, so

$$\mathbb{E}\left[\boldsymbol{a}_{k+1} - \boldsymbol{a}^*\right] = (I - 2s_k \mathbb{E}\left[\boldsymbol{x}_i \boldsymbol{x}_i^T\right])\mathbb{E}\left[\boldsymbol{a}_k - \boldsymbol{a}^*\right].$$

Let $B = \mathbb{E}\left[\boldsymbol{x}_i \boldsymbol{x}_i^T\right] = N^{-1} \sum_{j=1}^{N} \boldsymbol{x}_j \boldsymbol{x}_j^T$; this is a symmetric positive semi-definite matrix. We suppose that it is positive definite. If $\boldsymbol{e}_k = \boldsymbol{a}_k - \boldsymbol{a}^*$ then

$$\mathbb{E}\left[\boldsymbol{e}_{k+1}\right] = (I - 2s_k B)\,\mathbb{E}\left[\boldsymbol{e}_k\right].$$

If we keep $s_k = s$ for all $k$ then we still get $\mathbb{E}[\boldsymbol{e}_k] \to \boldsymbol{0}$ as $k \to \infty$ provided $s\,\lambda_{\max}(B) < 1$. Furthermore, this convergence is geometric:

$$\left\|\mathbb{E}\left[\boldsymbol{e}_{k+1}\right]\right\|_2 \leq \max(|1 - 2s_k\lambda_{\min}(B)|\,,\ |1 - 2s_k\lambda_{\max}(B)|)\ \left\|\mathbb{E}\left[\boldsymbol{e}_k\right]\right\|_2.$$

The real difficulty is that *variance* of $\boldsymbol{e}_k$ does *not* go to zero geometrically unless $\gamma_i = 0$ for all $i$ (the "no noise" case). The equations are

$$\begin{aligned} \boldsymbol{e}_{k+1}^T \boldsymbol{e}_{k+1} = \boldsymbol{e}_k^T (I - 2s_k \boldsymbol{x}_i \boldsymbol{x}_i^T)^2 \boldsymbol{e}_k + 4s_k \gamma_i \boldsymbol{x}_i^T (I - 2s_k \boldsymbol{x}_i \boldsymbol{x}_i^T)\boldsymbol{e}_k \\ + 4s_k^2 \gamma_i^2 \boldsymbol{x}_i^T \boldsymbol{x}_i. \end{aligned}$$

Taking expectations and using independence of $\gamma_i$, $\boldsymbol{x}_i$ with $\boldsymbol{e}_k$ gives

$$\begin{aligned}\mathbb{E}\left[\|\boldsymbol{e}_{k+1}\|_2^2\right] &= \mathbb{E}\left[\boldsymbol{e}_k^T(I - 2s_k\mathbb{E}\left[\boldsymbol{x}_i\boldsymbol{x}_i^T\right])^2\boldsymbol{e}_k\right] \\ &\quad + 4s_k\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T(I - 2s_k\boldsymbol{x}_i\boldsymbol{x}_i^T)\right]\mathbb{E}\left[\boldsymbol{e}_k\right] \\ &\quad + 4s_k^2\mathbb{E}\left[\gamma_i^2\boldsymbol{x}_i^T\boldsymbol{x}_i\right].\end{aligned}$$

Note that $\mathbb{E}\left[\boldsymbol{x}_i\boldsymbol{x}_i^T\right] = B$, and $\mathbb{E}\left[\boldsymbol{e}_k\right] \to \boldsymbol{0}$ geometrically if $s_k = s > 0$ for all $k$. Also, as $\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T\right] = \boldsymbol{0}^T$, $\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T(I - 2s_k\boldsymbol{x}_i\boldsymbol{x}_i^T)\right] = -2s_k\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T\boldsymbol{x}_i\boldsymbol{x}_i^T\right]$. If $\|\boldsymbol{x}_i\|_2 = 1$ for all $i$, then $\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T\boldsymbol{x}_i\boldsymbol{x}_i^T\right] = \mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T\right] = \boldsymbol{0}^T$. Let $\Gamma = \mathbb{E}\left[\gamma_i^2\boldsymbol{x}_i^T\boldsymbol{x}_i\right]$. Note that $\Gamma \geq 0$ with equality if and only if $\gamma_i = 0$ for all $i$ ("no noise"). Supposing, for example, that $\|\boldsymbol{x}_i\|_2 = 1$ for all $i$,

$$\begin{aligned}\mathbb{E}\left[\|\boldsymbol{e}_{k+1}\|_2^2\right] &= \mathbb{E}\left[\boldsymbol{e}_k^T(I - 2s_kB)^2\boldsymbol{e}_k\right] - 8s_k\mathbb{E}\left[\gamma_i\boldsymbol{x}_i^T\boldsymbol{x}_i\boldsymbol{x}_i^T\right]\mathbb{E}\left[\boldsymbol{e}_k\right] + 4s_k^2\Gamma \\ &= \mathbb{E}\left[\boldsymbol{e}_k^T(I - 2s_kB)^2\boldsymbol{e}_k\right] + 4s_k^2\Gamma \\ &\leq (1 - 2s_k\lambda_{\min}(B))^2\,\mathbb{E}\left[\|\boldsymbol{e}_k\|_2^2\right] + 4s_k^2\Gamma,\end{aligned}$$

provided $|1 - 2s_k\lambda_{\max}(B)| \leq 1 - 2s_k\lambda_{\min}(B)$ for all $k$. From these equations it is clear that for $\mathbb{E}\left[\|\boldsymbol{e}_k\|_2^2\right]$ to go to zero as $k \to \infty$ we need $s_k \to 0$ as $k \to \infty$. If $\sum_{k=0}^{\infty} s_k = +\infty$ and $\sum_{k=0}^{\infty} s_k^2 < +\infty$ then we get $\boldsymbol{e}_k \to \boldsymbol{0}$ as $k \to \infty$.

### 8.3.5 Simulated Annealing

*Simulated annealing* [203, 226] was developed as a means of global discrete optimization. The physical insight is that a physical system that is cooled very rapidly often only goes part way towards the minimum energy configuration, and often ends up in a local but not global minimum of the total potential energy. On the other hand, cooling the same system slowly allowed the system to come close to the global minimum of total potential energy.

Temperature here relates to thermal energy, which is kinetic energy at a microscopic level. Thermal energy allows increases in the potential energy at the level of individual molecules and atoms. This, like the stochastic gradient method, is a nonmonotone optimization method. That is, the objective function value can sometimes increase, even though we wish to minimize this value.

Simulated annealing is also a stochastic optimization process. However, there are some differences with Stochastic Gradient Method. While the step lengths $s_k$ in Stochastic Gradient Method are typically chosen with the property that $\sum_{k=0}^{\infty} s_k = +\infty$ and $\sum_{k=0}^{\infty} s_k^2$ finite, in simulated annealing, the corresponding step lengths $s_k$ decrease much more slowly.

The starting point for simulated annealing is the Metropolis–Hastings algorithm (Algorithm 71 of Section 7.4.2.3). Let $X$ be the state space we wish to optimize over. We choose the function $q$ generating the probability distribution $p(z) = q(z)/\sum_{x\in X} q(x)$ to be

**Algorithm 79** Simulated annealing. The input $U$ is a generator of independent uniformly distributed values over (0, 1).

```
1   generator simannealing(f, g, sched, x_0, n, U)
2     for k = 0, 1, 2, ..., n − 1
3       β_k ← sched(k)
4       sample x′ from g(x | x_k)
5       if next(U) < min(1, exp(−β_k(f(x′) − f(x_k))))
6         x_{k+1} ← x′
7       else
8         x_{k+1} ← x_k
9       end if
10      yield x_{k+1}
11    end for
12  end generator
```

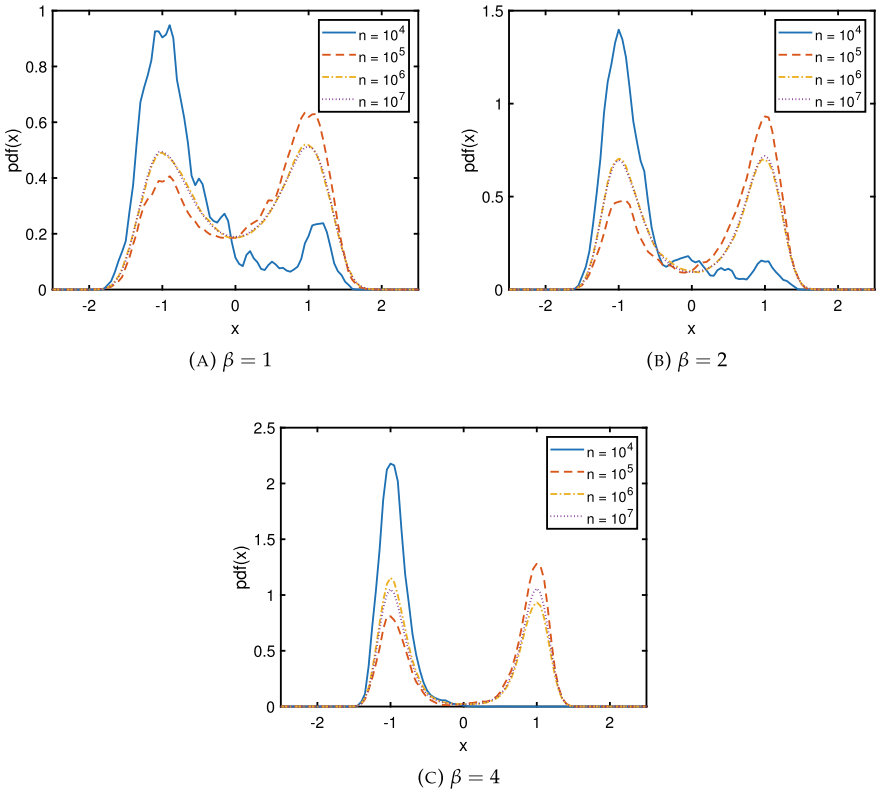
$$q(x) = \exp(-\beta f(x))$$

where $f$ is the function we wish to minimize. Physically $\beta$ corresponds to $1/(k_B T)$ where $k_B$ is *Boltzmann's constant* and $T$ the absolute temperature. The larger the value of $\beta$ the more concentrated the distribution is near the global minimum; at the other extreme, if $\beta = 0$ simulated annealing becomes essentially a random walk.

If we take $\beta = +\infty$, then the only steps of the Metropolis–Hastings algorithm that are accepted are steps where the candidate iterate $x'$ satisfies $f(x') < f(x_k)$; this is a randomized descent algorithm: pick a neighbor of $x_k$ at random. If the neighbor has a smaller value of $f$, this neighbor becomes $x_{k+1}$. Otherwise $x_{k+1} = x_k$. Clearly, this method will become stuck in local minima.

The task then is to choose a value of $\beta > 0$ that will give sufficient concentration near the global minimizer. Choosing the value of $\beta$ too small will mean that it takes a long time to leave a local minimizer; choosing the value of $\beta$ too large will result in the algorithm simply "exploring" the state space $X$ without much regard for the objective function. The ideal then is to start with a small value of $\beta$, but increase it slowly so as to prevent the iterates from becoming stuck in local minimizer. This corresponds to "annealing" a physical system by slowly reducing the temperature. A complete algorithm can be seen in Algorithm 79. Note that the factor $g(x' \mid x_k)/g(x_k \mid x')$ in the Metropolis–Hastings algorithm (Algorithm 71) is typically not used in simulating annealing, since accurately sampling from a given probability distribution is not crucial for optimization and leads to greater computational complexity.

In most applications, the state space $X$ is the set of vertices of an undirected graph $G$. The function $g(y \mid x)$ can then be taken to be $1/\deg(x)$ whenever $y$ is a neighbor of $x$ and zero otherwise. That is, sampling $x'$ from $g(x \mid x_k)$ amounts to picking a neighbor $x'$ of $x_k$ with equal probability. Note that in this case, if the degrees of the nodes are unequal, then the asymptotic probability distribution for fixed $\beta$ is no longer proportional to $\exp(-\beta f(x))$, but rather proportional to $\deg(x)\ \exp(-\beta f(x))$. The factor of $\deg(x)$ is usually not particularly important in most applications.




(A) $\beta = 1$ (B) $\beta = 2$

(C) $\beta = 4$

**Fig. 8.3.3** Simulated annealing estimated probability distribution for $f(x) = (x^2 - 1)^2$ for $\beta = 1$, 2, and 4

Figure 8.3.3 shows the estimated probability distributions obtained by simulated annealing for $f(x) = (x^2 - 1)^2$ where $n$ is the number of steps of simulated annealing used. The values of $\beta$ for the plots are 1, 2, and 4, respectively. At each step starting at $x$, we choose one of $x \pm \delta$ each with probability $1/2$. The value of $\delta$ used for generating Figure 8.3.3 was $\delta = 0.1$. The number of steps used $n = 10^4, 10^5, 10^6$, and $10^7$. Since $f$ has two local minima, both of which are also global minimizers, the equilibrium probability distributions are bi-modal. As $\beta$ increases, the probability distributions become increasingly concentrated around $x = \pm 1$. Note that the estimated probability distributions for $n = 10^7$ are very close to the actual equilibrium probability distribution $p(x) = \exp(-\beta f(x)) / \int_{-\infty}^{+\infty} \exp(-\beta f(y))\, dy$. However, for $n = 10^4$, the estimated distribution is very far from the actual equilibrium distribution.

As $n$ becomes larger, it is clear that the estimated probability distribution should come closer to actual equilibrium distribution. However, it is remarkable how large $n$ has to be to obtain a close approximation to the actual equilibrium probability distribution. There are two reasons for this. One is that where $f(x)$ varies slowly, the simulated annealing process acts like a random walk. This means that in one

dimension, the distance traveled in $n$ steps is $\mathcal{O}(\delta\sqrt{n})$. The second reason is that the local maximum of $f(x)$ at $x = 0$ acts as a barrier between the two local minimizers. For the largest value of $\beta = 4$, the barrier is the strongest, and for $n = 10^4$ the estimated probability distribution is almost entirely on the left-hand side ($x \leq 0$).

How slowly should $\beta$ be reduced in order to avoid becoming trapped? This is shown in Laarhoven and Aarts [251] to be

$$\beta_k \leq \frac{\ln(k + k_0)}{\Gamma} \tag{8.3.22}$$

where $\Gamma$ is the depth of the deepest local minimizer that is not the global minimizer. In practice, this is too slow for most users as it will result in every $x \in X$ being visited infinitely many times for finite $X$. Most users use a faster "cooling schedule" than this but lose the guarantee that the global minimizer will be found. The point of using randomized search algorithms is precisely to avoid exhaustive search.

For the common case where $X$ is the set of vertices of a graph $G$, with neighbors chosen equally likely, the structure of $G$ can have a great impact on the performance of stochastic search algorithms for optimization. Just how this structure affects the performance of simulated annealing and other stochastic search algorithms is, at time of writing, still only partly understood.

**Exercises.**

(1) Use gradient descent to minimize $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ starting from $(x, y) = (0, 0)$ using Algorithm 73, first with no linesearch (the *linesearch* function simply returns a constant value) using step length parameters $s = 10^{-k}$, $k = 1, 2, 3$, and stopping the algorithm when $\|\nabla f(\boldsymbol{x}_k)\|_2 < 10^{-3}$. Record the number of function and gradient evaluations for each step length. Also use gradient descent with the Armijo/backtracking algorithm (Algorithm 74) for finding the step length, using $s_0 = 1$. Compare the number of function and gradient evaluations for this method with the method without adaptive linesearch.

(2) Repeat the previous Exercise with the Rosenbrock function $f(x, y) = (x - 1)^2 + 100(y - x^2)^2$ starting at $(x, y) = (-1, 0)$.

(3) In this Exercise, we consider two simple modifications to the Armijo/backtracking linesearch: (1) set the "start" step length $s_0$ equal to the step length $s$ used in the previous step; (2) set the "start" step length $s_0$ equal to twice the step length $s$ used in the previous step. Compare the number of function and gradient evaluations used in applying the modified methods to minimizing the Rosenbrock function $f(x, y) = (x - 1)^2 + 100(y - x^2)^2$ starting at $(x, y) = (-1, 0)$, stopping when $\|\nabla f(\boldsymbol{x}_k)\|_2 < 10^{-3}$. Are either of these modifications worthwhile? Why do you think so?

(4) Use gradient descent to minimize $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ starting from $(x, y) = (0, 0)$ using Algorithm 73 with each of the three linesearch algorithms outlined: Armijo/backtracking (Algorithm 74), the Goldstein algorithm (Algorithm 75), and the Wolfe linesearch algorithm (Algorithm 76, 77).

Use the stopping criterion that $\|\nabla f(\boldsymbol{x}_k)\|_2 < 10^{-3}$. Use reasonable or suggested values for the parameters involved. Which method performs best?

(5) For $\boldsymbol{x}^*$ an approximate minimizer of $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ computed by one of the previous Exercises, compute the eigenvalues of Hess $f(\boldsymbol{x}^*)$. Compare the rate of convergence you observed with the results of Theorem 8.17. Note that Theorem 8.17 gives a bound for an exact linesearch method.

(6) Separate the terms of the function $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ into separate functions $\varphi_1(x, y) = x^2, \varphi_2(x, y) = xy^2$, etc., and set $\varphi(x, y, \xi) = \varphi_\xi(x, y)$ for $\xi = 1, 2, \ldots, 6$. We can then apply the stochastic gradient method (Algorithm 78) to minimize $f(x, y)$, sampling $\xi$ from a uniformly distribution over $\{1, 2, \ldots, 6\}$. Do this first for fixed step lengths $s_k = s_0$ for all $k$ with $s_0 = 10^{-j}$, $j = 1, 2, 3$. Stop when $\|\nabla f(\boldsymbol{x}_k)\|_2 < 10^{-2}$. Next use Algorithm 78 with $s_k = 10^{-1}/(k + 1)^{3/4}$. Compare the rates of convergence of these methods with each other and with deterministic gradient descent algorithms.

(7) Let $\phi(x) = -2/(1 + 2x^2) + x^2/10$, and let $f(x) = \frac{1}{2}(\phi(x - 1) + \phi(x + 1))$. Use the stochastic gradient method (Algorithm 78) for minimizing $f$ with update $x_{k+1} \leftarrow x_k - s_k\, \phi'(x - \xi_k)$ where $\xi_k$ is sampled from $\{\pm 1\}$ with each choice equally likely. Compare using $s_k = 0.2$ for all $k$, and using $s_k = 0.5\, k^{-3/4}$. There are two global minimizers of $f: x \approx \pm 0.9248753266450438$. Create a histogram for the values $x_k$ with bins of width $10^{-2}$. Use $N = 10^7$ steps of the stochastic gradient algorithm, and plot the resulting histograms against $x$; use a logarithmic scale in the vertical axis. What is different about the two histograms you generate? What does this imply about the ability of the stochastic gradient algorithm to find the global minimizer of a function with multiple local minimizers?

(8) Apply the simulated annealing method (Algorithm 79) to minimizing the function $f: \mathbb{R}^{20} \to \mathbb{R}$ from Exercise 4: for a point $\boldsymbol{x} \in \mathbb{R}^{20}$ the neighbors of $\boldsymbol{x}$ are the points $\boldsymbol{x} \pm \eta \boldsymbol{e}_j$ where $\boldsymbol{e}_j$ is the $j$th standard basis vector and $\eta = 10^{-1}$. Start from a randomly generated point, and run the method for $10^7$ steps using a constant value of $\beta$. Use the values $\beta = \frac{1}{2}$, 1, 2, 4. How close are the points generated by simulated annealing to the global minimizer? [**Note:** It should be expected that simulated annealing will give close to correct values for *some* components of the vector $\boldsymbol{x}_k$. How many components of $\boldsymbol{x}_k$ are close to the minimizer of $g$?]

## 8.4 Second Derivatives and Newton's Method

The standard first-order conditions for the unconstrained minimization of a function $f: \mathbb{R}^n \to \mathbb{R}$ are $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. This is a system of $n$ equations in $n$ unknowns. We can apply the multivariate Newton method (Algorithm 43 in Section 3.3.4) and many of its variations to the problem of solving $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. This requires solving the linear system $(\text{Hess } f(\boldsymbol{x}_k))\, \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ where Hess $f(\boldsymbol{x})$ is the Hessian matrix of $f$ at $\boldsymbol{x}$.

Newton's method has the advantage of rapid quadratic convergence when starting close to the solution and the Hessian matrix at the solution is invertible. However, the cost of each iteration can be substantial if the dimension $n$ is large. If $n$ is large, then sparse solution methods (see Section 2.3) or iterative methods (see Section 2.4) can be used to solve the linear systems that arise in Newton's method.

Newton's method in this form converges to solutions of $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. This is a necessary but not sufficient condition for a local minimizer except under special assumptions. For example, if $f$ is smooth and convex, then $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$ is both a necessary and sufficient condition for a global minimizer. In general, there are saddle points (where Hess $f(\boldsymbol{x})$ has both negative and positive eigenvalues) and local maximizers (where Hess $f(\boldsymbol{x})$ has only negative eigenvalues). Newton's method regards saddle points and local maximizers as equally valid solutions of $\nabla f(\boldsymbol{x}) = \boldsymbol{0}$. Yet, for optimization purposes, we wish to avoid these points. A consequence of this behavior of Newton's method is that the Newton step $\boldsymbol{d}_k$ satisfying Hess $f(\boldsymbol{x}_k)\,\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ might not be a descent direction: $\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) = -\nabla f(\boldsymbol{x}_k)^T \left(\text{Hess } f(\boldsymbol{x}_k)\right)^{-1} \nabla f(\boldsymbol{x}_k)$ could be positive for an indefinite Hessian matrix. This means that any line search method based on the sufficient decrease criterion (8.3.4) can fail. This includes the Armijo/backtracking, Goldstein, and Wolfe-based line search methods.

In order to accommodate Hessian matrices that are not positive definite, we need to modify the Newton method. We can do this by modifying the Hessian matrix used [190, Sec. 3.4]. We can also use a different globalization strategy than using line searches, such as trust region methods [190, Chap. 4].

