MATH 2160
Numerical Analysis I Course Notes

Richard M. Slevinsky

Richard.Slevinsky@umanitoba.ca
Department of Mathematics
University of Manitoba

Contents

Preface

These course notes are intended to offer a deep understanding of a set of classical topics in numerical analysis that may be covered in a single term of undergraduate study. They are designed to be accessible to undergraduates, self-contained, and complete. Certain sections of the notes are taken or inspired from other sources due to the excellent expository work found elsewhere. These sources include: a previous version of Shaun Lui's MATH 2160 Numerical Analysis notes at the University of Manitoba; the University of Oxford's Part A Numerical Analysis notes based on the book by Süli and Mayers [1]; Gautschi's "Numerical Analysis," [2]; Fletcher's "Practical Methods of Optimization," [3, 4]; and, course notes of Nick Trefethen and Sheehan Olver made available online.

Chapter 1

Fundamentals

Numerical analysis is the design and analysis of accurate and efficient algorithms to solve problems in science and engineering. Computers work with numbers which can be represented by finitely many bits, whereas some real numbers require infinitely many bits to represent exactly. Thus, there is an error involved in representing each real number, and this error propagates in subsequent arithmetic operations. An analysis is required to determine in which instances we can trust the result of a long sequence of calculations and in which instances we should design a better algorithm or use higher precision.

Standard domains used in these notes include the natural numbers $\mathbb{N}$, the integers $\mathbb{Z}$, the rationals $\mathbb{Q}$, and:

Name	Unit Interval	Real Line	Complex Plane	Torus	Unit Circle
Notation	$\mathbb{I}$	$\mathbb{R}$	$\mathbb{C}$	$\mathbb{T}$	$\mathbb{U}$
Definition	$[-1,1]$	$(-\infty,+\infty)$	$\{z = x + \mathrm{i}y : x, y \in \mathbb{R} \text{ and } \mathrm{i}^2 = -1\}$	$[0, 2\pi)$	$e^{\mathrm{i}\mathbb{T}}$

1.1 Vector Spaces

Definition 1.1.1. *A set V with elements from the field $\mathbb{F}$, together with operations of vector addition and scalar multiplication, is a* vector space *if it satisfies the following axioms. For all vectors $u, v, w \in V$ and for all scalars $\alpha, \beta \in \mathbb{F}$:*

1. *Associativity of Addition:* $u + (v + w) = (u + v) + w$;
2. *Commutativity of Addition:* $u + v = v + u$;
3. *Identity Element of Addition:* $\exists 0 \in V : u + 0 = u$;
4. *Inverse Elements of Addition:* $\exists - v \in V : v + (-v) = 0$;
5. *Compatibility of Scalar Multiplication:* $\alpha(\beta v) = (\alpha\beta)v$;
6. *Identity Element of Scalar Multiplication:* $1v = v$;
7. *Distributivity of Scalar Multiplication with Respect to Vector Addition:* $\alpha(u + v) = \alpha u + \alpha v$; *and,*

8. *Distributivity of Scalar Multiplication with Respect to Scalar Addition:* $(\alpha + \beta)u = \alpha u + \beta u$.

Example 1.1.2. *Let $n \in \mathbb{N}$. The spaces $\mathbb{R}^n$ and $\mathbb{C}^n$ are vector spaces. However, the space $\mathbb{R}_{\geq 0} = \{x \in \mathbb{R} : x \geq 0\}$ is not a vector space because inverse elements of addition are absent.*

Example 1.1.3. *The set of functions from any fixed set D to elements in the field $\mathbb{F}$ forms a vector space since for any $x \in D$ and $\alpha \in \mathbb{F}$:*

$$(f + g)(x) = f(x) + g(x), \quad \text{and} \quad (\alpha f)(x) = \alpha f(x).$$

1.1.1 Normed Vector Spaces

The essential notions of size and distance in a vector space are captured by norms. These are the yardsticks with which we measure approximations and convergence throughout numerical analysis.

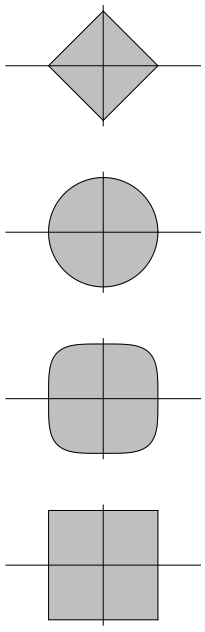
Definition 1.1.4. *A* norm *is a function $\|\cdot\| : V \to \mathbb{R}$ that assigns a real-valued length to each vector. In order to conform to a reasonable notion of length, a norm must satisfy the following three conditions. For all vectors x and y and for all scalars $\alpha \in \mathbb{F}$:*

1. $\|x\| \geq 0$, *and* $\|x\| = 0$ *only if* $x = 0$;
2. $\|x + y\| \leq \|x\| + \|y\|$; *and,*
3. $\|\alpha x\| = |\alpha| \, \|x\|$.

In words, these conditions require that 1. the norm of a nonzero vector is positive, 2. the norm of a vector sum does not exceed the sum of the norms of its parts–the *triangle inequality*, and 3. scaling a vector scales its norm by the same amount.

The most important class of vector norms, the p-norms, are defined below. The closed unit ball $\{x \in \mathbb{C}^n : \|x\| \leq 1\}$ corresponding to each norm is illustrated to the right for the case $n = 2$.

$$\begin{aligned}
\|x\|_1 &= \sum_{i=1}^{n} |x_i| \,, \\
\|x\|_2 &= \left(\sum_{i=1}^{n} |x_i|^2 \right)^{\frac{1}{2}} = \sqrt{x^* x}, \\
\|x\|_p &= \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \quad (1 \leq p < \infty), \\
\|x\|_\infty &= \max_{1 \leq i \leq n} |x_i| \,,
\end{aligned} \tag{1.1}$$



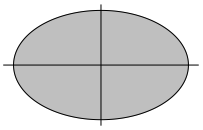
The 2-norm is the Euclidean length function; its unit ball is spherical. The 1-norm is known as the Manhattan norm since it is the distance a taxi has to drive in a street grid[1]. The Sergel plaza in Stockhom, Sweden has the shape of the unit ball in the 4-norm; the Danish poet Piet Hein popularized this "superellipse" as a pleasing shape for objects such as conference tables.