Hessian modification strategies solve the equation $B_k \boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$ for $\boldsymbol{d}_k$ where $B_k = \text{Hess } f(\boldsymbol{x}_k) + E_k$ where $E_k$ is a symmetric matrix designed to make $B_k$ positive definite. As long as $B_k$ is positive definite,

$$\boldsymbol{d}_k^T \nabla f(\boldsymbol{x}_k) = -\boldsymbol{d}_k^T B_k \boldsymbol{d}_k < 0,$$

provided $\boldsymbol{d}_k \neq \boldsymbol{0}$. We can choose $E = \alpha I$ where $\alpha \geq 0$. More specifically, we set $\alpha = 0$ if Hess $f(\boldsymbol{x}_k)$ is already positive definite. Alternatively, if $\lambda_{\min}(\text{Hess } f(\boldsymbol{x}_k)) < 0$, we can set $\alpha$ to be $\alpha = -\lambda_{\min}(\text{Hess } f(\boldsymbol{x}_k)) + \gamma \left\|\text{Hess } f(\boldsymbol{x}_k)\right\|_2$ where $\gamma > 0$ ensures that the condition number $\kappa_2(B_k)$ is bounded above by $1/\gamma$.

Another Hessian modification strategy is to modify the Cholesky factorization algorithm to give the Cholesky factorization $L_k L_k^T = B_k + D_k$ where $D_k$ is a diagonal matrix with non-negative diagonal entries. The diagonal entries of $D_k$ are computed as needed to ensure that a factorization exists. However, we should not use diagonal entry that is too close to the minimum necessary to continue the factorization, as this will result in excessively large numbers in following steps of the factorization. To see why consider

$$B = \begin{bmatrix} \beta & \boldsymbol{b}^T \\ \boldsymbol{b} & \widehat{B} \end{bmatrix}, \qquad D = \begin{bmatrix} \delta & \boldsymbol{0}^T \\ \boldsymbol{0} & \widehat{D} \end{bmatrix}.$$

If $\beta < 0$ then we need $\delta > -\beta > 0$. Computing one step of the factorization of $B + D$ gives

$$B + D = \left[\begin{array}{c|c} \sqrt{\delta+\beta} & \\ \hline \boldsymbol{b}/\sqrt{\delta+\beta} & I \end{array}\right] \left[\begin{array}{c|c} 1 & \\ \hline & \widehat{B} + \widehat{D} - \boldsymbol{b}\boldsymbol{b}^T/(\delta+\beta) \end{array}\right] \left[\begin{array}{c|c} \sqrt{\delta+\beta} & \boldsymbol{b}^T/\sqrt{\delta+\beta} \\ \hline & I \end{array}\right].$$

If $\delta + \beta \approx 0$ then $\boldsymbol{b}/\sqrt{\delta+\beta}$ will be large, and subsequent entries in $\widehat{D}$ will also need to be large to compensate for $-\boldsymbol{b}\boldsymbol{b}^T/(\delta+\beta)$. Since this can happen at many of the following steps in the factorization, the resulting linear system will not give a useful direction for searching. For more details of an algorithm that handles these issues by ensuring that $\delta + \beta$ is sufficiently positive, see [190, Sec. 3.4].

An alternative approach is to use a trust region method (see [190, Chap. 4] or [54]). At each step of a trust region method we have a quadratic model function $m_k(\boldsymbol{d}) := \frac{1}{2}\boldsymbol{d}^T B_k \boldsymbol{d} + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d} + f(\boldsymbol{x}_k)$ where $B_k$ is either the Hessian matrix Hess $f(\boldsymbol{x}_k)$ or some approximation to it. Instead of minimizing $m_k(\boldsymbol{d})$ over all possible $\boldsymbol{d}$ (which gives the usual Newton step $\boldsymbol{d} = -B_k^{-1}\nabla f(\boldsymbol{x}_k)$), we minimize over a trust region

$$\|\boldsymbol{d}\| \le \Delta_k. \tag{8.4.1}$$

Usually, the norm used for the trust region is the 2-norm, but other norms can be used if they are computationally convenient. If we use the 2-norm, then the following conditions are equivalent to minimizing $\boldsymbol{d}$ over the trust region [190, Sec. 4.3]:

$$\begin{aligned} (B_k + \lambda_k I)\boldsymbol{d}_k &= -\nabla f(\boldsymbol{x}_k), \\ \lambda_k \ge 0 \quad \text{and} \quad \Delta_k - \|\boldsymbol{d}_k\|_2 &\ge 0 \\ \lambda_k(\Delta_k - \|\boldsymbol{d}_k\|_2) &= 0. \end{aligned}$$

The update step depends on how well the model represents the true objective function. To make the comparison we compute

$$\rho_k = \frac{f(\boldsymbol{x}_k) - f(\boldsymbol{x}_k + \boldsymbol{d}_k)}{m_k(\boldsymbol{0}) - m_k(\boldsymbol{d}_k)}.$$

If $\rho_k$ is positive and sufficiently large (say $\rho_k > \eta$ where $\frac{1}{4} > \eta \ge 0$) then we accept the step: $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k + \boldsymbol{d}_k$. Otherwise $\boldsymbol{x}_{k+1} \leftarrow \boldsymbol{x}_k$, which is a "null step". If $\rho_k$ is too small or negative (say $\rho_k \le 1/4$), then the trust region should be shrunk (say $\Delta_{k+1} \leftarrow \frac{1}{2}\min(\Delta_k, \|\boldsymbol{d}_k\|)$). If $\rho_k$ is large enough (say $\rho_k \ge 3/4$) and $\|\boldsymbol{d}_k\| = \Delta_k$ then we can increase the size of the trust region up to a maximum size: $\Delta_{k+1} \leftarrow \min(2\,\Delta_k, \Delta_{\max})$. If neither of the update formulas for the trust region size is used then $\Delta_{k+1} \leftarrow \Delta_k$. Further details, including how to solve the trust region optimization problem are given in [190, Chap. 4].

The trust region method can be applied with $B_k = \text{Hess}\, f(\boldsymbol{x}_k)$ even where Hess $f(\boldsymbol{x}_k)$ is not positive semi-definite. Trust region methods can also be applied if $B_k$ is simply an approximation to Hess $f(\boldsymbol{x}_k)$, with the cost of degrading the rate of convergence.

**Exercises.**

(1) Use the basic Newton method for minimizing $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ with the following modifications: (1) use Armijo backtracking line search, and (2) if Hess $f(\boldsymbol{x}_k)$ is not positive definite, then use $\boldsymbol{d}_k = -\nabla f(\boldsymbol{x}_k)$. Report the number of function, gradient, and Hessian evaluations.

(2) Show how requiring the sufficient decrease criterion for $\|\nabla f(\boldsymbol{x})\|_2^2$ will prevent convergence to a local minimizer if $\boldsymbol{x}_k$ is close to a critical point $\widehat{\boldsymbol{x}}$ where Hess $f(\widehat{\boldsymbol{x}})$ has negative eigenvalues.

(3) For nonlinear least squares problems with $f(\boldsymbol{w}) = N^{-1}\sum_{i=1}^{N}(y_i - g_i(\boldsymbol{w}))^2$ the *Gauss–Newton method* just uses 1st derivatives of the $g_i$ functions. First show that

$$\text{Hess } f(\boldsymbol{w}) = 2N^{-1}\left[\sum_{i=1}^{N}\nabla g_i(\boldsymbol{w})\,\nabla g_i(\boldsymbol{w})^T + \sum_{i=1}^{N}(g_i(\boldsymbol{w}) - y_i)\,\text{Hess } g_i(\boldsymbol{w})\right].$$

The basic Gauss–Newton update is

$$\begin{aligned}\boldsymbol{d}_k &\leftarrow -\left[\sum_{i=1}^{N}\nabla g_i(\boldsymbol{w}_k)\,\nabla g_i(\boldsymbol{w}_k)^T\right]^{-1}\sum_{i=1}^{N}(g_i(\boldsymbol{w}_k) - y_i)\,\nabla g_i(\boldsymbol{w}_k)\\ s_k &\leftarrow \textit{linesearch}(\boldsymbol{x}_k, \boldsymbol{d}_k, \ldots)\\ \boldsymbol{x}_{k+1} &\leftarrow \boldsymbol{x}_k + s_k\boldsymbol{d}_k\end{aligned}$$

where the step length parameter $s_k$ is either one, or determined by a line search method with starting value $s = 1$.

(a) Show that $\boldsymbol{d}_k$ is always a descent direction provided the vectors $\{\nabla g_i(\boldsymbol{w}_k) \mid i = 1, 2, \ldots, N\}$ are linearly independent.

(b) While the rate of convergence is not expected to be superlinear, explain why convergence is rapid if $g_i(\boldsymbol{w}_k) \approx y_i$ for all $i$.

(c) For most statistical fitting problems, the Gauss–Newton method converges quickly for large $N$. Suppose that $g_i(\boldsymbol{w}) = g(\boldsymbol{w}, \boldsymbol{x}_i)$ where the data points $(\boldsymbol{x}_i, y_i)$, $i = 1, 2, \ldots, N$, are sampled independently from the same probability distribution. If $(\boldsymbol{X}, Y)$ random variables with the same probability distribution, use the Law of Large Numbers to show that

$$N^{-1}\sum_{i=1}^{N}\nabla g_i(\boldsymbol{w})\,\nabla g_i(\boldsymbol{w})^T \to \mathbb{E}\left[\nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})\nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})^T\right]$$

$$N^{-1}\sum_{i=1}^{N}(g_i(\boldsymbol{w}) - y_i)\,\text{Hess } g_i(\boldsymbol{w}) \to \mathbb{E}\left[(g(\boldsymbol{w}, \boldsymbol{X}) - Y)\,\text{Hess}_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})\right] \quad \text{and}$$

$$N^{-1}\sum_{i=1}^{N}(g_i(\boldsymbol{w}) - y_i)\,\nabla g_i(\boldsymbol{w}) \to \mathbb{E}\left[(g(\boldsymbol{w}, \boldsymbol{X}) - Y)\,\nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})\right].$$

Note that if $\nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})$ and $\mathrm{Hess}_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})$ are close to constant, then $\mathbb{E}[g(\boldsymbol{w}, \boldsymbol{X}) - Y] \approx 0$ and so $\mathrm{Hess}\, f(\boldsymbol{w}) \approx \mathbb{E}\left[\nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X}) \nabla_{\boldsymbol{w}} g(\boldsymbol{w}, \boldsymbol{X})^T\right]$ if $\boldsymbol{w}$ is close to the minimizer $\boldsymbol{w}^*$.

(d) Show that the Gauss–Newton method can be implemented by performing a QR factorization of $J(\boldsymbol{w}) := [\nabla g_1(\boldsymbol{w}), \nabla g_2(\boldsymbol{w}), \ldots, \nabla g_N(\boldsymbol{w})]$.

(4) Implement a method for solving the problem $\min_{\boldsymbol{x}} \boldsymbol{g}^T \boldsymbol{x} + \frac{1}{2} \boldsymbol{x}^T B \boldsymbol{x}$ subject to $\|\boldsymbol{x}\|_2 \le \Delta$ by solving the conditions $(B + \lambda I)\boldsymbol{x} = -\boldsymbol{g}$ with $\lambda \ge 0$, $\lambda > 0$ implies that $\|\boldsymbol{x}\|_2 = \Delta$ and $B + \lambda I$ is positive semi-definite. We can do this by solving $\varphi(\lambda) = 0$ where $\varphi(\lambda) = \left\|(B + \lambda I)^{-1} \boldsymbol{g}\right\|_2 - \Delta$ if $B + \lambda I$ is positive definite and $+\infty$ otherwise. If $B$ is positive definite and $\left\|B^{-1} \boldsymbol{g}\right\|_2 \le \Delta$ then we set $\lambda = 0$. Otherwise, find $\lambda_{\max}$ where we can guarantee that $\left\|(B + \lambda_{\max} I)^{-1} \boldsymbol{g}\right\|_2 < \Delta$ and perform the bisection method for solving $\varphi(\lambda) = 0$. Test this on some randomly generated symmetric matrices $B$ and randomly generated vectors $\boldsymbol{g}$. Make sure that the desired conditions are satisfied by the computed solution $\boldsymbol{x}$.

(5) Nesterov and Nemirovskii [187, Chap. 2] outline a theory of *self-concordant functions*: a convex function $f : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ is $M$-self-concordant if for any $\boldsymbol{x}$ and $\boldsymbol{d}$,

$$\frac{\left|(d^3/ds^3) f(\boldsymbol{x} + s\boldsymbol{d})\right|}{\left[(d^2/ds^2) f(\boldsymbol{x} + s\boldsymbol{d})\right]^{3/2}} \le 2M^{-1/2} \qquad \text{provided } f(\boldsymbol{x} + s\boldsymbol{d}) < +\infty.$$

Nesterov and Nemirovskii showed that if $f$ is $M$-self-concordant, then applying a guarded Newton method with the sufficient decrease criterion, the number of steps needed to achieve $\|\nabla f(\boldsymbol{x}_k)\|_2 < \eta$ is $\mathcal{O}(1 + M^{-1}(f(\boldsymbol{x}_0) - \min_{\boldsymbol{x}} f(\boldsymbol{x})) + \log\log(1/\eta))$ as $\eta \downarrow 0$. The hidden constants do not depend on $f$, $\boldsymbol{x}_0$, $M$, or $\eta$.

(a) Show that if $f$ is $M$-self-concordant, then $g(\boldsymbol{x}) = f(A\boldsymbol{x} + \boldsymbol{b})$ is also $M$-self-concordant *with the same* $M$.

(b) Show that if $f_1$ is $M_1$-self-concordant and $f_2$ is $M_2$-self-concordant, then $f_1 + f_2$ is $M$-self-concordant for $M = \min(M_1, M_2)$.

(c) Show that if $f$ is $M$-self-concordant, then $a\, f$ is $(a\, M)$-self-concordant for $a > 0$.

(d) Show that $f(u) = -\ln u$ is 1-self-concordant over $u > 0$.

(6) Implement the modified Cholesky factorization algorithm outlined in [190, Sec. 3.4] to compute $B + D = LL^T$ where $B$ is the given matrix and $D$ is diagonal with non-negative diagonal entries. Test it on randomly generated symmetric matrices, and on matrices $B = X^T X$ with $X$ randomly generated. Is it consistent with the standard Cholesky factorization for positive-definite matrices? For matrices $B$ that are not positive definite, compare the largest diagonal entry of $D$ with the size of the largest negative eigenvalue of $B$.

(7) The *dog-leg trust region method* gives an approximate solution to the trust region problem $\min_{\boldsymbol{p}} \boldsymbol{g}^T \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^T B \boldsymbol{p}$ subject to $\|\boldsymbol{p}\|_2 \le \Delta$: Let $\boldsymbol{p}_1 = -s\boldsymbol{g}$

where $0 \leq s \leq \Delta$ is chosen to minimize $\boldsymbol{g}^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T B \boldsymbol{p}$ with $\boldsymbol{p} = -s\boldsymbol{g}$. Let $\boldsymbol{p}_2 = -B^{-1}\boldsymbol{g}$ if $B$ is positive definite, and $\boldsymbol{p}_2 = \boldsymbol{p}_1$ otherwise. We choose the approximate solution to be the point $\boldsymbol{p} = (1-\theta)\boldsymbol{p}_1 + \theta \boldsymbol{p}_2$ that minimizes $\boldsymbol{g}^T \boldsymbol{p} + \frac{1}{2}\boldsymbol{p}^T B \boldsymbol{p}$ over $0 \leq \theta \leq 1$ subject to $\|\boldsymbol{p}\|_2 \leq \Delta$. Implement this method. If you wish, incorporate this into a trust region algorithm and test the complete method on the Rosenbrock function. How does the complete method perform compared to other Newton-type methods?

(8) A common criticism of Newton-based methods is that they are expensive to implement for large-scale problems as computing the Hessian matrix Hess $f(\boldsymbol{x}_k)$ is already very expensive in time and memory compared to computing function values and gradients. For $f(\boldsymbol{x}) = (\boldsymbol{x}^T \boldsymbol{x})^2$ with $\boldsymbol{x} \in \mathbb{R}^n$ show that Hess $f(\boldsymbol{x}) = 8\boldsymbol{x}\boldsymbol{x}^T + 4(\boldsymbol{x}^T \boldsymbol{x})I$ which requires $\mathcal{O}(n^2)$ floating point operations. On the other hand, show that for any vector $\boldsymbol{u}$, Hess $f(\boldsymbol{x})\boldsymbol{u}$ can be computed in $\mathcal{O}(n)$ floating point operations.

(9) ⚠ For large-scale optimization problems, keeping the critique of the previous Exercise in mind, first give arguments that the cost of computing Hess $f(\boldsymbol{x})\boldsymbol{u}$ is at most proportional to the cost of computing $f(\boldsymbol{x})$ using automatic differentiation. If Hess $f(\boldsymbol{x})$ is always positive definite, then the conjugate gradient method can be used to compute the solution of the Newton equation Hess $f(\boldsymbol{x})\boldsymbol{d} = -\nabla f(\boldsymbol{x})$. Otherwise, describe how to use the Lanczos iteration to implement a trust region method.

## 8.5 Conjugate Gradient and Quasi-Newton Methods

Conjugate gradient methods for solving $B\boldsymbol{d} = -\boldsymbol{g}$ with $B$ symmetric positive definite were developed in Sections 2.4.2 and 2.4.2.1. Part of the derivation relied on the fact that if $B$ is symmetric positive definite then solving $B\boldsymbol{d} = -\boldsymbol{g}$ is equivalent to finding the minimum of $\phi(\boldsymbol{d}) := \frac{1}{2}\boldsymbol{d}^T B\boldsymbol{d} + \boldsymbol{g}^T \boldsymbol{d} + c$. Conjugate gradient methods can be generalized from convex quadratic functions to general smooth functions. However, in generalizing the method, some of the properties that could be proven for solving linear systems no longer hold for general optimization problems. This further means that some expressions that are equivalent in the context of solving a linear system of equations are no longer equivalent in the context of general optimization.

Quasi-Newton methods build a sequence of positive-definite approximations $B_k \approx$ Hess $f(\boldsymbol{x}_k)$ by means of rank-1 or rank-2 updates using the changes in the gradient $\nabla f(\boldsymbol{x}_k) - \nabla f(\boldsymbol{x}_{k-1})$ and position $\boldsymbol{x}_k - \boldsymbol{x}_{k-1}$.

### *8.5.1 Conjugate Gradients for Optimization*

It is noted in Section 2.4.2.1 that part of the conjugate gradient method can be derived from the condition that $\boldsymbol{x}_{k+1}$ minimizes the convex quadratic function $\phi(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x}$ over $\boldsymbol{x} \in \text{span}\{\boldsymbol{p}_0, \boldsymbol{p}_1, \ldots, \boldsymbol{p}_k\}$, together with the property

**Algorithm 80** Conjugate gradients algorithm — version 2

```
1   function conjgrad2(A, b, x_0, ε)
2     k ← 0;  r_0 ← Ax_0 − b;  p_0 ← −r_0
3     while ‖r_k‖_2 ≥ ε
4       q_k ← A p_k
5       α_k ← −p_k^T r_k / p_k^T q_k
6       x_{k+1} ← x_k + α_k p_k
7       r_{k+1} ← r_k + α_k q_k
8       β_k ← r_{k+1}^T r_{k+1} / r_k^T r_k
9       p_{k+1} ← −r_{k+1} + β_k p_k
10      k ← k + 1
11    end while
12    return x_k
13  end function
```

that $\boldsymbol{p}_i^T A \boldsymbol{p}_j = 0$ for $i \neq j$ (the conjugacy property). The conjugacy condition implies that $\boldsymbol{x}_{k+1} - \boldsymbol{x}_k \in \text{span}\left\{\boldsymbol{p}_k\right\}$, so that we can find $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + s_k \boldsymbol{p}_k$ by using an exact line search.

We can generalize this algorithm to functions $f(\boldsymbol{x})$ that are neither quadratic nor convex, and to use inexact line search methods. However, in this process, we lose some properties of the method. For convex quadratic objective functions and exact line searches, we have the following properties of the iterates of the conjugate gradient method (recall that $\boldsymbol{r}_i = A\boldsymbol{x}_i - \boldsymbol{b}$):

$$
\begin{aligned}
\boldsymbol{r}_i^T \boldsymbol{r}_j &= 0 \qquad \text{if } i \neq j, \\
\boldsymbol{r}_i^T \boldsymbol{p}_j &= 0 \qquad \text{if } j < i, \\
\boldsymbol{p}_i^T A \boldsymbol{p}_j &= 0 \qquad \text{if } i \neq j, \\
\text{span}\left\{\boldsymbol{p}_0, \ldots, \boldsymbol{p}_k\right\} &= \text{span}\left\{\boldsymbol{r}_0, \ldots, \boldsymbol{r}_k\right\} \\
&= \text{span}\left\{\boldsymbol{r}_0, A\boldsymbol{r}_0, \ldots, A^k \boldsymbol{r}_0\right\} \quad \text{for all } k \geq 0.
\end{aligned}
$$

This is the main content of Theorem 2.20.

In order to avoid flipping pages to compare with the original algorithm, we repeat the unpreconditioned standard linear conjugate gradient algorithm (Algorithm 27) as *conjgrad2*.

From this, we will see how to derive the conjugate gradient algorithm for optimization. First, if $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} - \boldsymbol{b}^T \boldsymbol{x} + c$, then the residual $\boldsymbol{r} = A\boldsymbol{x} - \boldsymbol{b} = \nabla f(\boldsymbol{x})$. So we replace the computation of the residual $\boldsymbol{r}_k$ on line 7 with $\boldsymbol{r}_k \leftarrow \nabla f(\boldsymbol{x}_k)$. The other thing to remember is that in the general situation, there is no matrix $A$. The closest thing we have to $A$ for a general smooth function $f$ is Hess $f(\boldsymbol{x})$, the Hessian matrix of $f$ at $\boldsymbol{x}$. But in general, Hess $f(\boldsymbol{x})$ is neither constant nor easily computable. So we should avoid any explicit reference to $A$. These references occur on line 4 in computing $\boldsymbol{q}_k$, which is used to compute the step length $\alpha_k$ on line 5 and the update of the residual on line 7. For the step length, we simply use a suitable line search

**Algorithm 81** Conjugate gradients algorithm for optimization — version 1

```
1   function conjgradoptFR(f, ∇f, x_0, linesearch, params, ϵ)
2     k ← 0;  r_0 ← ∇f(x_k);  p_0 ← −r_0
3     while ‖r_k‖_2 ≥ ϵ
5       s_k ← linesearch(f, ∇f, x_k, p_k, params)
6       x_{k+1} ← x_k + s_k p_k
7       r_{k+1} ← ∇f(x_{k+1})
8       β_k ← r_{k+1}^T r_{k+1} / r_k^T r_k
9       p_{k+1} ← −r_{k+1} + β_k p_k
10      k ← k + 1
11    end while
12    return x_k
13  end function
```

algorithm. This gives Algorithm 81 (*conjgradoptFR*) below, which is known as the Fletcher–Reeves conjugate gradient algorithm.