Aside from the p-norms, the most useful norms are the *weighted p-norms*, where each of the coordinates of a vector space is given its own weight. In general, given any norm $\|\cdot\|$, a weighted norm can be written as:

$$\|x\|_W = \|Wx\|. \tag{1.2}$$

Here, W is the diagonal matrix in which the i^{th} diagonal entry is the weight $w_i \neq 0$. For example, a weighted 2-norm $\|\cdot\|_W$ on $\mathbb{C}^n$ is specified as follows:

$$\|x\|_W = \left(\sum_{i=1}^{n} |w_i x_i|^2\right)^{\frac{1}{2}}. \tag{1.3}$$



One can also generalize the idea of weighted norms by allowing W to be an arbitrary nonsingular matrix.

The most important norms in this book are the unweighted 2-norm and its induced matrix norm.

The concept of the p-norms can be extended to the case of vector spaces containing infinitely many elements. Such spaces are called sequence spaces, and the following spaces describe sequence spaces that have finite norm.

Definition 1.1.5. *Let $1 \leq p \leq \infty$. The space ℓ^p contains the set of all vectors $x \in V$ with finite p-norm:*

$$\ell^p = \{x \in V : \|x\|_p < \infty\}. \tag{1.4}$$

Example 1.1.6. *Every vector $x \in \mathbb{R}^n$ is in ℓ^p. Consider the vector $y \in \mathbb{R}^\infty$ whose entries are defined by:*

$$y_i = \frac{1}{i}, \quad \text{for} \quad i = 1, \ldots, \infty. \tag{1.5}$$

Then, $y \in \ell^p$ for every $p > 1$ since for every $\epsilon > 0$, $\sum_{i=1}^{\infty} i^{-1-\epsilon} < \infty$ by the integral test for convergence.

While we will not focus on ℓ^p spaces in this course, we will be interested in their analogues for functions.

1.1.2 Function Spaces

Let $D \subset \mathbb{R}$ or $\mathbb{C}$ denote the domain of a function f.

Definition 1.1.7. *A function is continuous at every point $c \in D$ if $f(c)$ exists and if $\lim_{\substack{x \to c \\ x \in D}} f(x) = f(c)$.*

Definition 1.1.8. *The space:*

- *$C(D)$ contains all continuous functions on D;*

[1]The 1-norm is also used by airlines to define the maximal allowable size of a suitcase.

- $C^n(D)$ *contains all functions whose* n^{th} *derivatives are continuous on* D*;*
- $C^\infty(D)$ *contains all functions with continuous derivatives of all orders on* D*; and,*
- $A(D)$ *contains all functions analytic on* D*. That is, they are equal to their Taylor series at every point in* D*.*

Example 1.1.9. *The functions:*

- $f(x) = |x| \in C(\mathbb{I})$*;*
- $f(x) = |x|^{n+1} \in C^n(\mathbb{I})$ *(*n *even*[2]*);*
- $f(x) = \begin{cases} e^{-1/x^2} & \text{for} \quad x \neq 0, \\ 0 & \text{for} \quad x = 0, \end{cases} \in C^\infty(D)$ *for every* $0 \in D \subset \mathbb{C}$*;*
- $f(x) = \dfrac{1}{1+x^2} \in A(D)$ *where* $D = \{z \in \mathbb{C} : |z| < 1\}$ *is the open unit disk; and,*
- $f(x) = \sin x \in A(\mathbb{C})$.

The analytic functions $A(\mathbb{C})$ are so smooth that they are further denoted as *entire* functions.

The notion of a vector p-norm extends readily to functions that are Lebesgue integrable. For $1 \leq p < \infty$:

$$\|f\|_p := \left(\int_D |f(x)|^p \, \mathrm{d}x \right)^{\frac{1}{p}}, \tag{1.6}$$

and for $p = \infty$:

$$\|f\|_\infty := \sup_{x \in D} |f(x)| \,. \tag{1.7}$$

Sometimes the measure is not uniform. These cases correspond to the weighted p-norms:

$$\|f\|_{p,\mu} := \left(\int_D |f(x)|^p \, \mathrm{d}\mu(x) \right)^{\frac{1}{p}}. \tag{1.8}$$

Definition 1.1.10. *The space* $L^p(D, \, \mathrm{d}\mu(x))$ *contains all functions Lebesgue*[3] *integrable on* D *with respect to the measure* $\mathrm{d}\mu(x)$*:*

$$L^p(D, \, \mathrm{d}\mu(x)) := \left\{ f : \|f\|_{p,\mu} < \infty \right\}. \tag{1.9}$$

Remark 1.1.11.

1. When it is clear from the context, the measure $\mathrm{d}\mu(x)$ will dropped from the weighted p-norm;
2. The space $L^p(D) = L^p(D, \, \mathrm{d}x)$; and,
3. If the measure $\mu \in C^1(D)$, then the space $L^p(D, \, \mathrm{d}\mu(x)) = L^p(D, w(x) \, \mathrm{d}x)$ where $w(x) = \mu'(x)$.

[2] For n odd $|x|^{n+1}$ is a polynomial and polynomials are entire.

[3] This definition ignores functions that are zero almost-everywhere, in which case the functions are indistinguishable from zero in norm since $\|f\|_p = 0$.