#### 8.5.1.1 Line Search Algorithms for Conjugate Gradient Optimization Algorithms

Now we need to ask: what makes for a suitable line search algorithm for this generalized conjugate gradient algorithm? All of the line search algorithms we have discussed assume the direction of the line search (here $\boldsymbol{p}_k$) must be a descent direction: $\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k) = \boldsymbol{p}_k^T \boldsymbol{r}_k < 0$. This is clearly true for $k = 0$ as $\boldsymbol{p}_0 = -\boldsymbol{r}_0$. But can we guarantee this will be true for $k = 1, 2, \ldots$? If we use exact line searches, then $s_k$ minimizes $f(\boldsymbol{x}_k + s\,\boldsymbol{p}_k)$ over all $s > 0$, and so $\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k + s_k \boldsymbol{p}_k) = \boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_{k+1}) = \boldsymbol{p}_k^T \boldsymbol{r}_{k+1} = 0$. From line 9, we then have

$$\begin{aligned} \boldsymbol{p}_{k+1}^T \boldsymbol{r}_{k+1} &= (-\boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k)^T \boldsymbol{r}_{k+1} \\ &= -\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} < 0, \end{aligned}$$

provided $\boldsymbol{r}_{k+1} \neq \boldsymbol{0}$, as we wanted. But in practice, we can only approximate exact line searches, and attempting something close to an exact line search can be very expensive in terms of function evaluations.

Since the descent direction condition involves gradients, we need a line search method that involves $\nabla f(\boldsymbol{x})$. This leaves Wolfe condition (8.3.7, 8.3.8) based line searches as the only practical way to properly implement conjugate gradient methods for optimization. As you may recall, we have two parameters for the Wolfe conditions: $c_1$ for the sufficient decrease criterion, and $c_2$ for the curvature condition. In order to guarantee the existence of a step satisfying (8.3.7, 8.3.8) we need $f$ bounded below as well as being smooth, and $0 < c_1 < c_2 < 1$. This can approximate exact line searches by making $c_2 > 0$ small.

If exact line searches are used so that $\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k + s_k \boldsymbol{p}_k) = 0$, then we can guarantee that $\boldsymbol{p}_{k+1}$ is a descent direction. If we use a Wolfe condition-based line search,

what value of $c_2 > 0$ can guarantee the same property? For the Fletcher–Reeve conjugate gradient method, it turns out that $0 < c_2 < \frac{1}{2}$ is sufficient to ensure that $\boldsymbol{p}_{k+1}$ is a descent direction. The reason for this is the following Lemma [190, Lemma 5.6, p. 125]:

**Lemma 8.18** *The Fletcher–Reeve conjugate gradient algorithm with Wolfe condition-based line search with* $0 < c_2 < \frac{1}{2}$ *satisfies the condition*

$$-\frac{1}{1-c_2} \leq \frac{\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k}{\|\nabla f(\boldsymbol{x}_k)\|_2^2} \leq \frac{2c_2 - 1}{1-c_2} \quad \textit{for all } k.$$

A proof is given in [190, pp. 125-126].

Lemma 8.18 implies that $\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k < 0$ for all $k$ and so $\boldsymbol{p}_k$ is a descent direction for all $k$. However, it does not imply that $\boldsymbol{p}_k$ is always a *good* descent direction. Zoutendijk's theorem (Theorem 8.16) shows good global performance if $-\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k / (\|\nabla f(\boldsymbol{x}_k)\|_2 \left\|\boldsymbol{p}_k\right\|_2)$ is bounded away from zero. The guarantee of Lemma 8.18 is that $-\nabla f(\boldsymbol{x}_k)^T \boldsymbol{p}_k / \|\nabla f(\boldsymbol{x}_k)\|_2^2$ is bounded away from zero. The Fletcher–Reeve method can run into trouble if $\left\|\boldsymbol{p}_k\right\|_2 / \|\nabla f(\boldsymbol{x}_k)\|_2 \gg 1$. Then $\boldsymbol{p}_k$ will be close to orthogonal to $-\nabla f(\boldsymbol{x}_k)$, and the line search will then be fairly short. This results in $\boldsymbol{x}_{k+1} \approx \boldsymbol{x}_k$, so $\nabla f(\boldsymbol{x}_{k+1}) \approx \nabla f(\boldsymbol{x}_k)$. The Fletcher–Reeve formula then gives $\beta_k = \|\nabla f(\boldsymbol{x}_{k+1})\|_2^2 / \|\nabla f(\boldsymbol{x}_k)\|_2^2 \approx 1$ and $\boldsymbol{p}_{k+1} = -\nabla f(\boldsymbol{x}_{k+1}) + \beta_k \boldsymbol{p}_k \approx \boldsymbol{p}_k$, so that $\left\|\boldsymbol{p}_{k+1}\right\|_2 / \|\nabla f(\boldsymbol{x}_{k+1})\|_2 \gg 1$ and the problem continues.

This effect can be seen in Figure 8.5.1, where the spiral shows the behavior of the Fletcher–Reeves conjugate gradient method where this happens. The objective function for this application of the method is $f(x, y) = y^2 - x^2 y - 3y + x^4 - x^3 + 2xy - x$ which has a global minimizer near $(-1.457,\ 4.019)$. Wolfe condition-based line searching was used with $c_1 = 10^{-2}$ and $c_2 = 0.2$. On the other hand, Figure 8.5.1 shows much better behavior of the Polak–Ribiére conjugate gradient method (8.5.1), which goes much more directly to the minimizer.

In spite of this possible poor performance of the Fletcher–Reeves method, it has been proven to be globally convergent under mild conditions [190, Thm. 5.7].

### *8.5.2 Variants on the Conjugate Gradient Method*

The formula for $\beta_k^{\text{FR}} = \boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k$ in the Fletcher–Reeves conjugate gradient algorithm could be changed to give the same behavior in the ideal case of convex quadratic objective functions and exact line searches, but hopefully better behavior for more general objective functions and inexact line searches.

One of these is the Polak–Ribiére method, which uses the formula

$$\beta_k^{\text{PR}} = (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)^T \boldsymbol{r}_{k+1} / \boldsymbol{r}_k^T \boldsymbol{r}_k. \tag{8.5.1}$$

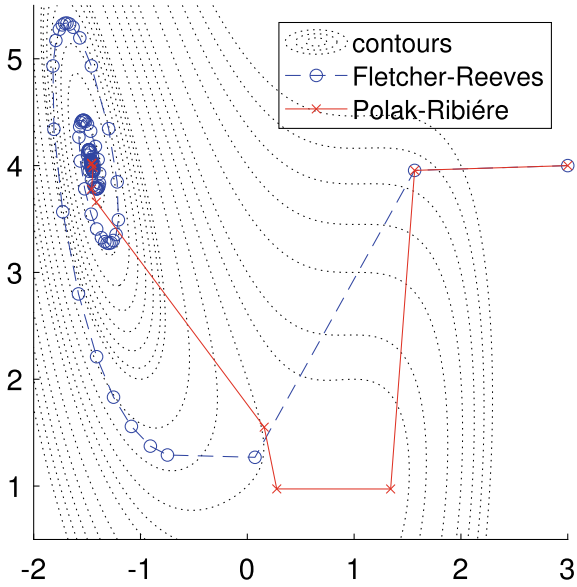



**Fig. 8.5.1** Fletchers–Reeves vs Polak Ribiére conjugate gradient methods

The Polak–Ribiére formula gives identical results to the Fletcher–Reeves formula in the ideal case since for these problems, $\boldsymbol{r}_{k+1}^T \boldsymbol{r}_k = 0$. Other formulas equivalent in the context of the ideal case include the Hestenes–Stiefel formula, which was used in the original paper on conjugate gradient methods [122]: $\beta_k^{\text{HS}} = \boldsymbol{r}_{k+1}^T (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)/(\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)^T \boldsymbol{p}_k$.

Of these alternatives, the Polak–Ribiére method is perhaps the most popular. However, even if exact line searches are used, it is possible for this method to fail to converge to a critical point [208]. Part of the issue with this example is that the value of $\beta_k^{\text{PR}}$ alternates in sign. Since in the ideal case, $\beta_k > 0$ for all $k$, another modification that works well is using the Polak–Ribiére-plus method: $\beta_k^{\text{PR+}} = \max\left((\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)^T \boldsymbol{r}_{k+1}/\boldsymbol{r}_k^T \boldsymbol{r}_k,\ 0\right)$, which can be shown to be globally convergent.

In the situation where the Fletcher–Reeves has difficulty ($\left\|\boldsymbol{p}_k\right\|_2 / \left\|\nabla f(\boldsymbol{x}_k)\right\|_2 = \left\|\boldsymbol{p}_k\right\|_2 / \left\|\boldsymbol{r}_k\right\|_2 \gg 1$), the Polak–Ribiére and Polak–Ribiére-plus methods do not have difficulty: if $\boldsymbol{r}_{k+1} = \nabla f(\boldsymbol{x}_{k+1}) \approx \nabla f(\boldsymbol{x}_k) = \boldsymbol{r}_k$ then $\beta_k^{\text{PR}} = (\boldsymbol{r}_{k+1} - \boldsymbol{r}_k)^T \boldsymbol{r}_{k+1}/\boldsymbol{r}_k^T \boldsymbol{r}_k \approx 0$ and $\boldsymbol{p}_{k+1} = -\boldsymbol{r}_{k+1} + \beta_k^{\text{PR}} \boldsymbol{p}_k \approx -\boldsymbol{r}_{k+1}$ and the method "resets" to a gradient descent algorithm. Since $\beta_k^{\text{PR+}} = \max(\beta_k^{\text{PR}}, 0)$, we also get $\beta_k^{\text{PR+}} \approx 0$ which also leads to a kind of "reset" of the Polak–Ribiére-plus method. This is illustrated in Figure 8.5.1.

### *8.5.3 Quasi-Newton Methods*

*Quasi-Newton methods* build an estimate of the Hessian matrix from the sequence of computed gradients and steps in order to obtain superlinear convergence, similar

to Newton's method. These methods are generally considered superior to conjugate gradient methods as quasi-Newton method do not rely as heavily on the line search.

The first quasi-Newton was developed by the physicist William Davidon[2], who was in search of a better method than co-ordinate descent for some energy calculations. The method, now known as the *Davidon–Fletcher–Powell* (DFP) method, starts with a default initial approximation $B_0$ to the Hessian matrix, and after each step updated the estimate $B_k$ as follows:

$$B_{k+1} = (1 - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) B_k (1 - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T) + \rho_k \boldsymbol{y}_k \boldsymbol{y}_k^T \qquad \text{where} \tag{8.5.2}$$

$$\rho_k = 1/(\boldsymbol{y}_k^T \boldsymbol{s}_k), \tag{8.5.3}$$

$$\boldsymbol{s}_k = \boldsymbol{x}_{k+1} - \boldsymbol{x}_k, \qquad \text{and } \boldsymbol{y}_k = \nabla f(\boldsymbol{x}_{k+1}) - \nabla f(\boldsymbol{x}_k). \tag{8.5.4}$$

The formula for $B_{k+1}$ is designed to satisfy $B_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$.

The standard approach is to perform a line search in the direction $\boldsymbol{d}_k = -B_k^{-1} \nabla f(\boldsymbol{x}_k)$, starting with a step length of one as for the Newton method. As for Newton methods, the Newton direction is guaranteed to be a descent direction if $B_k$ is positive definite and symmetric. In order to guarantee that $B_{k+1}$ is also symmetric positive definite, we need $\boldsymbol{s}_k^T \boldsymbol{y}_k > 0$, which does not hold in general for non-convex $f$. To ensure this, Wolfe condition-based line searches are typically used. The curvature condition (8.3.9) implies

$$\begin{aligned} \nabla f(\boldsymbol{x}_{k+1})^T (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) \geq c_2 \nabla f(\boldsymbol{x}_k)^T (\boldsymbol{x}_{k+1} - \boldsymbol{x}_k), \qquad \text{so} \\ \boldsymbol{y}_k^T \boldsymbol{s}_k \geq (c_2 - 1)\, s_k \nabla f(\boldsymbol{x}_k)^T \boldsymbol{d}_k > 0 \end{aligned}$$

provided $0 < c_2 < 1$. Thus Wolfe condition-based line searches can ensure that the $B_k$ remain symmetric positive definite provided $B_0$ is.

The DFP method can be made more efficient by introducing an update formula for $H_k = B_k^{-1}$:

$$H_{k+1} = H_k - \frac{H_k \boldsymbol{y}_k \boldsymbol{y}_k^T H_k}{\boldsymbol{y}_k^T H_k \boldsymbol{y}_k} + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T. \tag{8.5.5}$$

This update formula can be deduced through the Sherman–Morrison formula (2.1.16) applied twice. Note that $H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$.

There is a symmetry between $B_k$ and $H_k$, and between $\boldsymbol{s}_k$ and $\boldsymbol{y}_k$. Swapping both $B_k$ and $H_k$, and $\boldsymbol{s}_k$ and $\boldsymbol{y}_k$ gives new formulas for $B_{k+1}$ and $H_{k+1}$ satisfying the equations $B_{k+1}\boldsymbol{s}_k = \boldsymbol{y}_k$ and $H_{k+1}\boldsymbol{y}_k = \boldsymbol{s}_k$. This gives new update formulas that in fact perform better than Davidon's original choice. These update formulas are the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) quasi-Newton update formulas:

[2] Davidon is perhaps best known for participating in the March 8, 1971, FBI office break-in in Media, Pennsylvania, and for releasing the documents obtained to the press [176].

$$B_{k+1} = B_k - \frac{B_k \boldsymbol{s}_k \boldsymbol{s}_k^T B_k}{\boldsymbol{s}_k^T B_k \boldsymbol{s}_k} + \rho_k \boldsymbol{y}_k \boldsymbol{y}_k^T, \tag{8.5.6}$$

$$H_{k+1} = (1 - \rho_k \boldsymbol{s}_k \boldsymbol{y}_k^T) H_k (1 - \rho_k \boldsymbol{y}_k \boldsymbol{s}_k^T) + \rho_k \boldsymbol{s}_k \boldsymbol{s}_k^T. \tag{8.5.7}$$

Both the DFP and BFGS methods give superlinear convergence to a local minimizer with a positive-definite Hessian matrix, but the BFGS methods seem to be more robust.

There are other update formulas that can be used. In particular there is the Broyden family of updates:

$$B_{k+1} = (1 - \phi_k) B_{k+1}^{\text{BFGS}} + \phi_k \, B_{k+1}^{\text{DFP}}, \tag{8.5.8}$$

with $0 \le \phi_k \le 1$. Global convergence results for the BFGS method can be proven assuming that there are bounds $\mu_{\max} \ge \lambda_{\max}(\text{Hess}\, f(\boldsymbol{x}))$ and $0 < \mu_{\min} \le \lambda_{\min}(\text{Hess}\, f(\boldsymbol{x}))$ on the maximum and minimum eigenvalues of the Hessian matrices of $f$ [190, Thm. 6.5]. The key to the proof is showing that $\text{trace}(B_k) - \ln \det B_k$ cannot increase "too quickly" as $k \to \infty$. This means that neither $B_k$ nor $H_k = B_k^{-1}$ can become large "too quickly".

**Exercises.**

(1) Implement the Polak–Ribiére–plus conjugate gradient method with Wolfe condition-based line search, and apply it to the function $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ with initial point $\boldsymbol{x}_0 = (x_0, y_0) = (0, 0)$. Use default values for $c_1$ and $c_2$ in the Wolfe conditions.

(2) M.J.D. Powell stated in [207] that even for convex quadratic objective functions and exact line searches, the Fletcher–Reeve conjugate gradient method has only linear convergence and fails to have $n$-step exact convergence if $\boldsymbol{p}_0$ is *not* in the direction of $-\boldsymbol{r}_0 = -\nabla f(\boldsymbol{x}_0)$. Implement this method for $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T \boldsymbol{x}$, $\boldsymbol{x} \in \mathbb{R}^n$ but modified so that you can control $\boldsymbol{p}_0$ independently of $\boldsymbol{x}_0$. Report the angle between $\boldsymbol{p}_k$ and $-\boldsymbol{r}_k = -\nabla f(\boldsymbol{x}_k)$ as well as the sequence of values $f(\boldsymbol{x}_k)$ for all $k$.

(3) The conjugate gradient method is suitable for many large-scale problems as its memory requirements are small, and only function and gradient evaluations are required. However, it relies on a line search to approximately minimize $f(\boldsymbol{x}_k + s\,\boldsymbol{p}_k)$ over $s \ge 0$, and Wolfe condition-based line searches or similar must be used to ensure that the method converges. This is a particularly important issue in machine learning where even one pass over the data is expensive. An alternative is to pick $s_0 > 0$, compute $f(\boldsymbol{x}_k + s_0 \boldsymbol{p}_k)$, and compute the quadratic interpolant $\phi$ of the data $f(\boldsymbol{x}_k)$, $\boldsymbol{p}_k^T \nabla f(\boldsymbol{x}_k)$, and $f(\boldsymbol{x}_k + s_0 \boldsymbol{p}_k)$ for $\phi(0)$, $\phi'(0)$, and $\phi(s_0)$. Set $s_k^{(0)}$ to be the minimizer of $\phi$. Accept $s_k = s_k^{(0)}$ if the Wolfe conditions are satisfied at $s = s_k^{(0)}$, and otherwise perform the Wolfe condition-based line search. Also set $s_0$ for step $k$ to be $2\, s_{k-1}$ where $s_{k-1}$ is the accepted step size for step $k - 1$. Implement this method and compare with the default line search method used.

(4) Re-create Figure 8.5.1 using the data provided in Section 8.5.1.1. Try varying the starting point for the algorithms to see if the behavior of the methods is robust to these changes.

(5) ⚠ Suppose that the BFGS quasi-Newton method (see (8.5.6, 8.5.7)) is applied to $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T A\boldsymbol{x} + \boldsymbol{b}^T\boldsymbol{x}$ with $A$ symmetric positive definite using exact line searches. Show that if $B_0 = I$ then the search directions $\boldsymbol{d}_k$, $k = 1, 2, \ldots$, are $A$-conjugate (that is, $\boldsymbol{d}_j^T A\boldsymbol{d}_k = 0$ if $j \neq k$).

(6) Implement the BFGS method with Wolfe-condition based line search, and apply the method to minimizing $f(x, y) = x^2 + xy^2 - 10xy + 4x + y^4 - 2y$ with initial point $\boldsymbol{x}_0 = (x_0, y_0) = (0, 0)$. Use default values for the Wolfe condition parameters $c_1$ and $c_2$.

(7) Use the Shermann–Morrisson (2.1.17) or Sherman–Morrisson–Woodbury (2.1.17) formula to derive the update (8.5.7) for $H_k = B_k^{-1}$ from the update (8.5.6) for $B_k$ for the BFGS method.

(8) Show that in the BFGS update (8.5.7) for $H_{k+1}$ in terms of $H_k$, that if $H_k$ is positive definite and $1/\rho_k = \boldsymbol{y}_k^T\boldsymbol{s}_k > 0$, then $H_{k+1}$ is also positive definite. [**Hint:** First show that $H_{k+1}$ is positive definite. Then see what conditions on $\boldsymbol{z}$ hold if $\boldsymbol{z}^T H_{k+1}\boldsymbol{z} = 0$.]

## 8.6 Constrained Optimization

Constrained optimization can be represented most abstractly in terms of a *feasible set*, often denoted $\Omega \subseteq \mathbb{R}^n$:

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \qquad \text{subject to } \boldsymbol{x} \in \Omega. \tag{8.6.1}$$

Solutions exist if $f$ is continuous and either $\Omega$ is a compact (closed and bounded) subset of $\mathbb{R}^n$, or if $\Omega$ is closed and $f$ is coercive. Usually $\Omega$ is represented by equations and inequalities:

$$\Omega = \left\{ \boldsymbol{x} \in \mathbb{R}^n \mid g_i(\boldsymbol{x}) = 0 \text{ for } i \in \mathcal{E}, \text{ and } g_i(\boldsymbol{x}) \geq 0 \text{ for } i \in \mathcal{I} \right\}. \tag{8.6.2}$$

If $\mathcal{I}$ is empty but $\mathcal{E}$ is not empty, then we say (8.6.1) is an equality constrained optimization problem. If $\mathcal{I}$ is non-empty, we say (8.6.1) is an inequality constrained optimization problem.

For a general constrained optimization problem, first-order conditions can be given in terms of the *tangent cone*

$$T_\Omega(\boldsymbol{x}) = \left\{ \lim_{k\to\infty} \frac{\boldsymbol{x}_k - \boldsymbol{x}}{t_k} \mid \boldsymbol{x}_k \in \Omega,\ \boldsymbol{x}_k \to \boldsymbol{x} \text{ as } k \to \infty, \text{ and } t_k \downarrow 0 \text{ as } k \to \infty \right\}. \tag{8.6.3}$$

**Lemma 8.19** *If $\boldsymbol{x} = \boldsymbol{x}^*$ minimizes $f(\boldsymbol{x})$ over $\boldsymbol{x} \in \Omega$ and $f$ is differentiable at $\boldsymbol{x}^*$, then*

$$\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} \geq 0 \quad \text{for all } \boldsymbol{d} \in T_\Omega(\boldsymbol{x}^*). \tag{8.6.4}$$

***Proof*** Suppose $\boldsymbol{x} = \boldsymbol{x}^* \in \Omega$ minimizes $f(\boldsymbol{x})$ over $\boldsymbol{x} \in \Omega$ and $f$ is differentiable. Then for any $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}^*)$, there is a sequence $\boldsymbol{x}_k \to \boldsymbol{x}^*$ as $k \to \infty$ with $\boldsymbol{x}_k \in \Omega$ where $\boldsymbol{d}_k := (\boldsymbol{x}_k - \boldsymbol{x}^*)/t_k \to \boldsymbol{d}$ as $k \to \infty$. Since $f(\boldsymbol{x}^*) \leq f(\boldsymbol{x}_k) = f(\boldsymbol{x}^* + t_k \boldsymbol{d}_k)$,

$$0 \leq \lim_{k\to\infty} \frac{f(\boldsymbol{x}^* + t_k \boldsymbol{d}_k) - f(\boldsymbol{x}^*)}{t_k} = \nabla f(\boldsymbol{x}^*)^T \lim_{k\to\infty} \boldsymbol{d}_k = \nabla f(\boldsymbol{x}^*)^T \boldsymbol{d}.$$

This holds for any $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}^*)$ showing (8.6.4), as we wanted. □

*Constraint qualifications* relate the tangent cone $T_\Omega(\boldsymbol{x})$ to the linearizations of the constraint functions:

$$C_\Omega(\boldsymbol{x}) = \big\{ \boldsymbol{d} \in \mathbb{R}^n \mid \nabla g_i(\boldsymbol{x})^T \boldsymbol{d} = 0 \text{ for all } i \in \mathcal{E} \,, \\ \nabla g_i(\boldsymbol{x})^T \boldsymbol{d} \geq 0 \text{ for all } i \in \mathcal{I} \text{ where } g_i(\boldsymbol{x}) = 0 \big\}.$$

For equality constrained optimization ($\mathcal{I} = \emptyset$), the LICQ (8.1.2) implies that $T_\Omega(\boldsymbol{x}) = C_\Omega(\boldsymbol{x})$ as noted in Section 8.1.3.