1.1.3 Inner Product Spaces

Inner product spaces are vector spaces with an additional structure called an inner product.

Definition 1.1.12. *An inner product $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{F}$ is a sesquilinear map that assigns a value in the field $\mathbb{F}$ to two vectors in V. For all vectors $x, y, z \in V$ and for all scalars $\alpha, \beta \in \mathbb{F}$, the inner product satisfies:*

1. *conjugate symmetry: $\langle y, x \rangle = \overline{\langle x, y \rangle}$;*
2. *linearity in the second argument: $\langle x, \alpha y + \beta z \rangle = \alpha \langle x, y \rangle + \beta \langle x, z \rangle$; and,*
3. *positive-definiteness: $\langle x, x \rangle \geq 0$, and $\langle x, x \rangle = 0$ only if $x = 0$.*

Example 1.1.13. *For the vector spaces:*

1. *$V = \mathbb{R}^n$, we can define the inner product by the conventional dot product:*

$$\langle x, y \rangle := x^\top y = \sum_{i=1}^{n} x_i y_i;$$

2. *$V = \mathbb{C}^n$, we can define the inner product similarly:*

$$\langle x, y \rangle := x^* y = \sum_{i=1}^{n} x_i^* y_i; \quad \textit{and},$$

3. *$V = L^2(D, \, \mathrm{d}\mu(x))$, we can define the inner product as:*

$$\langle f, g \rangle := \int_D \overline{f(x)} g(x) \, \mathrm{d}\mu(x).$$

The inner products for $V = \mathbb{R}^n$ and $\mathbb{C}^n$ can be extended to the case where $n = \infty$.

Example 1.1.14. *For the normed vector spaces:*

1. *$V = \ell^2$, the norm can be defined in terms of the inner product:*

$$\|x\|_2 := \sqrt{\langle x, x \rangle}; \quad \textit{and},$$

2. *$V = L^2(D, \, \mathrm{d}\mu(x))$, the norm can be defined similarly:*

$$\|f\|_2 := \sqrt{\langle f, f \rangle}.$$

We can see that of the normed vector spaces described so far, ℓ^p spaces are generally *not* inner product spaces, and neither are the function spaces $L^p(D, \, \mathrm{d}\mu(x))$. It is only when $p = 2$ that we get normed inner product spaces.

The following inequality, known as the Cauchy–Schwarz inequality, is a useful inequality we will require further on.

Theorem 1.1.15. *Let u, v be arbitrary elements in an inner product space. Then, the following inequality holds:*

$$|\langle u, v\rangle| \leq \|u\|_2 \|v\|_2 . \tag{1.10}$$

Proof. If $\langle u, v\rangle = 0$, the inequality is trivially true. Assume $u \neq 0$ and $v \neq 0$. If $\lambda = \dfrac{\overline{\langle u, v\rangle}}{\|v\|_2^2}$, then:

$$\begin{aligned} 0 \leq \|u - \lambda v\|_2^2 &= \langle u - \lambda v, u - \lambda v\rangle \\ &= \|u\|_2^2 - \overline{\lambda}\langle v, u\rangle - \lambda\langle u, v\rangle + \lambda\overline{\lambda}\,\|v\|_2^2 , \\ &= \|u\|_2^2 - \frac{|\langle u, v\rangle|^2}{\|v\|_2^2} - \frac{|\langle u, v\rangle|^2}{\|v\|_2^2} + \frac{|\langle u, v\rangle|^2}{\|v\|_2^2}, \\ &= \|u\|_2^2 - \frac{|\langle u, v\rangle|^2}{\|v\|_2^2}, \end{aligned}$$

and the inequality follows by rearranging terms. □

Example 1.1.16. *For the inner product spaces:*

1. *$V = \ell^2$, the Cauchy–Schwarz inequality reads:*

$$\left|\sum_{i=1}^{n} \overline{x_i} y_i\right|^2 \leq \left(\sum_{i=1}^{n} |x_i|^2\right)\left(\sum_{i=1}^{n} |y_i|^2\right); \quad \textit{and},$$

2. *$V = L^2(D,\, \mathrm{d}\mu(x))$, the Cauchy–Schwarz inequality reads:*

$$\left|\int_D \overline{f(x)} g(x)\, \mathrm{d}\mu(x)\right|^2 \leq \int_D |f(x)|^2\, \mathrm{d}\mu(x) \int_D |g(x)|^2\, \mathrm{d}\mu(x).$$

1.1.4 Matrix Norms Induced by Vector Norms

An $m \times n$ matrix can be viewed as a vector in an mn-dimensional space: each of the mn entries of the matrix is an independent coordinate. Any mn-dimensional norm can therefore be used for measuring the "size" of such a matrix.

However, in dealing with a space of matrices, certain special norms are more useful than the vector norms already discussed. These are the *induced matrix norms*, defined in terms of the behaviour of a matrix as an operator between its normed domain and range spaces.

Given vector norms $\|\cdot\|_{(n)}$ and $\|\cdot\|_{(m)}$ on the domain and range of $A \in \mathbb{C}^{m\times n}$, respectively, the induced matrix norm $\|A\|_{(m,n)}$ is the smallest number C for which the following inequality holds for all $x \in \mathbb{C}^n$:

$$\|Ax\|_{(m)} \leq C \|x\|_{(n)} . \tag{1.11}$$

In other words, $\|A\|_{(m,n)}$ is the supremum of the ratios $\|Ax\|_{(m)} / \|x\|_{(n)}$ over all vectors $x \in \mathbb{C}^n$–the maximum factor by which A can "stretch" a vector x. We say that $\|\cdot\|_{(m,n)}$ is the matrix norm induced by $\|\cdot\|_{(m)}$ and $\|\cdot\|_{(n)}$.

Because of condition 3. of Definition 1.1.4 the action of A is determined by its action on unit vectors. Therefore, the matrix norm can be defined equivalently in terms of the images of the unit vectors under A:

$$\|A\|_{(m,n)} = \sup_{\substack{x\in\mathbb{C}^n \\ x\neq 0}} \frac{\|Ax\|_{(m)}}{\|x\|_{(n)}} = \sup_{\substack{x\in\mathbb{C}^n \\ \|x\|_{(n)}=1}} \|Ax\|_{(m)} \,. \tag{1.12}$$

This form of the definition can be convenient for visualizing induced matrix norms.

Example 1.1.17. *The matrix:*

$$A = \begin{bmatrix} 1 & 2 \\ 0 & 2 \end{bmatrix}, \tag{1.13}$$

maps $\mathbb{C}^2$ to $\mathbb{C}^2$. It also maps $\mathbb{R}^2$ to $\mathbb{R}^2$, which is more convenient if we want to draw pictures and also (it can be shown) sufficient for determining matrix p-norms, since the coefficients of A are real.

Figure 1.1 depicts the action of A on the unit balls of $\mathbb{R}^2$ defined by the 1-, 2-, and ∞-norms. From this figure, one can see a graphical interpretation of these three norms of A. Regardless of the norm, A maps $e_1 = (1,0)^\top$ to the first column of A, namely e_1 itself, and $e_1 = (0,1)^\top$ to the second column of A, namely, $(2,2)^\top$. In the 1-norm, the unit vector x that is amplified most by A is $(0,1)^\top$ (or its negative), and the amplification factor is 4. In the ∞-norm, the unit vector x that is amplified most by A is $(1,1)^\top$ (or its negative), and the amplification factor is 3. In the 2-norm, the unit vector that is amplified most by A is the vector indicated by the dashed line in the figure (or its negative), and the amplification factor is approximately 2.9208. (Note that it must be at least $\sqrt{8} \approx 2.8284$, since $(0,1)^\top$ maps to $(2,2)^\top$.) We shall consider how to calculate such 2-norm results in Chapter 2.

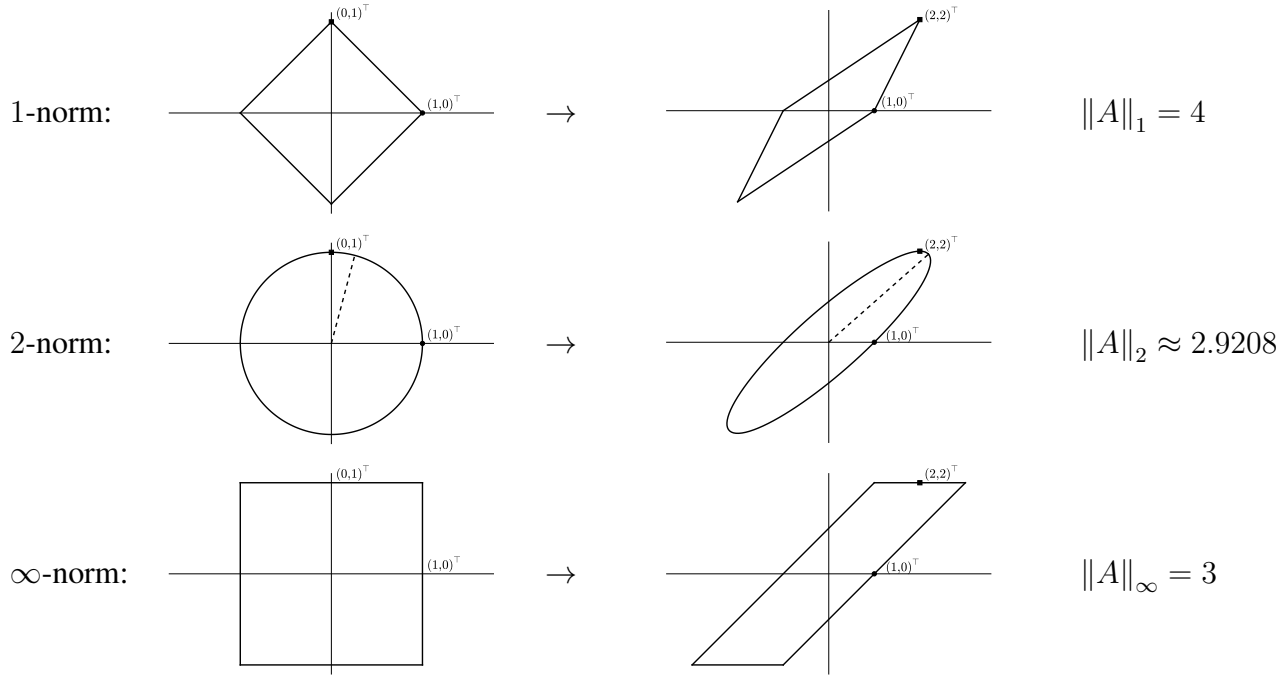


Figure 1.1: The induced matrix 1-, 2-, and ∞-norms of A.

Example 1.1.18. *The p-norm of a diagonal matrix. Let D be the diagonal matrix:*

$$D = \begin{bmatrix} d_1 & & & \\ & d_2 & & \\ & & \ddots & \\ & & & d_n \end{bmatrix}. \tag{1.14}$$

Then, as in the second row of Figure 1.1, the image of the 2-norm unit sphere under D is an n-dimensional ellipse whose semiaxis lengths are given by the numbers $|d_i|$. The unit vectors amplified most by D are those that are mapped to the longest semiaxis of the ellipse, of length $\max_i\{|d_i|\}$. Therefore, we have $\|D\|_2 = \max\limits_{1\le i\le n}\{|d_i|\}$.