### *8.6.1 Equality Constrained Optimization*

The theory of Section 8.1.3 for Lagrange multipliers and equality constrained optimization (8.1.5) can be immediately turned into a numerical method. To solve

$$\boldsymbol{0} = \nabla f(\boldsymbol{x}) - \sum_{i\in\mathcal{E}} \lambda_i \nabla g(\boldsymbol{x}) \tag{8.6.5}$$

$$0 = g_i(\boldsymbol{x}), \qquad i \in \mathcal{E} \tag{8.6.6}$$

for $(\boldsymbol{x}, \boldsymbol{\lambda})$ with $\boldsymbol{\lambda} = [\lambda_i \mid i \in \mathcal{E}]$ we can apply, for example, Newton's method. For unconstrained optimization, we can then perform a line search to ensure that the step improves the solution estimate. The issue in constrained optimization is that $f(\boldsymbol{x})$ alone is no longer suitable for measuring improvements. Constrained optimization problems have two objectives: staying on the feasible set, and minimizing $f(\boldsymbol{x})$. It may be necessary to increase $f(\boldsymbol{x})$ in order to return to the feasible set. Solving the Newton equations for (8.6.5, 8.6.6) gives a direction $\boldsymbol{d}$. Because of the curvature of the feasible set $\Omega$ for general functions $g_i$, moving in the direction $\boldsymbol{d}$ even if $\boldsymbol{x}$ is feasible may take the point $\boldsymbol{x} + s\,\boldsymbol{d}$ off the feasible set. This can be offset by having a *second order correction* step to move back toward the feasible set. This second order correction uses a least squares version of Newton's method to solve $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$.

**Algorithm 82** Basic Newton-based equality constrained optimization method

```
function optnewtonequality(f, ∇f, Hess f, g, ∇g, Hess g, x, λ, α, ϵ, η)
  while ‖∇f(x) − ∇g(x)^T λ‖ ≥ ϵ and ‖g(x)‖ ≥ ϵ
    H ← Hess f(x) − Σ_{i∈E} λ_i Hess g_i(x)
    β ←
max(η, −2 λ_min(H)) // ensures H + βI is positive definite
    solve [ H + βI | ∇g(x)^T ; ∇g(x) | 0 ] [ δx ; δλ ] = − [ ∇f(x) − ∇g(x)^T λ ; g(x) ]
    s ← 1; accept ← false
    while not accept
      x^+ ← x + s δx; λ^+ ← λ + s δλ
      factor ∇g(x^+)^T = Q_1 R_1 // reduced QR fact'n
      x^+ ← x^+ − R_1^{-1} Q_1^T g(x^+) // 2nd order correction
      define m_α(z) = f(z) + α Σ_{i∈E} |g_i(z)| // merit function
      if m_α(x^+) ≤ m_α(x) + c_1 ∇m_α(x)^T (x^+ − x)
        accept ← true
      else
        s ← s/2
      end if
    end while
    x ← x^+
  end while
  return (x, λ)
end function
```

Since this is an under-determined system for $|\mathcal{E}| < n$, we find the solution $\delta \boldsymbol{x}$ for $\nabla g_i(\boldsymbol{x})^T \delta \boldsymbol{x} = -g_i(\boldsymbol{x})$, $i \in \mathcal{E}$, that minimizes $\|\delta \boldsymbol{x}\|_2$, which can be done using the QR factorization of $[\nabla g_i(\boldsymbol{x}) \mid i \in \mathcal{E}]$.

For line search algorithms, we can use a merit function to determine the quality of the result of the step. Often, merit functions of the form $\boldsymbol{x} \mapsto f(\boldsymbol{x}) + \alpha \sum_{i \in \mathcal{E}} |g_i(\boldsymbol{x})|$ are used where $\alpha > \max_{i \in \mathcal{E}} |\lambda_i|$. A basic method for solving equality constrained optimization problems is shown in Algorithm 82.

If the second-order correction is skipped, then the Newton method may fail to give rapid convergence, as was noted by N. Maratos in his PhD thesis [170].

### 8.6.2 Inequality Constrained Optimization

Inequality constrained optimization is more complex, both in theory and practice. The theorem giving necessary conditions for inequality constrained optimization was only discovered in the middle of the twentieth century, while Lagrange used Lagrange multipliers in his *Mécanique Analytique* [151] (1788–1789). The necessary conditions for inequality constrained optimization are called *Kuhn–Tucker* or *Karush–Kuhn–Tucker conditions*. The first journal publication with these conditions was a paper by Kuhn and Tucker in [149] (1951), although the essence of these conditions was contained in an unpublished Master's thesis of Karush [139] (1939).

The work of Kuhn and Tucker was intended to build on the work of G. Dantzig and others [69] on *linear programming*:

$$\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x} \quad \text{subject to} \tag{8.6.7}$$

$$A\boldsymbol{x} \geq \boldsymbol{b}, \tag{8.6.8}$$

where "$\boldsymbol{a} \geq \boldsymbol{b}$" is understood to mean "$a_i \geq b_i$ for all $i$".

It was Dantzig who created the *simplex algorithm* in 1946 [69], being the first general-purpose and efficient algorithm for solving linear programs (8.6.7, 8.6.8). The simplex method can be considered an example of an *active set method* as it tracks which of the inequalities $(A\boldsymbol{x})_i \geq b_i$ is actually an equality as it updates the candidate optimizer $\boldsymbol{x}$. Since then there has been a great deal of work on alternative methods, most notably *interior point methods* that typically minimize a sequence of penalized problems such as

$$\boldsymbol{c}^T \boldsymbol{x} - \alpha \sum_{i=1}^{m} \ln((A\boldsymbol{x})_i - b_i)$$

where $\alpha > 0$ is a parameter that is reduced to zero in the limit. The first published *interior point method* was due to Karmarkar [138] (1984). Another approach is the *ellipsoidal method* of Khachiyan [120] (1979), which at each step $k$ minimizes $\boldsymbol{c}^T \boldsymbol{x}$ over $\boldsymbol{x}$ lying inside an ellipsoid centered at $\boldsymbol{x}_k$ that is guaranteed to be inside the feasible set $\{\, \boldsymbol{x} \mid A\boldsymbol{x} \geq \boldsymbol{b} \,\}$. Khachiyan's ellipsoidal method built on previous ideas of N.Z. Shor but was the first guaranteed polynomial time algorithm for linear programming. Karmarkar's algorithm also guaranteed polynomial time, but was much faster in practice than Khachiyan's method and the first algorithm to have a better time than the simplex method on average.

### 8.6.2.1 The Farkas Alternative

An important result for constrained optimization is the Farkas alternative:

**Theorem 8.20** (Farkas alternative) *Given real matrices $A$ and $B$ and a vector $\boldsymbol{b}$, then either (exclusive)*

- *there are vectors $\boldsymbol{u} \geq \boldsymbol{0}$ and $\boldsymbol{v}$ where $\boldsymbol{b} = A\boldsymbol{u} + B\boldsymbol{v}$, or*
- *there is a* vector $\boldsymbol{y} \geq \boldsymbol{0}$ *where $\boldsymbol{y}^T A \geq \boldsymbol{0}$, $\boldsymbol{y}^T B = \boldsymbol{0}$ and $\boldsymbol{y}^T \boldsymbol{b} < 0$.*

***Proof*** *(Outline)* The set $C := \{\, A\boldsymbol{u} + B\boldsymbol{v} \mid \boldsymbol{u} \geq \boldsymbol{0} \,\}$ is a closed convex set. The fact that it is convex follows from the fact that it is the image of the convex set $\{\, (\boldsymbol{u}, \boldsymbol{v}) \mid \boldsymbol{u} \geq \boldsymbol{0} \,\}$ under a linear transformation. The fact that $C$ is closed is a more technical condition which is shown in, for example, Nocedal and Wright [190, Lemma 12.15, p. 350] using an argument of R. Byrd.

The separating hyperplane theorem (Theorem 8.13) then implies that if $\boldsymbol{b} \notin C$ then there is a vector $\boldsymbol{n}$ where $\boldsymbol{n}^T\boldsymbol{b} < 0$ and $\boldsymbol{n}^T\boldsymbol{w} \geq 0$ for every $\boldsymbol{w} \in C$. So either $\boldsymbol{b} = A\boldsymbol{u} + B\boldsymbol{v}$ for some $\boldsymbol{u} \geq \boldsymbol{0}$ and $\boldsymbol{v}$ (that is, $\boldsymbol{b} \in C$) or there is $\boldsymbol{n}$ where $\boldsymbol{n}^T\boldsymbol{b} < 0$ and $\boldsymbol{n}^T(A\boldsymbol{u} + B\boldsymbol{v}) \geq 0$ for every $\boldsymbol{u} \geq \boldsymbol{0}$ and $\boldsymbol{v}$ (for $\boldsymbol{b} \notin C$).

In the case where $\boldsymbol{b} \notin C$, we first set $\boldsymbol{v} = \boldsymbol{0}$ so that $\boldsymbol{n}^T A\boldsymbol{u} \geq 0$ for all $\boldsymbol{u} \geq \boldsymbol{0}$, which implies that $\boldsymbol{n}^T A \geq \boldsymbol{0}$. Now consider $\boldsymbol{u} = \boldsymbol{0}$ so that $\boldsymbol{n}^T B\boldsymbol{v} \geq 0$ for all $\boldsymbol{v}$. Replacing $\boldsymbol{v}$ with $-\boldsymbol{v}$ (which is allowed since $\boldsymbol{v}$ can be *any* vector of the right dimension), we get $\boldsymbol{n}^T B\boldsymbol{v} \leq 0$ for all $\boldsymbol{v}$. Thus $\boldsymbol{n}^T B\boldsymbol{v} = 0$ for all $\boldsymbol{v}$. Since this is true for all vectors $\boldsymbol{v}$ we get $\boldsymbol{n}^T B = \boldsymbol{0}$. Setting $\boldsymbol{y} = \boldsymbol{n}$ gives $\boldsymbol{y}^T A \geq \boldsymbol{0}$ and $\boldsymbol{y}^T B = \boldsymbol{0}$ while $\boldsymbol{y}^T\boldsymbol{b} < 0$.

Thus if $\boldsymbol{b} \in C$ we obtain the first alternative; if $\boldsymbol{b} \notin C$ we obtain the second alternative. The two conditions are exclusive because $\boldsymbol{b} = A\boldsymbol{u} + B\boldsymbol{v}$ with $\boldsymbol{u} \geq \boldsymbol{0}$, $\boldsymbol{y}^T A \geq \boldsymbol{0}$ and $\boldsymbol{y}^T B = \boldsymbol{0}$, give $0 > \boldsymbol{y}^T\boldsymbol{b} = \boldsymbol{y}^T A\boldsymbol{u} + \boldsymbol{y}^T B\boldsymbol{v} \geq 0$ which is impossible. Thus we have established the Farkas alternative. □

#### 8.6.2.2 Proving the Karush–Kuhn–Tucker Conditions

To prove the Karush–Kuhn–Tucker conditions we need a constraint qualification to ensure that

$$\begin{aligned} T_\Omega(\boldsymbol{x}) = \big\{\, \boldsymbol{d} \mid \nabla g_i(\boldsymbol{x})^T\boldsymbol{d} = 0 \quad & \text{for all } i \in \mathcal{E}, \\ \nabla g_i(\boldsymbol{x})^T\boldsymbol{d} \geq 0 \quad & \text{for all } i \in \mathcal{I} \text{ where } g_i(\boldsymbol{x}) = 0 \,\big\} = C_\Omega(\boldsymbol{x}). \end{aligned} \tag{8.6.9}$$

A constraint $g_i(\boldsymbol{x}) \geq 0$ is called *active* at $\boldsymbol{x}$ if $g_i(\boldsymbol{x}) = 0$ and *inactive* at $\boldsymbol{x}$ if $g_i(\boldsymbol{x}) > 0$. Inactive inequality constraints at $\boldsymbol{x}$ do not affect the shape of the feasible set $\Omega$ near to $\boldsymbol{x}$. We designate the set of active constraints by

$$\mathcal{A}(\boldsymbol{x}) = \{\, i \mid i \in \mathcal{E} \cup \mathcal{I} \text{ and } g_i(\boldsymbol{x}) = 0 \,\}. \tag{8.6.10}$$

The equivalence (8.6.9) holds under a number of constraint qualifications, the most used of which is the Linear Independence Constraint Qualification (LICQ) for inequality constrained optimization:

$$\{\, \nabla g_i(\boldsymbol{x}) \mid i \in \mathcal{A}(\boldsymbol{x}) \,\} \quad \text{is a linearly independent set.} \tag{8.6.11}$$

Weaker constraint qualifications that guarantee (8.6.9) include the Mangasarian–Fromowitz constraint qualification (MFCQ):

$$\begin{aligned} &\{\, \nabla g_i(\boldsymbol{x}) \mid i \in \mathcal{E} \,\} \quad \text{is a linearly independent set, and} \\ &\quad \text{there is } \boldsymbol{d} \text{ where } \nabla g_i(\boldsymbol{x})^T\boldsymbol{d} = 0 \text{ for all } i \in \mathcal{E}, \text{ and} \\ &\qquad\qquad \nabla g_i(\boldsymbol{x})^T\boldsymbol{d} > 0 \text{ for all } i \in \mathcal{I} \cap \mathcal{A}(\boldsymbol{x}). \end{aligned} \tag{8.6.12}$$

With a suitable constraint qualification, we can prove the existence of Lagrange multipliers satisfying the Karush–Kuhn–Tucker conditions.

**Theorem 8.21** *Suppose (8.6.9) holds for all $\boldsymbol{x}$ in the feasible set*

$$\Omega = \{\, \boldsymbol{x} \mid g_i(\boldsymbol{x}) = 0 \ \text{for } i \in \mathcal{E},\ g_i(\boldsymbol{x}) \geq 0 \ \text{for } i \in \mathcal{I} \,\}.$$

*Then if $\boldsymbol{x}^*$ minimizes $f(\boldsymbol{x})$ subject to $\boldsymbol{x} \in \Omega$, there are Lagrange multipliers $\lambda_i$ for $i \in \mathcal{E} \cup \mathcal{I}$ where*

$$\mathbf{0} = \nabla f(\boldsymbol{x}^*) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i \nabla g_i(\boldsymbol{x}^*), \tag{8.6.13}$$

$$0 \leq \lambda_i,\ g_i(\boldsymbol{x}^*) \ \text{and} \ \lambda_i\, g_i(\boldsymbol{x}^*) = 0 \quad \text{for all } i \in \mathcal{I}, \tag{8.6.14}$$

$$\boldsymbol{x}^* \in \Omega. \tag{8.6.15}$$

***Proof*** Suppose $\boldsymbol{x}^*$ minimizes $f(\boldsymbol{x})$ subject to $\boldsymbol{x} \in \Omega$. Then by Lemma 8.19, there is no $\boldsymbol{d} \in T_\Omega(\boldsymbol{x}^*)$ where $\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} < 0$. By (8.6.9), there is no $\boldsymbol{d}$ where $\nabla f(\boldsymbol{x}^*)^T \boldsymbol{d} < 0$ and

$$\begin{aligned} 0 &= \nabla g_i(\boldsymbol{x}^*)^T \boldsymbol{d} \qquad \text{for all } i \in \mathcal{E}, \\ 0 &\leq \nabla g_i(\boldsymbol{x}^*)^T \boldsymbol{d} \qquad \text{for all } i \in \mathcal{I} \cap \mathcal{A}(\boldsymbol{x}^*). \end{aligned}$$

If we set $A = [\nabla g_i(\boldsymbol{x}^*) \mid i \in \mathcal{I} \cap \mathcal{A}(\boldsymbol{x}^*)]$ and $B = [\nabla g_i(\boldsymbol{x}^*) \mid i \in \mathcal{E}]$, from the Farkas alternative (Theorem 8.20), we must take the first alternative. That is, there must be $\boldsymbol{\mu} \geq \mathbf{0}$ and $\boldsymbol{\nu}$ where $\nabla f(\boldsymbol{x}^*) = A\boldsymbol{\mu} + B\boldsymbol{\nu}$. Let $\lambda_i = \mu_i$ for $i \in \mathcal{I} \cap \mathcal{A}(\boldsymbol{x}^*)$, and $\lambda_i = \nu_i$ if $i \in \mathcal{E}$. For inactive inequality constraints ($i \in \mathcal{I} \backslash \mathcal{A}(\boldsymbol{x}^*)$), we set $\lambda_i = 0$. Then we see that (8.6.13) holds:

$$\nabla f(\boldsymbol{x}^*) = \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i \nabla g_i(\boldsymbol{x}^*).$$

Also $\lambda_i \geq 0$ for all $i \in \mathcal{I}$. Since $\boldsymbol{x}^*$ is feasible, $g_i(\boldsymbol{x}^*) \geq 0$ for all $i \in \mathcal{I}$. Since $g_i(\boldsymbol{x}^*) > 0$ implies $i \notin \mathcal{A}(\boldsymbol{x}^*)$, we have $\lambda_i = 0$, so that $\lambda_i\, g_i(\boldsymbol{x}^*) = 0$. Thus (8.6.14) holds. Finally, (8.6.15) holds because $\boldsymbol{x}^*$ must be feasible to be a constrained minimizer. Thus all of the Karush–Kuhn–Tucker conditions (8.6.13–8.6.15) hold, as we wanted. □

### 8.6.2.3 Algorithms for Inequality Constrained Optimization

There is a wide range of algorithms for inequality constrained optimization.

Some algorithms add "slack variables" to turn inequality constrained optimization problems to create equivalent equality constrained problems: replace "$g_i(\boldsymbol{x}) \geq 0$" with "$g_i(\boldsymbol{x}) - s_i^2 = 0$" where $s_i$ is a new "slack" variable. The danger here is that if the MFCQ constraint qualification (8.6.12) holds, then the new equality constraints are likely to violate the LICQ for equality constrained optimization problems.

Some algorithms are set up like the interior point methods for linear programming: each inequality constraint $g_i(\boldsymbol{x}) \geq 0$ is turned into a penalty term, so that the objective function becomes

$$f(\boldsymbol{x}) - \alpha \sum_{i \in \mathcal{I}} \ln g_i(\boldsymbol{x}),$$

with $\alpha$ being reduced down toward zero in a stepwise fashion.

Some algorithms use linearizations of the constraints along with second order Taylor polynomials for the objective function:

$$\min_{\boldsymbol{p}} f(\boldsymbol{x}_k) + \nabla f(\boldsymbol{x}_k)^T \boldsymbol{p} + \frac{1}{2} \boldsymbol{p}^T \mathrm{Hess}_{\boldsymbol{x}} L(\boldsymbol{x}_k, \boldsymbol{\lambda}_k)\, \boldsymbol{p} \tag{8.6.16}$$

subject to

$$g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{p} = 0 \quad \text{for all } i \in \mathcal{E}, \tag{8.6.17}$$

$$g_i(\boldsymbol{x}_k) + \nabla g_i(\boldsymbol{x}_k)^T \boldsymbol{p} \geq 0 \quad \text{for all } i \in \mathcal{I}. \tag{8.6.18}$$

This is called the *Successive Quadratic Programming method* or *SQP method*. Difficulties can arise here if the linearized problem is infeasible. Each subproblem (8.6.16–8.6.18) is a *quadratic program*:

$$\min_{z} \boldsymbol{g}^T \boldsymbol{z} + \frac{1}{2} \boldsymbol{z}^T B \boldsymbol{z} \tag{8.6.19}$$

$$\text{subject to } A_{\mathcal{E}} \boldsymbol{z} = \boldsymbol{b}_{\mathcal{E}}, \tag{8.6.20}$$

$$A_{\mathcal{I}} \boldsymbol{z} \geq \boldsymbol{b}_{\mathcal{I}}. \tag{8.6.21}$$

Convex quadratic programs where $B$ is positive semi-definite can be solved in a finite process similar to the simplex method [95, 185]. It is possible that the constraints (8.6.20, 8.6.21) may be inconsistent, but the algorithms for quadratic programs can detect this. This can also be avoided by using non-smooth penalties instead of inequality constraints: instead of a constraint $\boldsymbol{a}^T \boldsymbol{z} \geq b$, we add a penalty $M \ \max(b - \boldsymbol{a}^T \boldsymbol{z},\ 0)$ with $M$ positive and large. Although this function is clearly nonlinear in $\boldsymbol{z}$, it can be represented by adding a slack variable $s$ satisfying the linear inequalities $s \geq 0$ and $s \geq b - \boldsymbol{a}^T \boldsymbol{z}$; the penalty term added to the objective is $M\, s$. This gives an alternative quadratic program

$$\begin{aligned} \min_{z} \boldsymbol{g}^T \boldsymbol{z} + \frac{1}{2} \boldsymbol{z}^T B \boldsymbol{z} + M \sum_{i \in \mathcal{E} \cup \mathcal{I}} s_i \\ \text{subject to } s_i \geq +(b_i - \boldsymbol{a}_i^T \boldsymbol{z}) \text{ for all } i \in \mathcal{E}, \\ s_i \geq -(b_i - \boldsymbol{a}_i^T \boldsymbol{z}) \text{ for all } i \in \mathcal{E}, \\ s_i \geq \quad b_i - \boldsymbol{a}_i^T \boldsymbol{z} \text{ for all } i \in \mathcal{I}, \end{aligned}$$

which always has a solution. The solution of this modified quadratic program is also equal to the solution of (8.6.19–8.6.21) provided the original quadratic program has a solution and $M$ is greater than the sum of the absolute values of the Lagrange multipliers.