This result for the 2-norm generalizes to any p: if D is diagonal, then $\|D\|_p = \max\limits_{1\le i\le n}\{|d_i|\}$.

Example 1.1.19. *The 1-norm of a matrix. If A is any $m \times n$ matrix, then $\|A\|_1$ is equal to the "maximum column sum" of A. We explain and derive this result as follows. Write A in terms of its columns:*

$$A = \left[\begin{array}{c|c|c} a_1 & \cdots & a_n \end{array}\right], \tag{1.15}$$

where each a_j is an m-vector. Consider the diamond-shaped 1-norm unit ball in $\mathbb{C}^n$, illustrated in Figure 1.1. This is the set $\{x \in \mathbb{C}^n : \sum_{j=1}^n |x_j| \le 1\}$. Any vector Ax in the image of this set satisfies:

$$\|Ax\|_1 = \left\| \sum_{j=1}^n x_j a_j \right\|_1 \le \sum_{j=1}^n |x_j| \, \|a_j\|_1 \le \max_{1\le j\le n} \|a_j\|_1 . \tag{1.16}$$

Therefore, the induced matrix 1-norm satisfies $\|A\|_1 \le \max_{1\le j\le n} \|a_j\|_1$. By choosing $x = e_j$, where j maximizes $\|a_j\|_1$, we attain this bound, and thus the matrix norm is:

$$\|A\|_1 = \max_{1\le j\le n} \|a_j\|_1 . \tag{1.17}$$

Example 1.1.20. *The ∞-norm of a matrix. By much the same argument, it can be shown that the ∞-norm of an $m \times n$ matrix is equal to the "maximum row sum:"*

$$\|A\|_\infty = \max_{1\le i\le m} \|a_i^*\|_1 , \tag{1.18}$$

where a_i^* denotes the i^{th} row of A.

Finally, it should be clear that induced norms are submultiplicative, since:

$$\|AB\|_p = \sup_{\substack{x\in\mathbb{C}^n \\ x\neq 0}} \frac{\|ABx\|_p}{\|x\|_p} = \sup_{\substack{x\in\mathbb{C}^n \\ x\neq 0}} \frac{\|ABx\|_p}{\|Bx\|_p} \frac{\|Bx\|_p}{\|x\|_p} \le \sup_{\substack{y\in\mathbb{C}^n \\ y\neq 0}} \frac{\|Ay\|_p}{\|y\|_p} \sup_{\substack{x\in\mathbb{C}^n \\ x\neq 0}} \frac{\|Bx\|_p}{\|x\|_p} = \|A\|_p \, \|B\|_p .$$

1.2 Computer Representation of Numbers

Bits are the smallest piece of information that can be stored in memory on a digital computer. Their role is similar to the role played by atomic particles in quantum physics. The simplest type of numbers we can store with one bit are the Boolean numbers `true` and `false`. If bits can take on at least two distinct values, such as yes/no or on/off, then Boolean numbers can be represented in one bit.

1.2.1 Integers

The most important integers we can represent on a digital computer are those nearest to 0. Let $b \in \mathbb{N}$ be a *radix*, which is Latin for root or base. Integers can be represented in radix-b as:

$$\pm(d_k d_{k-1} \cdots d_0)_b = \pm \left(d_k b^k + d_{k-1} b^{k-1} + \cdots + d_0 \right),$$

where the numbers $0 \leq d_k < b$. In this representation, integers from 0 to:

$$(b-1)(b^k + b^{k-1} + \cdots + 1) = b^{k+1} - 1,$$

each have a unique signature, and similarly for negative integers from[4] $-b^{k+1}$ up to -1. Using a finite number of bits, a finite number of integers can be represented on a digital computer. As the sign bit behaves just as a Boolean number, we need one extra bit for its inclusion.

1.2.2 Fixed-Point Numbers

Fixed-point numbers are approximations to real numbers $\mathbb{R}$ with a fixed number of digits before the radix point '.' and another fixed number of digits after the radix point:

$$\pm(d_k d_{k-1} \cdots d_0 . d_{-1} \cdots d_{-n})_b = \pm \left(d_k b^k + d_{k-1} b^{k-1} + \cdots + d_0 + d_{-1} b^{-1} + \cdots + d_{-n} b^{-n} \right).$$

Because fixed-point operations can easily produce results that require more bits than the operands[5], there is possibility for information loss. In scientific computing, fixed-point numbers have fallen from favour because of the need to have scale-invariance in calculations: consider performing any calculation with Planck's constant $h \approx 6.63 \times 10^{-34}\,\mathrm{J \cdot s}$ and the speed of light squared $c^2 \approx 8.99 \times 10^{16}\,\mathrm{m^2/s^2}$. This would be an expensive calculation in fixed-point arithmetic, yet it could more easily be done by hand.

1.2.3 Floating-Point Numbers

Modern computers are all binary[6], whose binary bits can take the value 0 or 1. Groups of eight contiguous bits make up a byte; 2^{10} bytes make a kilobyte (KB); 2^{20} bytes make a megabyte (MB); 2^{30} bytes make a gigabyte (GB) and so on and so forth.

The IEEE Standard for Floating-Point Arithmetic (IEEE 754) includes a directive on how to create floating-point numbers to approximate the real numbers on a computer. (Normalized) floating-point numbers are alternative approximations to the reals $\mathbb{R}$ with a fixed number of digits after the radix point *multiplied by* a power $p - M$ of the base 2:

$$\pm 2^{p-M} \times (1.d_{-1} \cdots d_{-n})_2 = \pm \left(2^{p-M} + d_{-1} 2^{p-M-1} + \cdots + d_{-n} 2^{p-M-n} \right).$$

As with the integers and fixed-point numbers, one bit is used to store the sign of the floating-point number, n bits are used for the fractional part known as the *mantissa*, and the *exponent* p is represented by an

[4] We can represent one more negative integer because we only need one 0.

[5] If $c = a + b$, then $+$ is the operator and a and b are operands.

[6] In the 1950's, the Soviets experimented with a balanced ternary (or trinary) computer called Setun, whose bits took the values $\{-1, 0, 1\}$. The mass production of binary computers has diminished the importance of ternary computing, though it may make a comeback with optical computing.

unsigned integer. If we use m bits to represent p, then we can choose to use the 2^m integers in the range $0 \leq p < 2^m$. In total, then, we require $1 + m + n$ bits to represent the floating-point number. M is known as the *exponent bias*.

The IEEE 754 standard prescribes Table 1.1 for binary floating-point numbers.

Table 1.1: IEEE 754 standard for binary signed floating-point representation of numbers.

Common Name	Base	Digits	Decimal Digits	E Bits	E min	E max	Decimal E max	M
Half precision	2	10	3.01	5	-14	$+15$	4.51	15
Single precision	2	23	6.92	8	-126	$+127$	38.23	127
Double precision	2	52	15.65	11	-1022	$+1023$	307.95	1023

Notice in Table 1.1 that the exponent minimum and maximum values fall just short of the maximum allowable range given the number of bits. Subnormal numbers, that is, numbers below 2^{-1022} in double precision for example, can only be represented with fewer digits because some of these digits take the surrogate role of extending the power of the exponent.[7] Indeed, the smallest representable number in double precision is $2^{-1022-52} = 2^{-1074}$.

Example 1.2.1. • *Numbers in double precision, with exponent bias* $M = 1023$*, are stored as:*

$\underbrace{a}_{\text{1 sign bit}}\ \underbrace{bbbbbbbbbbb}_{\text{11 exponent bits}}\ \underbrace{ccccccccccccccccccccccccccccccccccccccccccccccccccccc}_{\text{52 significand bits}}$

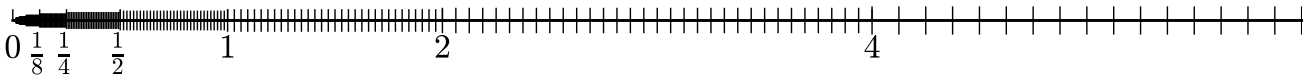
- *For example,* 1.0 *is represented by:*

  $0011111111110000000000000000000000000000000000000000000000000000$

- *and* -29.0 *is:*

  $1100000000111101000000000000000000000000000000000000000000000000$

Floating-point numbers are more versatile than fixed-point numbers because they are distributed in a way which preserves an approximately constant relative precision throughout the range of values they can represent: the density of floating-point numbers accumulates near 0 and disperses at infinity. In double precision, floating-point numbers are equally distributed in $[\frac{1}{2}, 1)$ with spacing 2^{-53}, equally distributed in $[1, 2)$ with spacing 2^{-52}, equally distributed in $[2, 4)$ with spacing 2^{-51} and so on and so forth, and equally distributed in $[\frac{1}{4}, \frac{1}{2})$ with spacing 2^{-54}, and so on and so forth, as illustrated below.



How continuous is real life?

Having decided to approximate real numbers by floating-point numbers on a computer, we might ask how accurately we can represent real life. Avogadro's number $N_A \approx 6.02 \times 10^{23}\,\text{molecules/mol}$ tells us how

[7] Furthermore, subnormal numbers are denormalized and thus start with a 0 rather than a 1.

many molecules of anything are in one mol. Now, for the sake of the argument, let us take $1\,\mathrm{m}^3$ of the air with density $\rho = 1.225\,\mathrm{kg/m}^3$ in between you and me. In this cubic meter of air, consisting of roughly 78% nitrogen with a molar mass of $M(N_2) = 28.02\,\mathrm{g/mol}$, we find that there are:

$$N_A \times M(N_2)^{-1} \times \frac{1000\,\mathrm{g}}{1\,\mathrm{kg}} \times \rho \times 1\,\mathrm{m}^3 \approx 2.63 \times 10^{25}\,\mathrm{molecules},$$

of nitrogen in this cubic meter of air. If we assume they are uniformly distributed, then there are only:

$$\sqrt[3]{2.63 \times 10^{25}}\,\mathrm{molecules} \approx 3 \times 10^8\,\mathrm{molecules},$$

lined up in $1\,\mathrm{m}$ in any given direction.

On the other hand, with IEEE double precision floating-point numbers we can distinguish numbers at the scale of one unit of least precision (ulp), that is, the value the least significant digit represents if it is 1, which is $\approx 2.2045 \times 10^{-16}$ in double precision. Therefore, while we sometimes think of real life as continuous and digital computation as discrete, it is now the case that the level of discreteness on today's computers can be regarded as more refined than real life!

1.2.4 The Standard Model

To store a real number, that is, to obtain a machine representation of a real number, we must truncate the fractional part of the binary expansion of the real number to n bits, the number of bits in the mantissa. This is done by truncating (ignoring all digits after the n^{th} bit) or by rounding. In the latter, if the $(n+1)^{\text{th}}$ bit of the expansion of the given number is one, then we round up (taking the first n bits and adding one to the last bit). Otherwise, we simply take the first n bits as in truncation.

Floating-point arithmetic does not obey the usual rules of arithmetic such as the laws of associativity and distributivity. When programming, it is better practice to avoid statements such as:

```
if x == .3 then...
```

Here, rounding errors may mean that the conditional code block is not executed even if the mathematical value of x is .3. In this case, it is better to test `if |x-.3| < tol` where `tol` is some small tolerance.