SQP methods typically use a "merit function" to represent the quality of the solution for purposes of line searches or trust regions, such as

$$m_\alpha(\boldsymbol{x}) := f(\boldsymbol{x}) + \alpha \left[ \sum_{i \in \mathcal{E}} |g_i(\boldsymbol{x})| + \sum_{i \in \mathcal{I}} \max(-g_i(\boldsymbol{x}),\, 0) \right],$$

similar to the function $m_\alpha$ in Algorithm 82. SQP methods also need to take into account the curvature of the constraints by incorporating a correction step similar to that for equality constrained problems, to restore the constraints. This constraint correction step should be taken before the merit function is evaluated, to determine the step length.

**Exercises.**

(1) Show that (8.6.9) holds if all the constraint functions are affine: $g_i(\boldsymbol{x}) = \boldsymbol{a}_i^T \boldsymbol{x} + b_i$. This is the affine constraint qualification. Use this to show that Lagrange multipliers exist for linear programs: $\min_{\boldsymbol{x}} \boldsymbol{c}^T \boldsymbol{x}$ subject to $A\boldsymbol{x} \geq \boldsymbol{b}$ (componentwise).

(2) The paper of Kuhn and Tucker [149] gives an example where constraint qualifications fail, and no Lagrange multiplier exists: $\min x$ over all $(x, y) \in \mathbb{R}^2$ subject to $y \geq 0$ and $y \leq x^3$. Show that the minimizer is $(x, y) = (0, 0)$. Then show that no Lagrange multiplier exists at this point.

(3) As in the previous Exercise, $f(\boldsymbol{x}) = \boldsymbol{x}^T A \boldsymbol{x}$ with $A$ symmetric. Now consider the problem $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ subject to the condition that $\boldsymbol{x}^T \boldsymbol{x} \leq 1$. With the same Lagrangian, what property does the Lagrange multiplier have? Express your property in terms of the eigenvalues of $A$.

(4) Prove that the Mangasarian–Fromowitz constraint qualification (8.6.12) implies (8.6.9). [**Hint:** First show that if $\nabla g_i(\boldsymbol{x})^T \boldsymbol{p} > 0$ for all $i$ where $g_i(\boldsymbol{x}) = 0$ implies $\boldsymbol{p} \in T_\Omega(\boldsymbol{x})$. Now suppose that $\nabla g_i(\boldsymbol{x})^T \widehat{\boldsymbol{d}} > 0$ and $\nabla g_i(\boldsymbol{x})^T \boldsymbol{d} \geq 0$ for all $i$ where $g_i(\boldsymbol{x}) = 0$. Show, therefore, that $\nabla g_i(\boldsymbol{x})^T (\boldsymbol{d} + \epsilon \widehat{\boldsymbol{d}}) > 0$ if $g_i(\boldsymbol{x}) = 0$ for any $\epsilon > 0$, so $\boldsymbol{d} + \epsilon \widehat{\boldsymbol{d}} \in T_\Omega(\boldsymbol{x})$. Finish by using the fact that $T_\Omega(\boldsymbol{x})$ is closed to conclude that $\boldsymbol{d} \in T_\Omega(\boldsymbol{x})$.]

(5) Suppose that all the constraint functions are concave; that is, $-g_i$ is convex, for all $i$. *Slater's constraint qualification* is that there is $\widehat{\boldsymbol{x}}$ such that $g_i(\widehat{\boldsymbol{x}}) > 0$ for all $i$. Show that under these conditions, (8.6.9) holds. [**Hint:** Set $\widehat{\boldsymbol{d}} = \widehat{\boldsymbol{x}} - \boldsymbol{x}$. Now $0 < g_i(\widehat{\boldsymbol{x}}) \leq g_i(\boldsymbol{x}) + \nabla g_i(\boldsymbol{x})^T (\widehat{\boldsymbol{x}} - \boldsymbol{x})$ since $g_i$ is concave. Conclude that $\nabla g_i(\boldsymbol{x})^T \widehat{\boldsymbol{d}} > 0$ if $g_i(\boldsymbol{x}) = 0$. Now use the fact that (8.6.12) implies (8.6.9) (see the previous Exercise).]

(6) Find the global minimizer of $f(x, y) = x^4 + x^2 y + y^2 - x - y$ subject to $x^2 + y^2 = 1$. Note that there are multiple critical points. Symbolic solvers can find all solutions of the Lagrange multiplier equations.

(7) Inequality constraints can be turned into equality constraints with the help of a "slack" variable. Consider the problem $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) \geq 0$. This is equivalent to $\min_{x,s} f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) - s^2 = 0$. Use the first- and second-order necessary conditions for equality constrained version to obtain the KKT conditions (8.6.13–8.6.15).

(8) Solve the following constrained optimization problem:

$$\begin{aligned} \min_{x,y} \; & e^{-x} - xy + y^2 \qquad \text{subject to} \\ & x,\ y \geq 0, \\ & x + y \leq 2. \end{aligned}$$

Compute the Lagrange multipliers for your solution. Check that the KKT conditions hold at your solution. [**Note:** You may need to solve a nonlinear equation in one variable numerically.] Check the second-order conditions for your solution as well.

(9) Suppose that $\boldsymbol{x}^*$ is a KKT point for the minimization problem

$$\begin{aligned} \min_{\boldsymbol{x}} \; & f(\boldsymbol{x}) \qquad \text{subject to} \\ & g_i(\boldsymbol{x}) \geq 0 \qquad \text{for } i = 1, 2, \ldots, m. \end{aligned}$$

Show that if $f$ and $-g_i$ are convex functions for $i = 1, 2, \ldots, m$, then $\boldsymbol{x}^*$ is a global minimum for the constrained optimization problem. [**Hint:** Show that $f(\boldsymbol{x}) - \sum_{i=1}^{m} \lambda_i\, g_i(\boldsymbol{x})$ is a convex function of $\boldsymbol{x}$ where $\lambda_i$ are the Lagrange multipliers at $\boldsymbol{x}^*$.]

(10) In a sense, constrained optimization problems $\min_{\boldsymbol{x}} f(\boldsymbol{x})$ subject to $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0} \in \mathbb{R}^m$ have two objectives: to minimize $f(\boldsymbol{x})$, and to satisfy the constraints $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{0}$. One way of combine these two is in a merit function $m_\alpha(\boldsymbol{x}) = f(\boldsymbol{x}) + \alpha \sum_{i=1}^{m} |g_i(\boldsymbol{x})|$. Show that if $\boldsymbol{x}^*$ is a strict local constrained minimizer, then $\boldsymbol{x}^*$ is a local (unconstrained) minimizer of $F_\mu$ provided $\mu > \max_{i=1,\ldots,m} |\lambda_i|$ where $\boldsymbol{\lambda}$ is the Lagrange multiplier at $\boldsymbol{x}^*$.

(11) N. Maratos [170] showed that if a merit function like $m_\alpha$ of the previous Exercise is used, then a second- order correction or something similar should also be used. Consider the problem $\min_{x,y} x$ subject to $x^2 + y^2 = 1$, and $\alpha = 2$. Show that if $\boldsymbol{x}_k = (x_k, y_k) = (\cos\theta_k, \sin\theta_k)$ with $\theta_k \approx \pi$, then the Newton step $\boldsymbol{d}_k$ for the Lagrange multiplier and constraint equations, will *not* reduce the merit function: $m_\alpha(\boldsymbol{x}_k + \boldsymbol{d}_k) > m_\alpha(\boldsymbol{x}_k)$. From this, explain the justification for the second-order correction step.

(12) Consider the minimax problem: $\min_{\boldsymbol{x}} f(\boldsymbol{x}) + \max\{g_1(\boldsymbol{x}),\ g_2(\boldsymbol{x}),\ \ldots,\ g_m(\boldsymbol{x})\}$ with all functions $f$ and $g_i$ smooth. Show that this can be represented as a smoothly constrained optimization problem:

$$\min_{\boldsymbol{x},s} f(\boldsymbol{x}) + s \qquad \text{subject to}$$
$$s \geq g_i(\boldsymbol{x}), \qquad i = 1, 2, \ldots, m.$$

Show also that this formulation satisfies the Mangasarian–Fromowitz constraint qualification (8.6.12).

**Project**

An optimal control problem is a differential equation involving a "control" variable that can be changed at will over time to minimize a certain objective function, subject to constraints on the control function. See, for example, [35, 142, 162] for more information about optimal control theory in general. A standard form is:

$$\min_{\boldsymbol{u}(\cdot),\boldsymbol{x}(\cdot)} g(\boldsymbol{x}(T)) \qquad \text{subject to}$$
$$\frac{d\boldsymbol{x}}{dt}(t) = \boldsymbol{f}(t,\, \boldsymbol{x}(t),\, \boldsymbol{u}(t)), \qquad \boldsymbol{x}(t_0) = \boldsymbol{x}_0,$$
$$\boldsymbol{u}(t) \in U \qquad \text{for all } t.$$

The set $U$ should be a convex set. The differential equation can be discretized by, for example, Euler method (6.1.8): $\boldsymbol{x}_k \approx \boldsymbol{x}(t_k)$ with $t_k = t_0 + k\,h$ where

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + h\,\boldsymbol{f}(t_k,\, \boldsymbol{x}_k,\, \boldsymbol{u}_k), \qquad k = 0, 1, 2, \ldots, N-1. \tag{8.6.22}$$

Each of these constraints has a Lagrange multiplier $\boldsymbol{\lambda}_k$ and we define the Lagrangian

$$L(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) = g(\boldsymbol{x}_N) - \sum_{k=0}^{N-1} \boldsymbol{\lambda}_k^T \left[\boldsymbol{x}_{k+1} - \boldsymbol{x}_k - h\,\boldsymbol{f}(t_k, \boldsymbol{x}_k, \boldsymbol{u}_k)\right].$$

If we set the linearization of $L(\boldsymbol{x} + \delta\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda}) - L(\boldsymbol{x}, \boldsymbol{u}, \boldsymbol{\lambda})$ to zero and noting that $\delta\boldsymbol{x}_0 = \boldsymbol{0}$ as $\boldsymbol{x}_0$ fixed, we obtain the equations for the $\boldsymbol{\lambda}_k$:

$$\nabla g(\boldsymbol{x}_N) = \boldsymbol{\lambda}_{N-1}, \tag{8.6.23}$$
$$\boldsymbol{\lambda}_{k-1} = \boldsymbol{\lambda}_k + h\,\nabla_{\boldsymbol{x}} \boldsymbol{f}(t_k,\, \boldsymbol{x}_k,\, \boldsymbol{u}_k), \qquad k = 1, 2, \ldots, N-1. \tag{8.6.24}$$

The gradient of the objective function $g(\boldsymbol{x}_N)$ with respect to the $\boldsymbol{u}_k$ *constrained by (8.6.22)* is given by the linearized change

$$\delta\left[g(\boldsymbol{x}_N)\right] = h \sum_{k=0}^{N-1} \boldsymbol{\lambda}_k^T \nabla_{\boldsymbol{u}} \boldsymbol{f}(t_k,\, \boldsymbol{x}_k,\, \boldsymbol{u}_k)\, \delta\boldsymbol{u}_k.$$

To perform the optimization, we use a *gradient projection algorithm*: take a step in the negative gradient direction, then project that back to the feasible set, which in this case is $\boldsymbol{u}_k \in U$ for $k = 0, 1, 2, \ldots, N-1$:

(8.6.25)
$$\boldsymbol{u}_k^+ \leftarrow \text{proj}_U(\boldsymbol{u}_k - s\, h\, \nabla_{\boldsymbol{u}} \boldsymbol{f}(t_k,\, \boldsymbol{x}_k,\, \boldsymbol{u}_k)^T \boldsymbol{\lambda}_k), \qquad k = 0, 1, 2, \ldots, N-1,$$

where $\text{proj}_U(\boldsymbol{w})$ returns the nearest point in $U$ to $\boldsymbol{w}$. The step length $s$ can either be constant (chosen after some experimentation) or chosen according to a sufficient decrease criterion based on the objective function value *after* projection using $\boldsymbol{u}_k^+$. The inner parts of the algorithm are: (1) simulate the dynamics for $\boldsymbol{x}_k$ given $\boldsymbol{u}_k$ by (8.6.22); (2) compute the $\boldsymbol{\lambda}_k$ using (8.6.23, 8.6.24) going *backwards* in $k$; (3) perform the gradient projection step (8.6.25); and (4) accept or reject $\boldsymbol{u}_k^+$ (and update $s$) according to the sufficient decrease criterion (optional).

Use this to approximately solve the optimal harvesting problem: max $m(T)$ where

$$\begin{aligned} \frac{dm}{dt} &= \delta\, m + u(t)\, f, \qquad m(0) = 0, \\ \frac{df}{dt} &= a\, f - b\, f^2 - u(t)\, f, \qquad f(0) = a/b, \end{aligned}$$

subject to $u(t) \in [0,\, 0.20]$. Use $T = 100$, $a = 0.05$, $b = 0.02$, $\delta = 0.10$. In this, $f(t)$ is the quantity of fish in the sea, $m(t)$ is the amount of money in fishermen's bank accounts, $\delta$ is the interest rate at the bank, $a$ is the rate of reproduction of fish (at low populations), and $u(t)$ represents the fishing effort. Re-run with $\delta = 0.05$ and $a = 0.10$.

# Appendix A
# What You Need from Analysis

## A.1 Banach and Hilbert Spaces

### *A.1.1 Normed Spaces and Completeness*

A *vector space* is a collection $V$ of vectors $\boldsymbol{v}$ where there is vector addition $\boldsymbol{v} + \boldsymbol{w}$ and scalar multiplication $s\boldsymbol{v}$ for $s$ a real number ($s \in \mathbb{R}$ or possibly $\mathbb{C}$) satisfying the usual commutative, associative, and distributive properties:

$$\begin{aligned} \boldsymbol{u} + (\boldsymbol{v} + \boldsymbol{w}) &= (\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}, \\ \boldsymbol{u} + \boldsymbol{v} &= \boldsymbol{v} + \boldsymbol{u}, \\ r(s\boldsymbol{v}) &= (rs)\boldsymbol{v}, \\ (r + s)\boldsymbol{v} &= r\boldsymbol{v} + s\boldsymbol{v}, \\ r(\boldsymbol{u} + \boldsymbol{v}) &= r\boldsymbol{u} + r\boldsymbol{v}, \\ \boldsymbol{v} + \boldsymbol{0} &= \boldsymbol{v} = \boldsymbol{0} + \boldsymbol{v}, \\ 0\boldsymbol{v} &= \boldsymbol{0}, \qquad 1\boldsymbol{v} = \boldsymbol{v}. \end{aligned}$$

The simplest examples of interest to us are $\mathbb{R}^n$, the set of $n$-dimensional vectors, represented by columns of $n$ real numbers:

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

Addition is understood componentwise, scalar multiplication is applied to each entry, and the zero vector is the vector where each entry is zero. Functions $f : D \to \mathbb{R}$ also form a vector space with addition defined by $(f + g)(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$ and scalar



D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7

multiplication by $(s \cdot f)(\boldsymbol{x}) = s \cdot f(\boldsymbol{x})$. Matrices with $m$ rows and $n$ columns also form a vector space with entrywise addition and scalar multiplication.

A *normed vector space* is a vector space $V$ with a norm $\|\cdot\|$ with the usual properties of norms:

- $\|\boldsymbol{v}\| \geq 0$ and $\|\boldsymbol{v}\| = 0$ implies $\boldsymbol{v} = \boldsymbol{0}$,
- $\|s\boldsymbol{v}\| = |s| \, \|\boldsymbol{v}\|$, and
- $\|\boldsymbol{v} + \boldsymbol{w}\| \leq \|\boldsymbol{v}\| + \|\boldsymbol{w}\|$.

A sequence of vectors $\boldsymbol{v}_i$, $i = 1, 2, 3, \ldots$, is a *Cauchy sequence* if for every $\epsilon > 0$ there is an $N$ where $k, \ell \geq N$ implies $\|\boldsymbol{v}_k - \boldsymbol{v}_\ell\| < \epsilon$. We say $\boldsymbol{v}_i$ *converges to* $\boldsymbol{v}$ as $i \to \infty$ if for every $\epsilon > 0$ there is an $N$ where $k \geq N$ implies $\|\boldsymbol{v}_k - \boldsymbol{v}\| < \epsilon$, in which case we say $\boldsymbol{v}$ is the limit of the $\boldsymbol{v}_i$: $\lim_{i\to\infty} \boldsymbol{v}_i = \boldsymbol{v}$. We also denote convergence to a limit by $\boldsymbol{v}_i \to \boldsymbol{v}$ as $i \to \infty$.

The vector space $V$ is a *complete normed space* or *Banach space* if every Cauchy sequence converges. Not all normed vector spaces are complete, such as the polynomials over $[0, 1]$ with the norm $\|f\|_\infty = \max_{0\leq x\leq 1} |f(x)|$. The sequence $f_k(x) = \sum_{j=0}^{k} x^j/j!$ is a Cauchy sequence ($\|f_k - f_\ell\|_\infty \leq \sum_{j=\min(k,\ell)}^{\infty} 1/j! < 2/\min(k, \ell)!$) but does not converge to a polynomial; instead $f_k \to f$ where $f(x) = e^x$. Every normed vector space $V$ has a *completion* $\widetilde{V}$ which is a complete vector space, and consists of all possible limits of Cauchy sequences in $V$. If $V$ is already complete, then $\widetilde{V} = V$. Every finite dimensional normed space is already complete. Any two norms of the same finite dimensional vector space are equivalent in the sense of (1.5.2).

Note that $C(D)$, the space of continuous functions $D \to \mathbb{R}$ where $D$ is a closed and bounded set in $\mathbb{R}^d$, is a complete normed space with norm $\|f\|_\infty = \max_{\boldsymbol{x}\in D} |f(\boldsymbol{x})|$.

Most Banach spaces are best defined through completions. The spaces $L^p(D)$ for $1 \leq p < \infty$ and $D$ closed but bounded and positive volume in $\mathbb{R}^d$, are defined as the completion of $C(D)$ with respect to the norm

$$\|f\|_{L^p(D)} = \left[\int_D |f(\boldsymbol{x})|^p \, d\boldsymbol{x}\right]^{1/p}.$$

The space $L^\infty(D)$ is defined as the set of integrable functions on $D$ for which

$$\|f\|_{L^\infty(D)} = \inf\{\, s \mid \mathrm{vol}_d(\{\boldsymbol{x} \in D \mid |f(\boldsymbol{x})| > s\}) = 0\,\}.$$

The $W^{m,p}(D)$ norm for $1 \leq p < \infty$ is given by

$$\|f\|_{W^{m,p}(D)} = \left[\sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|\leq m} \int_D \left|\frac{\partial^{|\boldsymbol{\alpha}|} f}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x})\right|^p d\boldsymbol{x}\right]^{1/p}$$

for $1 \leq p < \infty$ for $f$ having continuous order $m$ derivatives. The space $W^{m,p}(D)$ is the completion of all functions with continuous order $m$ derivatives under this norm. For $p = \infty$, we have

$$\|f\|_{W^{m,\infty}(D)} = \max_{\alpha:|\alpha|\le m} \max_{x\in D} \left| \frac{\partial^{|\alpha|} f}{\partial x^{\alpha}}(x) \right|^{p} dx \tag{A.1.1}$$

for functions with continuous order $m$ derivatives. The space $W^{m,\infty}(D)$ is the set of all functions $f$ where for every multi-index $\alpha$ with $|\alpha| \le m$, $\partial^{|\alpha|} f/\partial x^{\alpha} \in L^{\infty}(D)$.

Functions $F: V \to W$ between normed spaces are *continuous* if $v_i \to v$ implies $F(v_i) \to F(v)$ as $i \to \infty$. If $F$ is also linear ($F(ru + sv) = r\,F(u) + s\,F(v)$ for all $u, v \in V$ and $r, s \in \mathbb{R}$) then $F$ is continuous if and only if $\sup\{\|F(v)\|_W \mid \|v\|_V \le 1\}$ is finite. In fact, there is the norm for linear functions $V \to W$ given by

$$\|F\|_{V\to W} = \sup\{\|F(v)\|_W \mid \|v\|_V \le 1\}. \tag{A.1.2}$$

### *A.1.2 Inner Products*

An *inner product space* $V$ is a vector space with a *real inner product* $(\cdot,\ \cdot): V \times V \to \mathbb{R}$ which for real scalars has the properties:

- $(v, v) \ge 0$ for all $v$ and $(v, v) = 0$ implies $v = \mathbf{0}$
- $(v,\ w) = (w,\ v)$ for all $v$ and $w$,
- $(u,\ rv + sw) = r(u,\ v) + s(u,\ w)$ for all $v$ and $w$, and scalars $r, s \in \mathbb{R}$.

Complex inner product spaces have an inner product with the following properties:

- $(v, v) \ge 0$ for all $v$ and $(v, v) = 0$ implies $v = \mathbf{0}$
- $(v,\ w) = \overline{(w,\ v)}$ for all $v$ and $w$,
- $(u,\ rv + sw) = r(u,\ v) + s(u,\ w)$ for all $v$ and $w$, and scalars $r, s \in \mathbb{C}$,

where $\overline{z}$ is the complex conjugate of $z \in \mathbb{C}$.

Standard examples include $\mathbb{R}^n$ with the inner product $(x, y) = x^T y = \sum_{j=1}^{n} x_j y_j$, and continuous functions $D \to \mathbb{R}$ where $D$ is a region in $\mathbb{R}^d$ with positive volume. Then we can define $(f, g) = \int_D f(x)\, g(x)\, dx$ as an inner product on continuous functions on $D$. For complex vectors we define $(x, y) = \overline{x}^T y = \sum_{j=1}^{n} \overline{x}_j y_j$; for complex functions we usually use $(f, g) = \int_D \overline{f(x)}\, g(x)\, dx$.

Inner products define a norm: $\|v\| = \sqrt{(v,\ v)}$. This is the inner product norm. If $D$ is a closed and bounded subset of $\mathbb{R}^d$, then $C(D)$, the space of continuous functions $D \to \mathbb{R}$, is an inner product space with $(f, g) = \int_D f(x)\, g(x)\, dx$. However, this space is *not* complete with respect to the inner product norm $\|f\|_2 = \sqrt{(f, f)} = \sqrt{\int_D f(x)^2\, dx}$. Instead the completion of this space with respect to this norm is denoted $L^2(D)$, the set of square-integrable functions $f: D \to \mathbb{R}$. Complete inner product spaces are called *Hilbert spaces*.