For a real number x, we denote floating-point approximations to quantities by $\mathrm{fl}(x)$. Using Higham's standard model for floating-point arithmetic [5], we assume that whenever x and y are floating-point numbers and $\circledast$ is one of the four arithmetic operations $+$, $-$, $\times$, or $\div$, we have:

$$\mathrm{fl}(x \circledast y) = (x \circledast y)(1+\delta)^{\pm 1}, \quad \text{where} \quad |\delta| \le \epsilon, \tag{1.19}$$

where ϵ is a unit of least precision.

Lemma 1.2.2 (Higham [5])**.** *If $|\delta_i| \le \epsilon$, and $\rho_i = \pm 1$ for $i = 1, \ldots, n$, and $n\epsilon < 1$, then:*

$$\langle n \rangle := \prod_{i=1}^{n} (1+\delta_i)^{\rho_i} = 1 + \theta_n, \qquad |\theta_n| \le \frac{n\epsilon}{1-n\epsilon} =: \gamma_n. \tag{1.20}$$

Proof. The proof is by induction. □

The notation $\langle n \rangle$ is useful as a counter to show the accumulation of n relative errors due to rounding. The counters can be manipulated using the rules:

$$\langle m \rangle + \langle n \rangle = \langle \max\{m, n\} \rangle, \tag{1.21}$$
$$\langle m \rangle - \langle n \rangle = \langle \max\{m, n\} \rangle, \tag{1.22}$$
$$\langle m \rangle \langle n \rangle = \langle m + n \rangle, \tag{1.23}$$
$$\frac{\langle m \rangle}{\langle n \rangle} = \langle m + n \rangle. \tag{1.24}$$

1.2.5 Cancellation Error

Cancellation error is the loss of significant digits when subtracting two nearly equal floating-point numbers. In base-10 and with 6 digits, suppose we wish to find the difference of 1.2345671 and 1.2345660. After rounding:

$$\mathrm{fl}(1.234567 - 1.234566) = 0.000001 = 1.000000 \times 10^{-6},$$

a result which has only one correct digit. This phenomenon can lead to an answer which is completely different from the exact one, especially over the course of a long sequence of operations.

Example 1.2.3. *In base-*10 *and with three digits:*

$$\mathrm{fl}(\,\mathrm{fl}(\sqrt{9.01}) - 3.00) = \mathrm{fl}(3.00 - 3.00) = 0,$$

which is very different from the exact answer $\approx 1.6662 \times 10^{-3}$.

Observe that:

$$\sqrt{9.01} - 3 = (\sqrt{9.01} - 3)\frac{\sqrt{9.01} + 3}{\sqrt{9.01} + 3} = \frac{0.01}{\sqrt{9.01} + 3}.$$

Calculation of this quantity using the last expression has no cancellation error and leads to 1.67×10^{-3} *which is as good as one can expect with three digits.*

Example 1.2.4. *Solve* $x^2 + 10^9 x - 3 = 0$. *By the quadratic formula, the two roots are:*

$$x_- = \frac{-10^9 - \sqrt{10^{18} + 12}}{2}, \qquad x_+ = \frac{-10^9 + \sqrt{10^{18} + 12}}{2}.$$

There is no cancellation in x_- *which can be computed accurately in double precision. On the other hand,* x_+ *is a difference of nearly equal numbers. In double precision, the result is exactly zero. As above, a better calculation is:*

$$x_+ = \frac{-10^9 + \sqrt{10^{18} + 12}}{2}\frac{10^9 + \sqrt{10^{18} + 12}}{10^9 + \sqrt{10^{18} + 12}} = \frac{6}{10^9 + \sqrt{10^{18} + 12}},$$

which is accurately computed in double precision.

Example 1.2.5. *Consider evaluation of* $f(x) = \dfrac{\log(1 + x)}{x}$ *in a neighbourhood of* $x = 0$. *By l'Hôpital's rule,* $f(0) = 1$. *The naïve way to evaluate* f *suffers from serious cancellation error near* $x = 0$. *In floating-point arithmetic, a better way is the following:*

$$\begin{cases} \dfrac{\log(\,\mathrm{fl}(1 + x))}{\mathrm{fl}(\,\mathrm{fl}(1 + x) - 1)}, & \textit{for} \quad \mathrm{fl}(1 + x) \neq 1, \\ 1, & \textit{for} \quad \mathrm{fl}(1 + x) = 1. \end{cases}$$

Another alternative is to use the (standard) library function $\log 1\mathrm{p}$*, defined by its Taylor series near* $x = 1$ *and by* $\log \mathrm{fl}(1 + x)$ *sufficiently far away to cause no loss of accuracy, such that* $f(x) = \log 1\mathrm{p}(x)/x$*. Figure 1.2 shows the relative error in approximating* f *using all three methods.*

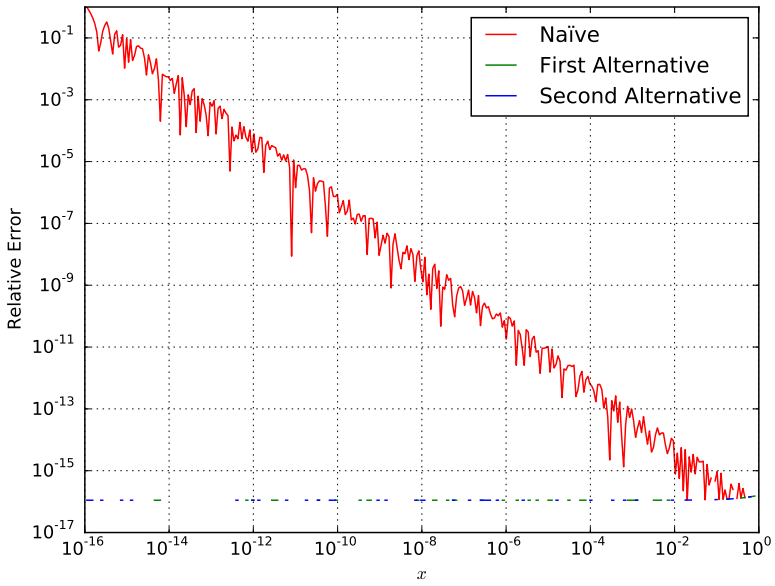


Figure 1.2: Comparison of the naïve and alternative methods for computing $f(x) = \frac{\log(1+x)}{x}$.