The *Cauchy–Schwarz inequality* applies to all inner product spaces:

$$|(a,\ b)| \le \|a\|\,\|b\| \tag{A.1.3}$$

where the norm $\|\boldsymbol{a}\| = \sqrt{(\boldsymbol{a},\ \boldsymbol{a})}$. In the case where $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ and $(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^T \boldsymbol{b}$ we have

$$\left| \sum_{i=1}^{n} a_i b_i \right| \leq \left[ \sum_{i=1}^{n} a_i^2 \right]^{1/2} \left[ \sum_{i=1}^{n} b_i^2 \right]^{1/2}; \tag{A.1.4}$$

for $f, g \in L^2(D)$ there is the integral version:

$$\left| \int_D f(\boldsymbol{x})\, g(\boldsymbol{x})\, d\boldsymbol{x} \right| \leq \left[ \int_D f(\boldsymbol{x})^2\, d\boldsymbol{x} \right]^{1/2} \left[ \int_D g(\boldsymbol{x})^2\, d\boldsymbol{x} \right]^{1/2}. \tag{A.1.5}$$

### *A.1.3 Dual Spaces and Weak Convergence*

A *linear functional* $\ell$ is a continuous linear function $\ell\colon V \to \mathbb{R}$ where $V$ is a normed space. The norm of a linear functional is

$$\|\ell\| = \sup \{\, |\ell(\boldsymbol{v})| \mid \|\boldsymbol{v}\|_V \leq 1 \,\}.$$

With this norm, the linear functionals on $V$ is another normed space $V'$, which is called the *dual* of $V$. If $V$ is a Banach space, so is $V'$. For $V = \mathbb{R}^n$, we can identify the dual space $(\mathbb{R}^n)'$ with $\mathbb{R}^n$; we can think of vectors $\boldsymbol{x}$ in $\mathbb{R}^n$ as column vectors, and vectors $\boldsymbol{y}$ in $(\mathbb{R}^n)'$ as row vectors so that $\boldsymbol{y}(\boldsymbol{x}) = \boldsymbol{y}\,\boldsymbol{x}$ using regular matrix operations to multiply a row vector by a column vector. We can identify $\boldsymbol{x} \in \mathbb{R}^n$ with $\boldsymbol{x}^T \in (\mathbb{R}^n)'$. For complex vectors, we identify $\boldsymbol{x} \in \mathbb{C}^n$ with $\overline{\boldsymbol{x}}^T \in (\mathbb{C}^n)'$.

Note that if $V$ is a real inner product space, $\boldsymbol{w} \mapsto (\boldsymbol{v},\ \boldsymbol{w})$ is a linear functional on $V$. If $V$ is a complete inner product space (that is, Hilbert space), then every linear functional on $V$ can be represented in this way.

We say that $\boldsymbol{v}_i$ *converges weakly to* $\boldsymbol{v}$ as $i \to \infty$ (denoted $\boldsymbol{v}_i \rightharpoonup \boldsymbol{v}$ as $i \to \infty$) if $\ell(\boldsymbol{v}_i) \to \ell(\boldsymbol{v})$ as $i \to \infty$ for every linear functional $\ell$. We say that a sequence of linear functionals $\ell_j$ *converges weakly* to* $\ell$ as $i \to \infty$ (denoted $\ell_j \rightharpoonup^* \ell$ as $j \to \infty$) if $\ell_j(\boldsymbol{v}) \to \ell(\boldsymbol{v})$ as $j \to \infty$ for every $\boldsymbol{v} \in V$.

If $A\colon V \to W$ is a continuous linear function between Banach spaces, then the *adjoint* to $A$, denoted $A^*\colon W' \to V'$, is defined by $A^*(\ell)(\boldsymbol{v}) = \ell(A\boldsymbol{v})$ for $\boldsymbol{v} \in V$ where $\ell$ is a linear functional on $W$. If $A$ is represented by an $m \times n$ matrix (for $A\colon \mathbb{R}^n \to \mathbb{R}^m$) then identifying the dual space $(\mathbb{R}^n)'$ with $\mathbb{R}^n$, we have $A^* = A^T$. For $A\colon \mathbb{C}^n \to \mathbb{C}^m$ then $A^* = \overline{A}^T$. Note that if $A$ is a bounded linear function, so is $A^*$:

$$\begin{aligned}\left\|A^*\right\|_{W'\to V'} &= \sup\left\{\left\|A^*(\ell)\right\|_{V'} \mid \|\ell\|_{V'} \le 1\right\}\\ &= \sup\left\{\left|A^*(\ell)(\boldsymbol{v})\right| \mid \|\ell\|_{V'} \le 1,\ \|\boldsymbol{v}\|_V \le 1\right\}\\ &= \sup\{\,|\ell(A\boldsymbol{v})| \mid \|\ell\|_{V'} \le 1,\ \|\boldsymbol{v}\|_V \le 1\,\}\\ &\le \sup\{\,\|\ell\|_{V'}\,\|A\|_{V\to W}\,\|\boldsymbol{v}\|_V \mid \|\ell\|_{V'} \le 1,\ \|\boldsymbol{v}\|_V \le 1\,\}\\ &\le \|A\|_{V\to W}\,.\end{aligned}$$

In fact, $\|A^*\|_{W'\to V'} = \|A\|_{V\to W}$.

There is a linear function $J: V \to V''$ ($V''$ is the dual of the dual of $V$) given by $J(\boldsymbol{v})(\ell) = \ell(\boldsymbol{v})$ where $\ell \in V'$. The Banach space $V$ is *reflexive* if range$(J) = V''$. Note that both $\mathbb{R}^n$ and $\mathbb{C}^n$ are reflexive, and we identify $(\mathbb{R}^n)''$ with $\mathbb{R}^n$ by $\left(\boldsymbol{x}^T\right)^T = \boldsymbol{x}$ and $(\mathbb{C}^n)''$ with $\mathbb{C}^n$ by $\overline{\overline{\boldsymbol{x}}^T}^T = \boldsymbol{x}$.

In reflexive spaces, $V$ and $V''$ can be identified, and weak and weak* convergence are identical. The $W^{m,p}(D)$ spaces are reflexive for $1 < p < \infty$. The space $C(D)$ of continuous functions $D \to \mathbb{R}$ is *not* reflexive. In fact, the dual space to $C(D)$ can be identified with the space of finite signed Borel measures $\mu$ with functionals being represented via integrals with respect to $\mu$:

$$\ell(f) = \int_D f(\boldsymbol{x})\,\mu(d\boldsymbol{x}). \tag{A.1.6}$$

The Dirac "$\delta$-function" is actually a measure concentrated at zero with the linear functional:

$$f \mapsto f(\mathbf{0}) = \int_{\mathbb{R}^d} f(\boldsymbol{x})\,\delta(\boldsymbol{x})\,d\boldsymbol{x}.$$

If $W$ is a subspace of $V$ then $W^\perp$ is the set of all $\ell \in V'$ where $\ell(\boldsymbol{w}) = 0$ for all $\boldsymbol{w} \in W$. If $V$ is an inner product space then we can identify $V$ and $V'$ through the inner product, and

$$W^\perp = \{\,\boldsymbol{v} \in V \mid (\boldsymbol{v}, \boldsymbol{w}) = 0 \text{ for all } \boldsymbol{w} \in W\,\}$$

is the orthogonal complement of $W$ in $V$.

## A.2 Distributions and Fourier Transforms

### *A.2.1 Distributions and Measures*

Distributions extend the idea of Dirac "$\delta$-functions" and measures, which are given meaning as linear functionals on spaces of continuous functions:

$$\varphi \mapsto \int_{\mathbb{R}^d} \varphi(\boldsymbol{x})\, \mu(d\boldsymbol{x}).$$

The Dirac $\delta$-function is the measure that represents the functional $\varphi \mapsto \varphi(\mathbf{0})$. Distributions generalize the idea of measure as follows: distributions are dual spaces to spaces of "nice" functions. More specifically, they are continuous linear functionals on the space of $\varphi$ with the following properties:

- derivatives $\partial^{|\boldsymbol{\alpha}|}\varphi/\partial \boldsymbol{x}^{\boldsymbol{\alpha}}$ exist everywhere for every multi-index $\boldsymbol{\alpha}$;
- the set $\left\{ \boldsymbol{x} \in \mathbb{R}^d \mid \varphi(\boldsymbol{x}) \neq 0 \right\}$ is a bounded set.

This is the space of *test functions* $\mathcal{S}(\mathbb{R}^d)$. This is a vector space of functions. We do not give this space a single norm, but instead an infinite family of norms:

$$\|\varphi\|_{C^r(\mathbb{R}^d)} = \max_{\boldsymbol{x}\in\mathbb{R}^d} \max_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|\leq r} \left| \frac{\partial^{|\boldsymbol{\alpha}|}\varphi}{\partial \boldsymbol{x}^{\boldsymbol{\alpha}}}(\boldsymbol{x}) \right|, \qquad r = 1, 2, 3, \ldots. \tag{A.2.1}$$

Continuous functions on $\mathcal{S}(\mathbb{R}^d)$ can depend not only just values of the function but also values of derivatives, and integrals of derivatives of whatever order we wish, as long as no functional uses an unbounded number of derivatives. *Distributions* are continuous linear functionals on $\mathcal{S}(\mathbb{R}^d)$, denoted $\mathcal{S}(\mathbb{R}^d)'$. As for $\mathcal{S}(\mathbb{R}^d)$, there is no norm for $\mathcal{S}(\mathbb{R}^d)'$. A conventional function $f : \mathbb{R} \to \mathbb{R}$ that is integrable is represented by the distribution given by

$$\varphi \in \mathcal{S}(\mathbb{R}^d) \mapsto \int_{\mathbb{R}^d} f(\boldsymbol{x})\, \varphi(\boldsymbol{x})\, d\boldsymbol{x}. \tag{A.2.2}$$

Derivatives can be defined on distributions, even where the corresponding functions are not differentiable via integration by parts: in one dimension,

$$\begin{aligned} \int_{-\infty}^{+\infty} f'(x)\, \varphi(x)\, dx &= f(x)\varphi(x)|_{x=-\infty}^{x=+\infty} - \int_{-\infty}^{+\infty} f(x)\, \varphi'(x)\, dx \\ &= -\int_{-\infty}^{+\infty} f(x)\, \varphi'(x)\, dx. \end{aligned}$$

We use the notation $\ell[\varphi]$ to represent the application of a distribution (as a linear functional) to a test function $\varphi \in \mathcal{S}(\mathbb{R}^d)$, to distinguish it from ordinary function evaluation. Thus if $\ell \in \mathcal{S}(\mathbb{R})'$ is a linear functional representing a function $f$, then the functional of the derivative $f'$ is

$$\ell'(\varphi) = -\ell(\varphi'),$$

which is well defined because for any $\varphi \in \mathcal{S}(\mathbb{R})$, $\varphi' \in \mathcal{S}(\mathbb{R})$ as well. For $\ell \in \mathcal{S}(\mathbb{R}^d)'$ we define the linear functional $\partial\ell/\partial x_j$ as

$$\frac{\partial \ell}{\partial x_j}(\varphi) = -\ell(\frac{\partial \varphi}{\partial x_j}).$$

Higher derivatives can be treated in the same way. The main issue with these definitions and manipulations is interpreting what the resulting derivative is. For example, while the *Heaviside function*

$$H(x) = \begin{cases} 1, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0 \end{cases}$$

is a regular function for which the linear functional

$$H[\varphi] = \int_{-\infty}^{+\infty} H(x)\,\varphi(x)\,dx = \int_0^{\infty} \varphi(x)\,dx$$

is well defined for every test function $\varphi \in \mathcal{S}(\mathbb{R})$, the derivative is given by

$$H'[\varphi] = -H[\varphi'] = -\int_0^{\infty} \varphi'(x) = -(\varphi(\infty) - \varphi(0)) = \varphi(0),$$

which matches the Dirac $\delta$-function: $\delta[\varphi] = \varphi(0)$. That is, $H' = \delta$. Thus $H'$ is a measure. On the other hand, the derivative of the Dirac $\delta$-function

$$\delta'[\varphi] = -\delta[\varphi'] = -\varphi'(0)$$

is *not* a measure as it is not a continuous functional on functions that are only continuous.

### *A.2.2 Fourier Transforms*

The Fourier transform of a function $f : \mathbb{R} \to \mathbb{R}$ or $\mathbb{R} \to \mathbb{C}$ is given by

$$\mathcal{F}f(\xi) = \int_{-\infty}^{+\infty} e^{-i\xi x}\, f(x)\,dx, \tag{A.2.3}$$

or for functions on $\mathbb{R}^d$,

$$\mathcal{F}f(\boldsymbol{\xi}) = \int_{\mathbb{R}^d} e^{-i\boldsymbol{\xi}^T \boldsymbol{x}}\, f(\boldsymbol{x})\,d\boldsymbol{x}. \tag{A.2.4}$$

This is defined as long as $\int_{\mathbb{R}^d} |f(\boldsymbol{x})|\,d\boldsymbol{x}$ is finite, although the definition can be extended to functions where this is not true. We can do this in the spirit of distributions,

but we need to modify the kind of test functions we work with: a *tempered test function* is a function $\varphi$ where

- derivatives $\partial^{|\alpha|}\varphi/\partial \boldsymbol{x}^{\boldsymbol{\alpha}}$ exist everywhere for every multi-index $\boldsymbol{\alpha}$;
- $\boldsymbol{x}^{\boldsymbol{\beta}}\,\partial^{|\alpha|}\varphi/\partial \boldsymbol{x}^{\boldsymbol{\alpha}}(\boldsymbol{x}) \to 0$ as $\|\boldsymbol{x}\| \to \infty$ for all multi-indexes $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$.

Such functions constitute the space of tempered test functions denoted $\mathcal{T}(\mathbb{R}^d)$. It can be shown fairly easily that $\mathcal{F}\colon \mathcal{T}(\mathbb{R}^d) \to \mathcal{T}(\mathbb{R}^d)$. Furthermore, we can prove a number of properties of the Fourier transform that hold for every $\varphi \in \mathcal{T}(\mathbb{R}^d)$, and many other functions besides:

$$\begin{aligned}
\mathcal{F}\left[\frac{\partial \varphi}{\partial x_j}\right](\boldsymbol{\xi}) &= i\xi_j\,\mathcal{F}\varphi(\boldsymbol{\xi}),\\
\mathcal{F}\left[\boldsymbol{x} \mapsto x_j\varphi(\boldsymbol{x})\right](\boldsymbol{\xi}) &= i\,\frac{\partial}{\partial \xi_j}\mathcal{F}\varphi(\boldsymbol{\xi}),\\
\mathcal{F}\left[\varphi(\cdot - \boldsymbol{a})\right](\boldsymbol{\xi}) &= e^{-i\boldsymbol{a}^T\boldsymbol{\xi}}\mathcal{F}\varphi(\boldsymbol{\xi}),\\
\mathcal{F}\left[\boldsymbol{x} \mapsto \varphi(\alpha\boldsymbol{x})\right](\boldsymbol{\xi}) &= \alpha^{-d}\mathcal{F}\varphi(\boldsymbol{\xi}/\alpha).
\end{aligned}$$

The most important formula is the Fourier inversion formula: if $g = \mathcal{F}f$, then

$$f(\boldsymbol{x}) = (2\pi)^{-d}\int_{\mathbb{R}^d} e^{+i\boldsymbol{x}^T\boldsymbol{\xi}}g(\boldsymbol{\xi})\,d\boldsymbol{\xi}. \tag{A.2.5}$$

This can be expressed more succinctly as $\mathcal{F}^{-1} = (2\pi)^{-d}\,\mathcal{F}^*$ where $\mathcal{F}^*$ is the adjoint of $\mathcal{F}$ with respect to the standard complex inner product $(f, g) = \int_{\mathbb{R}^d} \overline{f(\boldsymbol{x})}\,g(\boldsymbol{x})\,d\boldsymbol{x}$. This means that

$$(2\pi)^{-d}\,(\mathcal{F}f,\ \mathcal{F}g) = (2\pi)^{-d}\,(f,\ \mathcal{F}^*\mathcal{F}g) = (f,\ \mathcal{F}^{-1}\mathcal{F}g) = (f,\ g). \tag{A.2.6}$$

The way of applying the distribution trick to Fourier transforms starts by noting that for an integrable, bounded function $f$ and a tempered test function $\varphi$,

$$\begin{aligned}
f[\mathcal{F}\varphi] &= \int_{\mathbb{R}^d} f(\boldsymbol{\xi})\,\mathcal{F}\varphi(\boldsymbol{\xi})\,d\boldsymbol{\xi} = \int_{\mathbb{R}^d} f(\boldsymbol{\xi})\int_{\mathbb{R}^d} e^{-i\boldsymbol{x}^T\boldsymbol{\xi}}\varphi(\boldsymbol{x})\,d\boldsymbol{x}\,d\boldsymbol{\xi}\\
&= \int_{\mathbb{R}^d} \varphi(\boldsymbol{x})\int_{\mathbb{R}^d} e^{-i\boldsymbol{x}^T\boldsymbol{\xi}f(\boldsymbol{\xi})}\,d\boldsymbol{\xi}\,d\boldsymbol{x} \quad \text{(changing order of integration)}\\
&= \int_{\mathbb{R}^d} \varphi(\boldsymbol{x})\,\mathcal{F}f(\boldsymbol{x})\,d\boldsymbol{x} = \mathcal{F}f[\varphi].
\end{aligned}$$

We extend this definition from functions like $f$ to *all* linear functionals $\ell$ on $\mathcal{T}(\mathbb{R}^d)$:

$$\mathcal{F}\ell[\varphi] = \ell[\mathcal{F}\varphi].$$

This means we can compute Fourier transforms of $\delta$ functions: $\mathcal{F}\delta(\xi) = 1$, and other exotic things such as the comb function: $\mathrm{III}(x) = \sum_{k\in\mathbb{Z}} \delta(x-k)$: $\mathcal{F}\mathrm{III}(\xi) = (2\pi)^{1/2}\mathrm{III}(\xi/(2\pi)) = (2\pi)^{-1/2}\sum_{k\in\mathbb{Z}} \delta(\xi - 2\pi k)$. Applied to test functions $\varphi$ we get

$$(2\pi)^{-1/2}\sum_{k\in\mathbb{Z}} \varphi(2\pi k) = \mathcal{F}\mathrm{III}[\varphi] = \mathrm{III}[\mathcal{F}\varphi] = \sum_{k\in\mathbb{Z}} \mathcal{F}\varphi(k),$$

which is essentially the *Poisson summation formula*.

Some other properties of Fourier transforms relate to convolutions:

$$f * g(\boldsymbol{x}) = \int_{\mathbb{R}^d} f(\boldsymbol{y})\, g(\boldsymbol{x}-\boldsymbol{y})\, d\boldsymbol{y}. \tag{A.2.7}$$

The most important of these is the fact that the Fourier transform of a convolution of two functions is the ordinary product of the Fourier transforms:

$$\mathcal{F}[f * g](\boldsymbol{\xi}) = \mathcal{F}f(\boldsymbol{\xi}) \cdot \mathcal{F}g(\boldsymbol{\xi}). \tag{A.2.8}$$

A similar property holds for the Fourier transform of an ordinary product of two functions:

$$\mathcal{F}[f \cdot g] = (2\pi)^{-d}\, \mathcal{F}f * \mathcal{F}g. \tag{A.2.9}$$

## A.3 Sobolev Spaces

Sobolev spaces are families of Banach spaces of functions $\Omega \to \mathbb{R}$ where $\Omega$ is a suitable set in $\mathbb{R}^d$. This section is a brief summary. More details can be found in [11, 31, 213, 245] and other sources. These spaces include all smooth functions $\Omega \to \mathbb{R}$. The Sobolev $W^{m,p}(\Omega)$ norms, with $\Omega$ an open subset of $\mathbb{R}^d$, are defined first for $m$ a non-negative integer and $1 \le p \le \infty$:

$$\|f\|_{W^{m,p}(\Omega)} = \left[\int_\Omega \sum_{j=0}^{m} \left\|D^j f(\boldsymbol{x})\right\|^p \, d\boldsymbol{x}\right]^{1/p} \quad \text{for } 1 \le p < \infty, \text{ and} \tag{A.3.1}$$

$$\|f\|_{W^{m,\infty}(\Omega)} = \sup_{\boldsymbol{x}\in\Omega} \sum_{j=0}^{m} \left\|D^j f(\boldsymbol{x})\right\|. \tag{A.3.2}$$

Once the norm is defined for all smooth functions $\Omega \to \mathbb{R}$, the *space* $W^{m,p}(\Omega)$ is defined as the *completion* of the smooth functions $\Omega \to \mathbb{R}$ with respect to the $W^{m,p}(\Omega)$ norm. The space $H^m(\Omega)$ is defined to be $W^{m,2}(\Omega)$, which is a Hilbert space. This completion can be understood in an abstract sense in terms of equivalence classes of Cauchy sequences, or as functions in $L^p(\Omega)$ with distributional derivatives

of order up to $m$ in $L^p(\Omega)$. Functions in $W^{m,p}(\Omega)$ can be extended across $\mathbb{R}^d$ by a linear extension operator $\mathcal{E}\colon W^{m,p}(\Omega) \to W^{m,p}(\mathbb{R}^d)$ provided $\Omega$ has a boundary where the boundary can be locally represented as the graph of a Lipschitz function [237] (see Figure 4.3.9 for an illustration). With the extension to $\mathbb{R}^d$, it is possible to replace the $W^{m,p}(\mathbb{R}^d)$ norm with an equivalent weighted integral of the Fourier transform if $p = 2$:

$$\|f\|_{H^m(\mathbb{R}^d)} = (2\pi)^{-d/2} \left[ \int_{\mathbb{R}^d} \left( \sum_{j=0}^{m} \|\boldsymbol{\xi}\|_2^2 \right) |\mathcal{F}f(\boldsymbol{\xi})|^2 \, d\boldsymbol{\xi} \right]^{1/2}.$$

Fractional order Sobolev spaces can also be defined through Sobolev–Slobodetskii norms

$$\|f\|_{W^{s,p}(\Omega)} = \left[ \|f\|_{W^{m,p}(\Omega)}^p + \sum_{\boldsymbol{\alpha}:|\boldsymbol{\alpha}|=m} \int_\Omega \int_\Omega \frac{\left| \partial^{|\boldsymbol{\alpha}|} f/\partial \boldsymbol{x}^{\boldsymbol{\alpha}}(\boldsymbol{x}) - \partial^{|\boldsymbol{\alpha}|} f/\partial \boldsymbol{x}^{\boldsymbol{\alpha}}(\boldsymbol{y}) \right|^p}{\|\boldsymbol{x} - \boldsymbol{y}\|^{p\sigma+d}} d\boldsymbol{x}\, d\boldsymbol{y} \right]^{1/p}$$

where $m = \lfloor s \rfloor$, the floor of $s$, and $\sigma = s - \lfloor s \rfloor$. If $p = 2$ and $\Omega = \mathbb{R}^d$, equivalent norms can be given in terms of Fourier transforms:

$$\|f\|_{H^s(\mathbb{R}^d)} = (2\pi)^{-d/2} \left[ \int_{\mathbb{R}^d} \left(1 + \|\boldsymbol{\xi}\|_2^2\right)^s |\mathcal{F}f(\boldsymbol{\xi})|^2 \, d\boldsymbol{\xi} \right]^{1/2}.$$

As might be expected, increasing order of derivatives involved or level of smoothness means stronger norms and smaller spaces: if $r < s$ then $W^{s,p}(\Omega) \subset W^{r,p}(\Omega)$. Furthermore, this embedding is compact: bounded sequences in $W^{s,p}(\Omega)$ get mapped to sequences in $W^{r,p}(\Omega)$ that have convergent subsequences. This can be important for showing the existence of solutions.

In partial differential equations, fractional order Sobolev spaces are important for dealing with boundary values. More specifically, if the region $\Omega$ has a smooth boundary, then the restriction of a function in $W^{s,p}(\Omega)$ to the boundary $\partial\Omega$ is in $W^{s-1/p,p}(\partial\Omega)$. In general, for a $k$-dimensional submanifold $M$ in $\Omega$, the restriction of a function $f \in W^{s,p}(\Omega)$ to $M$ is in $W^{s-k/p,p}(M)$. If $s - k/p \leq 0$, this restriction is typically not even defined. In the extreme case, if $M$ is a single point in $\Omega$, then the restriction of $f \in W^{s,p}(\Omega)$ to $M$ is only defined if $s > d/p$. In the case where $p = \infty$, then $W^{m,\infty}(\Omega)$ has restrictions to $W^{m,\infty}(M)$ for any $k$-dimensional submanifold $M$.

The restriction operator $\gamma\colon W^{s,p}(\Omega) \to W^{s-1/p,p}(\partial\Omega)$ is called the *trace operator* on $W^{s,p}(\Omega)$.

Even more important for solving partial differential equations is that every function $g \in W^{s-1/p,p}(\partial\Omega)$ can be extended to a function $\widetilde{g} \in W^{s,p}(\Omega)$ so that $\gamma\widetilde{g} = g$.

# References


1. A. Agrawal, P. Klein, R. Ravi, Cutting down on fill using nested dissection: provably good elimination orderings, *Graph Theory and Sparse Matrix Computation*. IMA Volumes in Mathematics and Its Applications, vol. 56 (Springer, New York, 1993), pp. 31–55
2. L.V. Ahlfors, *Complex Analysis*. International Series in Pure and Applied Mathematics, 3rd edn. (McGraw-Hill Book Co., New York, 1978). An introduction to the theory of analytic functions of one complex variable
3. R. Alexander, Diagonally implicit Runge-Kutta methods for stiff O.D.E.'s. SIAM J. Numer. Anal. **14**(6), 1006–1021 (1977)
4. E.L. Allgower, K. Georg, *Introduction to Numerical Continuation Methods*. Classics in Applied Mathematics, vol. 45 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2003). Reprint of the 1990 edn. (Springer, Berlin; MR1059455 (92a:65165))
5. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen, *LAPACK Users' Guide*, 3rd edn. (SIAM, Philadelphia, 1999)
6. I.A. Antonov, V.M. Saleev, An effective method for the computation of $\lambda P_\tau$-sequences. Zh. Vychisl. Mat. i Mat. Fiz. **19**(1), 243–245, 271 (1979)
7. N. Apostolatos, U. Kulisch, R. Krawczyk, B. Lortz, K. Nickel, H.W. Wippermann, The algorithmic language Triplex-ALGOL 60. Numer. Math. **11**(2), 175–180 (1968)
8. L. Armijo, Minimization of functions having Lipschitz continuous first partial derivatives. Pac. J. Math. **16**, 1–3 (1966)
9. W.E. Arnoldi, The principle of minimized iterations in the solution of the matrix eigenvalue problem. Q. Appl. Math. **9**, 17–29 (1951)
10. U.M. Ascher, L.R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations* (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1998)
11. K. Atkinson, W. Han, *Theoretical Numerical Analysis*. Texts in Applied Mathematics, vol. 39, 3rd edn. (Springer, Dordrecht, 2009). A functional analysis framework
12. K. Atkinson, W. Han, *Theoretical Numerical Analysis*. Texts in Applied Mathematics, vol. 39, 3rd edn. (Springer, Dordrecht, 2009). A functional analysis framework
13. K.E. Atkinson, *An Introduction to Numerical Analysis*, 2nd edn. (Wiley, New York, 1989)
14. J. Backus, *The History of Fortran I, II, and III* (Association for Computing Machinery, New York, 1978), pp. 25–74
15. L. Badger, Lazzarini's lucky approximation of $\pi$. Math. Mag. **67**(2), 83–91 (1994)
16. J.L. Barlow, A. Smoktunowicz, Reorthogonalized block classical Gram-Schmidt. Numer. Math. **123**(3), 395–423 (2013)




D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7

17. F. Bashforth, J.C. Adams, *An Attempt to Test the Theories of Capillary Action by Comparing the Theoretical and Measured Forms of Drops of Fluid* (University Press, 1883)
18. A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey (2018), arXiv:1907.07587
19. R. Bellman, The stability of solutions of linear differential equations. Duke Math. J. **10**, 643–647 (1943)
20. A. Berman, R.J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*. Classics in Applied Mathematics, vol. 9 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1994). Revised reprint of the 1979 original
21. J.-P. Berrut, L.N. Trefethen, Barycentric Lagrange interpolation. SIAM Rev. **46**(3), 501–517 (2004)
22. I. Bihari, A generalization of a lemma of Bellman and its application to uniqueness problems of differential equations. Acta Math. Acad. Sci. Hung. **7**, 81–94 (1956)
23. Å. Björck, Solving linear least squares problems by Gram-Schmidt orthogonalization. Nord. Tidskr. Informationsbehandling (BIT) **7**, 1–21 (1967)
24. G. Blanchet, B. Dupouy, G. Blanchet, *Computer Architecture* (Wiley, New York, 2012)
25. A.N. Borodin, *Stochastic Processes*. Probability and Its Applications (Springer International Publishing, Cham, 2017)
26. L.P. Bos, Bounding the Lebesgue function for Lagrange interpolation in a simplex. J. Approx. Theory **38**(1), 43–59 (1983)
27. G.E.P. Box, M.E. Muller, A note on the generation of random normal deviates. Ann. Math. Stat. **29**(2), 610–611 (1958)
28. J.H. Bramble, P.H. Sammon, Efficient higher order single step methods for parabolic problems. I. Math. Comput. **35**(151), 655–677 (1980)
29. P. Bratley, B.L. Fox, Algorithm 659: implementing Sobol's quasirandom sequence generator. ACM Trans. Math. Softw. **14**(1), 88–100 (1988)
30. K.E. Brenan, S.L. Campbell, L.R. Petzold, *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Classics in Applied Mathematics, vol. 14 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1996). Revised and corrected reprint of the 1989 original
31. S.C. Brenner, L.R. Scott, *The Mathematical Theory of Finite Element Methods*. Texts in Applied Mathematics, vol. 15, 3rd edn. (Springer, New York, 2008)
32. R.P. Brent, *Algorithms for Minimization Without Derivatives*. Prentice-Hall Series in Automatic Computation (Prentice-Hall, Inc., Englewood Cliffs, 1973)
33. W.L. Briggs, V.E. Henson, S.F. McCormick, *A Multigrid Tutorial*, 2nd edn. (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2000)
34. L.D. Broemeling, An account of early statistical inference in Arab cryptology. Am. Stat. **65**(4), 255–257 (2011)
35. A.E. Bryson Jr., Y.C. Ho, *Applied Optimal Control* (Hemisphere Publishing Corporation, Washington); distributed by (Halsted Press (Wiley), New York-London-Sydney, 1975). Optimization, estimation, and control, Revised printing
36. G.L.L. Buffon, *Histoire naturelle, générale et particulière, servant de suite à la Théorie de la Terre et d'introduction à l'Histoire des Minéraux...Supplément Tome premier [septième]*, Number v. 4 (de l'imprimerie royale, 1777)
37. M.D. Buhmann, *Radial Basis Functions: Theory and Implementations*. Cambridge Monographs on Applied and Computational Mathematics, vol. 12 (Cambridge University Press, Cambridge, 2003)
38. J. Bunch, L. Kaufman, B. Parlett, Decomposition of a symmetic matrix. Numer. Math. **27**, 95–109 (1976)
39. J.R. Bunch, L. Kaufman, Some stable methods for calculating inertia and solving symmetric linear systems. Math. Comput. **31**(137), 163–179 (1977)
40. K. Burrage, P.M. Burrage, High strong order explicit Runge-Kutta methods for stochastic ordinary differential equations. Appl. Numer. Math. **22**(1), 81–101 (1996). Special Issue Celebrating the centenary of Runge-Kutta methods

41. K. Burrage, J.C. Butcher, Stability criteria for implicit Runge-Kutta methods. SIAM J. Numer. Anal. **16**(1), 46–57 (1979)
42. K. Burrage, E. Platen, Runge-Kutta methods for stochastic differential equations, *Scientific Computation and Differential Equations (Auckland, 1993)*, vol. 1 (1994), pp. 63–78
43. P. Bussotti, On the genesis of the Lagrange multipliers. J. Optim. Theory Appl. **117**(3), 453–459 (2003)
44. J.C. Butcher, Coefficients for the study of Runge-Kutta integration processes. J. Aust. Math. Soc. **3**(2), 185–201 (1963)
45. J.C. Butcher, Implicit Runge-Kutta processes. Math. Comput. **18**, 50–64 (1964)
46. J.C. Butcher, *Numerical Methods for Ordinary Differential Equations*, 3rd edn. (Wiley, Chichester, 2016). With a foreword by J.M. Sanz-Serna
47. R.E. Caflisch, Monte Carlo and quasi-Monte Carlo methods, *Acta Numerica, 1998*, vol. 7 (Cambridge University Press, Cambridge, 1998), pp. 1–49
48. G. Cardano, *Liber de ludo aleae ("Book on Games of Chance")* (1663). Written around 1564, but published posthumously
49. R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonald, R. Menon, *Parallel Programming in OpenMP* (Morgan Kaufmann Publishers Inc., San Francisco, 2001)
50. S.-N. Chow, J. Mallet-Paret, J.A. Yorke, Finding zeros of maps: homotopy methods that are constructive with probability one. Math. Comput. **32**, 887–899 (1978)
51. C.W. Clenshaw, A.R. Curtis, A method for numerical integration on an automatic computer. Numer. Math. **2**, 197–205 (1960)
52. C.W. Clenshaw, F.W.J. Olver, Beyond floating point. J. Assoc. Comput. Mach. **31**(2), 319–328 (1984)
53. C.W. Clenshaw, F.W.J. Olver, P.R. Turner, Level-index arithmetic: an introductory survey, *Numerical Analysis and Parallel Processing (Lancaster, 1987)*. Lecture Notes in Mathematics, vol. 1397 (Springer, Berlin, 1989), pp. 95–168
54. A.R. Conn, N.I.M. Gould, P.L. Toint, *Trust-Region Methods*. MPS/SIAM Series on Optimization (Society for Industrial and Applied Mathematics (SIAM), Philadelphia; Mathematical Programming Society (MPS), Philadelphia, 2000)
55. R. Cook, *An Introduction to Parallel Programming with OpenMP, PThreads and MPI*, 2nd edn. (Cook's Books, 2011)
56. J.W. Cooley, J.W. Tukey, An algorithm for the machine calculation of complex Fourier series. Math. Comput. **19**, 297–301 (1965)
57. R. Cools, Monomial cubature rules since "Stroud": a compilation. II. J. Comput. Appl. Math. **112**(1–2), 21–27 (1999). Numerical evaluation of integrals
58. R. Cools, An encyclopaedia of cubature formulas. J. Complex. **19**(3), 445–453 (2003). Numerical integration and its complexity (Oberwolfach, 2001), The encyclopedia itself is at http://nines.cs.kuleuven.be/ecf/
59. T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduction to Algorithms*, 3rd edn. (MIT Press, Cambridge, 2009)
60. R. Courant, K. Friedrichs, H. Lewy, Über die partiellen Differenzengleichungen der mathematischen Physik. Math. Ann. **100**(1), 32–74 (1928)
61. G.R. Cowper, Gaussian quadrature formulas for triangles. Int. J. Numer. Methods Eng. **7**(3), 405–408 (1973)
62. M. Crouzeix, P.A. Raviart, *Approximation des problèmes d'évolution*. Unpublished Lecture Notes (Université de Rennes, 1980)
63. M. Crouzeix, Sur la $B$-stabilité des méthodes de Runge-Kutta. Numer. Math. **32**(1), 75–82 (1979)
64. J.K. Cullum, R.A. Willoughby, *Lánczos Algorithms for Large Symmetric Eigenvalue Computations. Vol. I* (Birkhäuser Boston Inc., Boston, 1985). Theory
65. J.K. Cullum, R.A. Willoughby, *Lánczos Algorithms for Large Symmetric Eigenvalue Computations. Vol. II* (Birkhäuser Boston Inc., Boston, 1985). Programs
66. E. Cuthill, J. McKee, Reducing the bandwidth of sparse symmetric matrices, in *Proceedings of the 1969 24th National Conference, ACM'69* (ACM, New York, 1969), pp. 157–172

67. G.G. Dahlquist, A special stability problem for linear multistep methods. Nord. Tidskr. Informationsbehandling (BIT) **3**, 27–43 (1963)
68. G.C. Danielson, C. Lanczos, Some improvements in practical Fourier analysis and their application to X-ray scattering from liquids. J. Frankl. Inst. **233**, 365–380, 435–452 (1942)
69. G.B. Dantzig, Reminiscences about the origins of linear programming. Oper. Res. Lett. **1**(2), 43–48 (1982)
70. M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, *Computational Geometry*, 3rd edn. (Springer, Berlin, 2008). Algorithms and applications
71. C. de Boor, *A Practical Guide to Splines*. Applied Mathematical Sciences, vol. 27, Revised edn. (Springer, New York, 2001)
72. A.P. de Camargo, An exponential lower bound for the condition number of real Vandermonde matrices. Appl. Numer. Math. **128**, 81–83 (2018)
73. P. de Fermat, Methodus ad disquirendam maximam et minimam (Method for the study of maxima and minima) (1636)
74. T.J. Dekker, Finding a zero by means of successive linear interpolation, in *Constructive Aspects of the Fundamental Theorem of Algebra (Proceedings of Symposium, Zürich-Rüschlikon, 1967)* (Wiley-Interscience, New York, 1969), pp. 37–48
75. J.W. Demmel, S.C. Eisenstat, J.R. Gilbert, X.S. Li, J.W.H. Liu, A supernodal approach to sparse partial pivoting. SIAM J. Matrix Anal. Appl. **20**(3), 720–755 (1999)
76. J.W. Demmel, A numerical analyst's Jordan canonical form. Ph.D. thesis, University of California, Berkeley (1983). Advisor: W. Kahan
77. E.J. Doedel, AUTO: software for continuation and bifurcation problems in ordinary differential equations. Technical report, California Institute of Technology (1986). Software available from E. Doedel on request; email address: `keqfe82` at `vax2.concordia.ca`
78. E. Doedel, AUTO: a program for the automatic bifurcation analysis of autonomous systems. Congr. Numer. **30**, 265–284 (1981)
79. J.J. Dongarra, J. Du Croz, S. Hammarling, I.S. Duff, A set of level 3 basic linear algebra subprograms. ACM Trans. Math. Softw. **16**(1), 1–17 (1990)
80. J. Dongarra, G.H. Golub, E. Grosse, C. Moler, K. Moore, Netlib and NA-Net: building a scientific computing community. IEEE Ann. Hist. Comput. **30**(2), 30–41 (2008)
81. J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson, An extended set of Fortran basic linear algebra subprograms. ACM Trans. Math. Softw. **14**(1), 1–17 (1988)
82. J.J. Dongarra, C.B. Moler, J.R. Bunch, G.W. Stewart, *LINPACK Users' Guide* (SIAM, Philadelphia, 1979)
83. D.A. Dunavant, High degree efficient symmetrical Gaussian quadrature rules for the triangle. Int. J. Numer. Methods Eng. **21**(6), 1129–1148 (1985)
84. A. Elahi, T. Arjeski, *ARM Assembly Language with Hardware Experiments* (Springer International Publishing, Cham, 2015)
85. K. Entacher, Bad subsequences of well-known linear congruential pseudorandom number generators. ACM Trans. Model. Comput. Simul. **8**(1), 61–70 (1998)
86. J.F. Epperson, On the Runge example. Am. Math. Mon. **94**(4), 329–341 (1987), http://links.jstor.org/sici?sici=0002-9890%28198704%2994%3A4%3C329%3AOTRE%3E2.0.CO%3B2-T
87. R. Farber, *CUDA Application Design and Development* (Elsevier Science & Technology, Amsterdam, 2011)
88. G.E. Fasshauer, *Meshfree Approximation Methods with MATLAB*. Interdisciplinary Mathematical Sciences, vol. 6 (World Scientific Publishing Co. Pte. Ltd., Hackensack, 2007). With 1 CD-ROM (Windows, Macintosh and UNIX)
89. E. Fehlberg, New high-order Runge-Kutta formulas with step size control for systems of first- and second-order differential equations. Z. Angew. Math. Mech. **44**, T17–T29 (1964)
90. L. Fejér, Mechanische Quadraturen mit positiven Cotesschen Zahlen. Math. Z. **37**(1), 287–309 (1933)
91. L. Fejér, Untersuchungen über fouriersche reihen. Math. Ann. **58**, 51–69 (1904)

92. A. Figalli, A simple proof of the Morse-Sard theorem in Sobolev spaces. Proc. Am. Math. Soc. **136**(10), 3675–3681 (2008)
93. J.B.J. Fourier, *The Analytical Theory of Heat*. Cambridge Library Collection - Mathematics (Cambridge University Press, Cambridge, 2009). English translation of *Théorie Analytique de la Chaleur*, originally published 1822. Translated by Alexander Freeman
94. J.G.F. Francis, The $QR$ transformation: a unitary analogue to the $LR$ transformation. I. Comput. J. **4**, 265–271 (1961/1962)
95. J.N. Franklin, *Methods of Mathematical Economics*. Classics in Applied Mathematics, vol. 37 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2002). Linear and nonlinear programming, fixed-point theorems, Reprint of the 1980 original
96. D.C. Fraser, Newton's interpolation formulas. J. Inst. Actuar. **60**, 77–106 and 211–232 (1918–1919)
97. R.W. Freund, A transpose-free quasi-minimal residual algorithm for non-Hermitian linear systems. SIAM J. Sci. Comput. **14**(2), 470–482 (1993)
98. R.W. Freund, N.M. Nachtigal, QMR: a quasi-minimal residual method for non-Hermitian linear systems. Numer. Math. **60**(3), 315–339 (1991)
99. M.J. Gander, G. Wanner, From Euler, Ritz, and Galerkin to modern computing. SIAM Rev. **54**(4), 627–666 (2012)
100. C.W. Gear, Algorithm 407: DIFSUB for solution of ordinary differential equations [d2]. Commun. ACM **14**, 185–190 (1971)
101. A. George, J. Liu, *Computer Solution of Large, Sparse, Positive-Definite Systems*. Series in Computational Mathematics (Prentice-Hall, Englewood Cliffs, 1981)
102. D. Goldberg, What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv. **23**(1), 5–48 (1991)
103. A.A. Goldstein, On steepest descent. J. SIAM Control Ser. A **3**, 147–151 (1965)
104. G. Golub, W. Kahan, Calculating the singular values and pseudo-inverse of a matrix. J. Soc. Ind. Appl. Math. Ser. B Numer. Anal. **2**, 205–224 (1965)
105. G.H. Golub, C.F. Van Loan, *Matrix Computations*. Johns Hopkins Studies in the Mathematical Sciences, 3rd edn. (Johns Hopkins University Press, Baltimore, 1996)
106. G.H. Golub, J.H. Welsch, Calculation of Gauss quadrature rules. Math. Comput. **23**, 221–230 (1969); Addendum, ibid. **23**(106, loose microfiche suppl), A1–A10 (1969)
107. B. Gough, *GNU Scientific Library Reference Manual*, 3rd edn. (Network Theory Ltd., 2009)
108. J.P. Gram, Ueber die entwickelung reeller funtionen in reihen mittelst der methode der kleinsten quadrate. J. Reine Angew. Math. (1883)
109. J.F. Grcar, John von Neumann's analysis of Gaussian elimination and the origins of modern numerical analysis. SIAM Rev. **53**(4), 607–682 (2011)
110. D.A. Grier, *When Computers Were Human* (Princeton University Press, Princeton, 2005)
111. A. Griewank, A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Other Titles in Applied Mathematics, vol. 105, 2nd edn. (SIAM, Philadelphia, 2008)
112. T.H. Gronwall, Note on the derivatives with respect to a parameter of the solutions of a system of differential equations. Ann. Math. (2) **20**(4), 292–296 (1919)
113. W. Gropp, E. Lusk, A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 3rd edn. (MIT Press, Cambridge, 2014)
114. V. Guillemin, A. Pollack, *Differential Topology* (AMS Chelsea Publishing, Providence, 2010). Reprint of the 1974 original
115. J.L. Gustafson, *The End of Error*. Chapman & Hall/CRC Computational Science Series (CRC Press, Boca Raton, 2015). Unum computing
116. W.W. Hager, Updating the inverse of a matrix. SIAM Rev. **31**(2), 221–239 (1989)
117. E. Hairer, G. Wanner, *Solving Ordinary Differential Equations. II*. Springer Series in Computational Mathematics, vol. 14 (Springer, Berlin, 2010). Stiff and differential-algebraic problems, Second revised edn., Paperback
118. E. Hairer, C. Lubich, M. Roche, *The Numerical Solution of Differential-Algebraic Systems by Runge-Kutta Methods*. Lecture Notes in Mathematics, vol. 1409 (Springer, Berlin, 1989)

119. W.K. Hastings, Monte Carlo sampling methods using Markov chains and their applications. Biometrika **57**(1), 97–109 (1970)
120. L.G. Hačijan, A polynomial algorithm in linear programming. Dokl. Akad. Nauk SSSR **244**(5), 1093–1096 (1979)
121. O. Heaviside, *Electrical Papers (2 Volumes, Collected Works)* (The Electrician Printing and Publishing Co., London, 1892)
122. M.R. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems. J. Res. Natl. Bur. Stand. **49**, 409–436 (1953), (1952)
123. K. Heun, Neue methoden zur approximativen integration der differential-gleichungen einer unabhängigen veränderlichen. Z. Math. Phys. **45**, 23–38 (1900)
124. Y. Hida, X.S. Li, D.H. Bailey, Algorithms for quad-double precision floating point arithmetic, in *Proceedings of the 15th IEEE Symposium on Computer Arithmetic, ARITH'01, USA, 2001* (IEEE Computer Society, 2001), p. 155
125. N.J. Higham, *Accuracy and Stability of Numerical Algorithms* (SIAM, Philadelphia, 1996)
126. A.C. Hindmarsh, ODE solvers for use with the method of lines. Technical report, Lawrence Livermore National Lab., CA (USA) (1981)
127. A.C. Hindmarsh, P.N. Brown, K.E. Grant, S.L. Lee, R. Serban, D.E. Shumaker, C.S. Woodward, SUNDIALS: suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Softw. **31**(3), 363–396 (2005)
128. D.G. Hough, The IEEE standard 754: one for the history books. Computer **52**(12), 109–112 (2019)
129. T.E. Hull, A.R. Dobell, Random number generators. SIAM Rev. **4**, 230–254 (1962)
130. C. Huygens, De Ratiociniis in Ludo Aleae (On reasoning in games and chance) (1657)
131. M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V.B. Shah, W. Tebbutt, A differentiable programming system to bridge machine learning and scientific computing (2019), arXiv:1907.07587
132. Intel Corp. Intel Math Kernel Library One API, https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/onemkl.html. Accessed 22 Apr 2021
133. D. Jackson, *The Theory of Approximation* (Colloquium Publications. American Mathematical Society, New York, 1930)
134. N. Jacobson, *Basic Algebra. I*, 2nd edn. (W. H. Freeman and Company, New York, 1985)
135. L. Jay, Convergence of Runge-Kutta methods for differential-algebraic systems of index 3. Appl. Numer. Math. **17**(2), 97–118 (1995)
136. S. Kaniel, Estimates for some computational techniques in linear algebra. Math. Comput. **20**, 369–378 (1966)
137. L.V. Kantorovič, Functional analysis and applied mathematics. Usp. Mat. Nauk (N.S.) **3**(6(28)), 89–185 (1948)
138. N. Karmarkar, A new polynomial-time algorithm for linear programming. Combinatorica **4**(4), 373–395 (1984)
139. W. Karush, Minima of functions of several variables with inequalities as side constraints. Master's thesis, Department of Mathematics, University of Chicago, Chicago, Illinois (1939)
140. R.B. Kearfott, M. Dawande, K. Du, C. Hu, Algorithm 737: INTLIB: a portable Fortran-77 elementary function library. ACM Trans. Math. Softw. **20**(4), 447–459 (1994)
141. P. Keast, Moderate-degree tetrahedral quadrature formulas. Comput. Methods Appl. Mech. Eng. **55**(3), 339–348 (1986)
142. D.E. Kirk, *Optimal Control Theory: an Introduction*. Dover Books on Electrical Engineering (Dover Publications, 2012). Reprint of 1970 original
143. P.E. Kloeden, E. Platen, Higher-order implicit strong numerical schemes for stochastic differential equations. J. Stat. Phys. **66**(1–2), 283–314 (1992)
144. P.E. Kloeden, E. Platen, *Numerical Solution of Stochastic Differential Equations*. Applications of Mathematics (New York), vol. 23 (Springer, Berlin, 1992)
145. D.E. Knuth, Deciphering a linear congruential encryption. IEEE Trans. Inf. Theory **31**(1), 49–52 (1985)
146. M.J. Kochenderfer, T.A. Wheeler, *Algorithms for Optimization* (MIT Press, Cambridge, 2019)

147. A.N. Kolmogorov, *Foundations of the Theory of Probability* (Chelsea Publishing Company, New York, 1950). Translation of *Grundbegriffe der Wahrscheinlichkeitsrechnung* (1933)
148. V.N. Kublanovskaja, Some algorithms for the solution of the complete problem of eigenvalues. Ž. Vyčisl. Mat i Mat. Fiz. **1**, 555–570 (1961)
149. H.W. Kuhn, A.W. Tucker, Nonlinear programming, in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, 1950, Berkeley and Los Angeles* (University of California Press, 1951), pp. 481–492
150. M.W. Kutta, Beitrag zur näherunsweisen Integration oder Differentialgleichungen. Z. Math. Phys. **46**, 435–453 (1901)
151. J.-L. Lagrange, *Mécanique Analytique* (1788–1789). Two-volume set
152. C. Lanczos, An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. J. Res. Natl. Bur. Stand. **45**, 255–282 (1950)
153. J. Langou, Translation and modern interpretation of Laplace's théorie analytique des probabilités (2009), pp. 505–512, 516–520
154. P.S. Laplace, *Théorie analytique des probabilités* (V. Courcier, Paris, 1812)
155. P.-S. Laplace, *Theorie Analytique des Probabilités*, 3rd edn. (Courcier, Paris, 1820)
156. J. LaSalle, Uniqueness theorems and successive approximations. Ann. Math. (2) **50**, 722–730 (1949)
157. C.L. Lawson, R.J. Hanson, D.R. Kincaid, F.T. Krogh, Basic linear algebra subprograms for Fortran usage. ACM Trans. Math. Softw. **5**(3), 308–323 (1979)
158. P.D. Lax, *Functional Analysis*. Pure and Applied Mathematics (New York) (Wiley-Interscience (Wiley), New York, 2002)
159. V. Lefevre, J.-M. Muller, A. Tisserand, Toward correctly rounded transcendentals. IEEE Trans. Comput. **47**(11), 1235–1243 (1998)
160. R.B. Lehoucq, D.C. Sorensen, Deflation techniques for an implicitly restarted Arnoldi iteration. SIAM J. Matrix Anal. Appl. **17**(4), 789–821 (1996)
161. P.P. Lévy, *Théorie de l'addition des variables aléatoires* (Gauthier-Villars, Paris, 1937)
162. D. Liberzon, *Calculus of Variations and Optimal Control Theory* (Princeton University Press, Princeton, 2012). A concise introduction
163. S. Linnainmaa, Taylor expansion of the accumulated rounding error. BIT **16**(2), 146–160 (1976)
164. R.J. Lipton, R.E. Tarjan, A separator theorem for planar graphs. SIAM J. Appl. Math. **36**(2), 177–189 (1979)
165. W.H. Liu, A.H. Sherman, Comparative analysis of the Cuthill-McKee and the reverse Cuthill-McKee ordering algorithms for sparse matrices. SIAM J. Numer. Anal. **13**(2), 198–213 (1976)
166. E.N. Lorenz, Deterministic nonperiodic flow. J. Atmos. Sci. **20**(2), 130–141 (1963)
167. J.N. Lyness, R. Cools, A survey of numerical cubature over triangles, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics (Vancouver, BC, 1993)*. Proceedings of Symposia in Applied Mathematics, vol. 48 (American Mathematical Society, Providence, 1994), pp. 127–150
168. J.N. Lyness, D. Jespersen, Moderate degree symmetric quadrature rules for the triangle. J. Inst. Math. Appl. **15**, 19–32 (1975)
169. M.S. Mahoney, *The Mathematical Career of Pierre de Fermat, 1601–1665*. Princeton Paperbacks, 2nd edn. (Princeton University Press, Princeton, 1994)
170. N. Maratos, Exact penalty function algorithms for finite dimensional and control optimization problems. Ph.D. thesis, University of London (1978)
171. G. Marsaglia, Random numbers fall mainly in the planes. Proc. Natl. Acad. Sci. USA **61**, 25–28 (1968)
172. G. Marsaglia, W.W. Tsang, The ziggurat method for generating random variables. J. Stat. Softw. Artic. **5**(8), 1–7 (2000)
173. M. Matsumoto, Mersenne twister with improved initialization (2011), http://www.math.sci.hiroshima-u.ac.jp/m-mat/MT/MT2002/emt19937ar.html. Accessed 5 July 2021
174. M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. ACM Trans. Model. Comput. Simul. **8**(1), 3–30 (1998)

175. T.G. Mattson, Y. (Helen) He, A.E. Koniges, *The OpenMP Common Core: Making OpenMP Simple Again* (MIT Press, Cambridge, 2019)
176. M. Mazzetti, Burglars who took on F.B.I. abandon shadows. New York Times (2014)
177. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, E. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. **21**(6), 1087–1092 (1953)
178. B. Mityagin, The zero set of a real analytic function (2015)
179. G.E. Moore, Cramming more components onto integrated circuits. Electronics **38**(8) (1965)
180. R.E. Moore, R.B. Kearfott, M.J. Cloud, *Introduction to Interval Analysis* (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2009)
181. W.J. Morokoff, R.E. Caflisch, Quasi-random sequences and their discrepancies. SIAM J. Sci. Comput. **15**(6), 1251–1279 (1994)
182. D.D. Morrison, Remarks on the unitary triangularization of a nonsymmetric matrix. J. Assoc. Comput. Mach. **7**, 185–186 (1960)
183. F.R. Moulton, *New Methods in Exterior Ballistics* (University of Chicago Press, Chicago, 1926)
184. J.R. Munkres, *Topology* (Prentice Hall, Inc., Upper Saddle River, 2000). Second edition of [MR0464128]
185. K.G. Murty, *Linear Complementarity, Linear and Nonlinear Programming*. Sigma Series in Applied Mathematics, vol. 3 (Heldermann Verlag, Berlin, 1988)
186. R.D. Neidinger, Introduction to automatic differentiation and MATLAB object-oriented programming. SIAM Rev. **52**(3), 545–563 (2010)
187. Y. Nesterov, A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming*. SIAM Studies in Applied Mathematics, vol. 13 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1994)
188. L.M. Nguyen, P.H. Nguyen, P. Richtárik, K. Scheinberg, M. Takáč, M. van Dijk, New convergence aspects of stochastic gradient algorithms. J. Mach. Learn. Res. **20**(176), 1–49 (2019)
189. H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 63 (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1992)
190. J. Nocedal, S.J. Wright, *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering, 2nd edn. (Springer, New York, 2006)
191. B. Oksendal, *Stochastic Differential Equations*, 6th edn. (Springer, Berlin, 2019)
192. S. Oliveira, D.E. Stewart, *Writing Scientific Software: a Guide to Good Style* (Cambridge University Press, Cambridge, 2006)
193. M.E. O'Neill, PCG: a family of simple fast space-efficient statistically good algorithms for random number generation. Technical report HMC-CS-2014-0905, Harvey Mudd College, Claremont, CA (2014)
194. M.L. Overton, *Numerical Computing with IEEE Floating Point Arithmetic* (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2001). Including one theorem, one rule of thumb, and one hundred and one exercises
195. C.C. Paige, M.A. Saunders, LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**, 43–71 (1982)
196. C.C. Paige, The computation of eigenvalues and eigenvectors of very large sparse matrices. Ph.D. thesis, London University (1971)
197. F. Panneton, P. L'ecuyer, M. Matsumoto, Improved long-period generators based on linear recurrences modulo 2. ACM Trans. Math. Softw. **32**(1), 1–16 (2006)
198. B.N. Parlett, D.R. Taylor, Z.A. Liu, A look-ahead Lánczos algorithm for unsymmetric matrices. Math. Comput. **44**(169), 105–124 (1985)
199. D.A. Patterson, J.L. Hennessy, *Computer Organization and Design RISC-V Edition: the Hardware Software Interface*, 1st edn. (Morgan Kaufmann Publishers Inc., San Francisco, 2017)
200. G. Peano, Sull'integrabilità delle equazioni differenziali del primo ordine. Atti Accad. Sci. Torino **21**, 437–445 (1886)
201. P.-O. Persson, G. Strang, A simple mesh generator in Matlab. SIAM Rev. **46**(2), 329–345 (electronic) (2004)

202. L.R. Petzold, A description of DASSL: a differential/algebraic system solver, *Scientific Computing (Montreal, Quebec, 1982)*. IMACS Transactions on Scientific Computation, I (IMACS, New Brunswick, 1983), pp. 65–68
203. M. Pincus, A Monte Carlo method for the approximate solution of certain types of constrained optimization problems. Oper. Res. **18**, 1225–1228 (1970)
204. J. Pitt-Francis, J. Whiteley, *Guide to Scientific Computing in C++* (Springer International Publishing, Cham, 2017)
205. R.B. Platte, L.N. Trefethen, Chebfun: a new kind of numerical computing, *Progress in Industrial Mathematics at ECMI 2008*. Mathematics in Industry, vol. 15 (Springer, Heidelberg, 2010), pp. 69–87
206. G. Pólya, Über die Konvergenz von Quadraturverfahren. Math. Z. **37**(1), 264–286 (1933)
207. M.J.D. Powell, Some convergence properties of the conjugate gradient method. Math. Program. **11**(1), 42–49 (1976)
208. M.J.D. Powell, Nonconvex minimization calculations and the conjugate gradient method, *Numerical Analysis (Dundee, 1983)*. Lecture Notes in Mathematics, vol. 1066 (Springer, Berlin, 1984), pp. 122–141
209. A. Prothero, A. Robinson, On the stability and accuracy of one-step methods for solving stiff systems of ordinary differential equations. Math. Comput. **28**, 145–162 (1974)
210. J. Radon, Zur mechanischen Kubatur. Monatsh. Math. **52**, 286–300 (1948)
211. A. Ralston, P. Rabinowitz, *A First Course in Numerical Analysis*, 2nd edn. (Dover Publications Inc., Mineola, 2001)
212. E.Y. Remez, Sur le calcul effectiv des polynômes d'approximation des tschebyscheff. Comptes Rendus Acad. Sci. **199**, 337 (1934)
213. M. Renardy, R.C. Rogers, *An Introduction to Partial Differential Equations*. Texts in Applied Mathematics, vol. 13, 2nd edn. (Springer, New York, 2004)
214. T.J. Rivlin, *An Introduction to the Approximation of Functions* (Dover Publications, Inc., New York, 1981). Corrected reprint of the 1969 original, Dover Books on Advanced Mathematics
215. W. Romberg, Vereinfachte numerische Integration. Nor. Vid. Selsk. Forh., Trondheim **28**, 30–36 (1955)
216. K.H. Rosen, *Elementary Number Theory and Its Applications* (Addison-Wesley, Boston, 2011)
217. M. Rosen, *Number Theory in Function Fields (Graduate Texts in Mathematics)* (Springer, Berlin, 2002)
218. J.F. Ross, Pascal's legacy. EMBO Rep. **5**(S1) (2004)
219. K.F. Roth, On irregularities of distribution. Mathematika **1**, 73–79 (1954)
220. W. Rudin, *Real and Complex Analysis*, 3rd edn. (McGraw-Hill Book Co., New York, 1987)
221. W. Rümelin, Numerical treatment of stochastic differential equations. SIAM J. Numer. Anal. **19**(3), 604–613 (1982)
222. C. Runge, Über die numerische Auflösung von differentialgleichungen. Math. Ann. **46**(2), 167–178 (1895)
223. C. Runge, Ueber empirische Funktionen und die Interpolation zwischen aequidistanent Ordinaten. Z. Math. Phys. **46**, 224–243 (1901)
224. Y. Saad, M. Schultz, GMRES: a generalized minimal residual algorithm for solving nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. **7**, 856–869 (1986)
225. Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd edn. (Society for Industrial and Applied Mathematics, Philadelphia, 2003)
226. P. Salamon, P. Sibani, R. Frost, *Facts, Conjectures, and Improvements for Simulated Annealing*. SIAM Monographs on Mathematical Modeling and Computation (Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2002)
227. A. Sard, The measure of the critical values of differentiable maps. Bull. Am. Math. Soc. **48**, 883–890 (1942)
228. T. Sauer, *Numerical Analysis*, 2nd edn. (Addison-Wesley Publishing Company, Boston, 2011)
229. S. Schaefer, J. Hakenberg, J. Warren, Smooth subdivision of tetrahedral meshes, in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, SGP'04, New York, NY, USA* (Association for Computing Machinery, 2004), pp. 147–154

230. E. Schmidt, Zur theorie der linearen und nichtlinearen integralgleichungen. i. teil: Entwicklung willkurlicher funktionen nach systemen vorgeschriebener. Math. Ann. (1907)
231. A. Schönhage, Fehlerfortpflanzung bei Interpolation. Numer. Math. **3**, 62–71 (1961)
232. L.F. Shampine, M.W. Reichelt, The MATLAB ODE suite. SIAM J. Sci. Comput. **18**(1), 1–22 (1997). Dedicated to C. William Gear on the occasion of his 60th birthday
233. M.L. Shetterly, *Hidden Figures: the American Dream and the Untold Story of the Black Women Mathematicians Who Helped Win the Space Race*, 1st edn. (William Morrow, New York, 2016)
234. I.H. Sloan, H. Woźniakowski, When are quasi-Monte Carlo algorithms efficient for high-dimensional integrals? J. Complex. **14**(1), 1–33 (1998)
235. I.M. Sobol′, Distribution of points in a cube and approximate evaluation of integrals. Ž. Vyčisl. Mat i Mat. Fiz. **7**, 784–802 (1967)
236. P. Sonneveld, CGS: a fast Lanczos-type solver for non-symmetric linear systems. SIAM J. Sci. Stat. Comput. **10**, 36–52 (1989)
237. E.M. Stein, *Singular Integrals and Differentiability Properties of Functions*. Princeton Mathematical Series, vol. 30 (Princeton University Press, Princeton, 1970)
238. D.E. Stewart, A graph-theoretic model of symmetric Givens operations and its implications. Linear Algebra Appl. **257**, 311–320 (1997)
239. G.W. Stewart, Building an old-fashioned sparse solver (2003), https://drum.lib.umd.edu/handle/1903/1312
240. J. Stoer, R. Bulirsch, *Introduction to Numerical Analysis* (Springer, New York, 1983)
241. J. Stoer, R. Bulirsch, *Introduction to Numerical Analysis*. Texts in Applied Mathematics, vol. 12, 3rd edn. (Springer, New York, 2002). Translated from the German by R. Bartels, W. Gautschi and C. Witzgall
242. M.H. Stone, The generalized Weierstrass approximation theorem. Math. Mag. **21**, 167–184, 237–254 (1948)
243. G. Strang, On the Kantorovich inequality. Proc. Am. Math. Soc. **11**, 468 (1960)
244. A.H. Stroud, *Approximate Calculation of Multiple Integrals*. Prentice-Hall Series in Automatic Computation (Prentice-Hall, Inc., Englewood Cliffs, 1971)
245. M.E. Taylor, *Partial Differential Equations I. Basic Theory*. Applied Mathematical Sciences, vol. 115, 2nd edn. (Springer, New York, 2011)
246. P. (Chebyshev, P.) Tchebichef, Des valeurs moyennes. J. Math. Pures Appl. **2**(12), 177–184 (1867)
247. R. Trobec, B. Slivnik, P. Bulić, B. Robič, *Introduction to Parallel Computing*. Undergraduate Topics in Computer Science (Springer, Cham, 2018). From algorithms to programming on state-of-the-art platforms
248. U. Trottenberg, C.W. Oosterlee, A. Schüller, *Multigrid* (Academic, San Diego, 2001). With contributions by A. Brandt, P. Oswald and K. Stüben
249. A.M. Turing, Rounding-off errors in matrix processes. Q. J. Mech. Appl. Math. **1**, 287–308 (1948)
250. R.A. van de Geijn, *Using PLAPACK - Parallel Linear Algebra Package* (MIT Press, Cambridge, 1997)
251. P.J.M. van Laarhoven, E.H.L. Aarts, *Simulated Annealing: Theory and Applications* (Springer Netherlands, Dordrecht, 1987)
252. J.R. Van Zandt, Efficient cubature rules. Electron. Trans. Numer. Anal. **51**, 219–239 (2019)
253. S. Vigna, Further scramblings of Marsaglia's `xorshift` generators. J. Comput. Appl. Math. **315**, 175–181 (2017)
254. S. Vigna, It is high time we let go of the Mersenne twister. CoRR (2019), arXiv:1910.06437
255. J. von Neumann, Various techniques used in connection with random digits, in *Monte Carlo Method*. National Bureau of Standards Applied Mathematics Series, vol. 12, ed. by A.S. Householder, G.E. Forsythe, H.H. Germond (US Government Printing Office, Washington, 1951), pp. 36–38
256. J. von Neumann, H.H. Goldstine, Numerical inverting of matrices of high order. Bull. Am. Math. Soc. **53**, 1021–1099 (1947)

257. L.T. Watson, S.C. Billups, A.P. Morgan, Algorithm 652: HOMPACK: a suite of codes for globally convergent homotopy algorithms. ACM Trans. Math. Softw. **13**, 281–310 (1987)
258. H. Wendland, *Scattered Data Approximation*. Cambridge Monographs on Applied and Computational Mathematics, vol. 17 (Cambridge University Press, Cambridge, 2005)
259. R.C. Whaley, J. Dongarra, Automatically tuned linear algebra software, in *Ninth SIAM Conference on Parallel Processing for Scientific Computing. CD-ROM Proceedings* (1999)
260. H. Whitney, Analytic extensions of differentiable functions defined in closed sets. Trans. Am. Math. Soc. **36**(1), 63–89 (1934)
261. J.H. Wilkinson, Error analysis of direct methods of matrix inversion. J. Assoc. Comput. Mach. **8**, 281–330 (1961)
262. P. Wolfe, Convergence conditions for ascent methods. SIAM Rev. **11**, 226–235 (1969)
263. P. Wolfe, Convergence conditions for ascent methods. II. Some corrections. SIAM Rev. **13**, 185–188 (1971)
264. F. Yates, *The Design and Analysis of Factorial Experiments* (Imperial Bureau of Soil Science, London, 1937)
265. D.M. Young, Generalizations of property *A* and consistent orderings. SIAM J. Numer. Anal. **9**, 454–463 (1972)
266. J. Yu, Symmetric Gaussian quadrature formulae for tetrahedronal regions. Comput. Methods Appl. Mech. Eng. **43**(3), 349–353 (1984)

# Index





D. E. Stewart, *Numerical Analysis: A Graduate Course*, CMS/CAIMS Books in Mathematics 4, https://doi.org/10.1007/978-3-031-08121-7

**T**