Sums of floating-point numbers are ubiquitous in scientific computing. They occur when evaluating inner products, norms, means, variances, and all kinds of nonlinear functions. Although at first sight summation might appear to offer little scope for algorithmic ingenuity, the usual "recursive summation" (with various orderings) is just one of a variety of possible techniques. We describe several summation methods and their error analyses in the next example. No one method is uniformly more accurate than the others, but some guidelines can be given on the choice of method in particular cases.

Example 1.2.6. *Consider the computation of* $S_n = \sum_{i=1}^{n} x_i$ *using the following algorithms:*

1. *Recursive summation, where we set* $s = 0$ *and recursively add* x_i *to* s *for* $i = 1, \ldots, n$*; and,*

2. *Pairwise summation (also know as cascade summation), where the* x_i *are summed in pairs according to:*

$$y_i = x_{2i-1} + x_{2i}, \quad \text{for} \quad i = 1, \ldots \lfloor \tfrac{n}{2} \rfloor, \quad (y_{(n+1)/2} = x_n \text{ if } n \text{ is odd}),$$

and this pairwise summation repeats recursively on the y_i*, obtaining the sum in* $\lceil \log_2 n \rceil$ *stages.*

Using the standard model, recursive summation leads to the accumulation of $\langle n - 1 \rangle$ *rounding errors, while pairwise summation only leads to the accumulation of* $\langle \lceil \log_2 n \rceil \rangle$ *rounding errors. For example, for*

recursive summation of S_8, we have:

$$\begin{aligned}
\mathrm{fl}(S_8) &= \mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(x_1+x_2)+x_3)+x_4)+x_5)+x_6)+x_7)+x_8),\\
&= ((((((((x_1+x_2)\langle 1\rangle + x_3)\langle 1\rangle + x_4)\langle 1\rangle + x_5)\langle 1\rangle + x_6)\langle 1\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= ((((((((x_1+x_2)\langle 2\rangle + x_3\langle 1\rangle) + x_4)\langle 1\rangle + x_5)\langle 1\rangle + x_6)\langle 1\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= ((((((x_1+x_2+x_3)\langle 2\rangle + x_4)\langle 1\rangle + x_5)\langle 1\rangle + x_6)\langle 1\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= (((((x_1+x_2+x_3+x_4)\langle 3\rangle + x_5)\langle 1\rangle + x_6)\langle 1\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= ((((x_1+x_2+x_3+x_4+x_5)\langle 4\rangle + x_6)\langle 1\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= (((x_1+x_2+x_3+x_4+x_5+x_6)\langle 5\rangle + x_7)\langle 1\rangle + x_8)\langle 1\rangle,\\
&= ((x_1+x_2+x_3+x_4+x_5+x_6+x_7)\langle 6\rangle + x_8)\langle 1\rangle,\\
&= S_8\langle 7\rangle.
\end{aligned}$$

On the other hand, for pairwise summation of S_8, we have:

$$\begin{aligned}
\mathrm{fl}(S_8) &= \mathrm{fl}(\,\mathrm{fl}(\,\mathrm{fl}(x_1+x_2)+\,\mathrm{fl}(x_3+x_4))+\,\mathrm{fl}(\,\mathrm{fl}(x_5+x_6)+\,\mathrm{fl}(x_7+x_8))),\\
&= (((x_1+x_2)\langle 1\rangle + (x_3+x_4)\langle 1\rangle)\langle 1\rangle + ((x_5+x_6)\langle 1\rangle + (x_7+x_8)\langle 1\rangle)\langle 1\rangle)\langle 1\rangle,\\
&= ((x_1+x_2+x_3+x_4)\langle 2\rangle + (x_5+x_6+x_7+x_8)\langle 2\rangle)\langle 1\rangle,\\
&= S_8\langle 3\rangle.
\end{aligned}$$

Note that there are even other alternative methods, and even changing the direction of the recursive summation can lead to an improvement in certain sums. For example, if the sequence $|x_i|$ is decreasing, the reverse recursive summation can be even more accurate than pairwise summation, exhibiting no error growth with n! This shows that when we start making assumptions on x_i, the error analysis with counters can be made more precise. This also shows that no algorithm is uniformly more accurate. Figure 1.3 compares forward/reverse recursive and pairwise summation for a given sequence.

1.3 Conditioning and Stability

A problem typically has an input and an output. The input consists of a set of data, say, the coefficients of some equation, and the output of another set of numbers uniquely determined by the input, say, all the roots of the equation in some prescribed order. If we collect the input in a vector $x \in X$, and the output in the vector $y \in Y$, where X and Y are normed vector spaces, we may thus think of a problem as a map f, given by:

$$f : X \to Y, \quad y = f(x). \tag{1.25}$$

What we are interested in is the sensitivity of the map f at some given point x to a small perturbation of x, that is, how much bigger (or smaller) is the perturbation in y compared to the perturbation in x? In particular, we wish to measure the degree of sensitivity by a single number – the *condition number* of the map f at the point x. We emphasize that, as we perturb x, the function f is always assumed to be evaluated exactly, with infinite precision. The condition of f, therefore is an inherent property of the map f and does not depend on any algorithmic considerations concerning its implementation in finite precision on a computer.

This is not to say that knowledge of the condition of a problem is irrelevant to any algorithmic solution of the problem. On the contrary, the reason is that quite often the *computed solution* y^*, using a specific

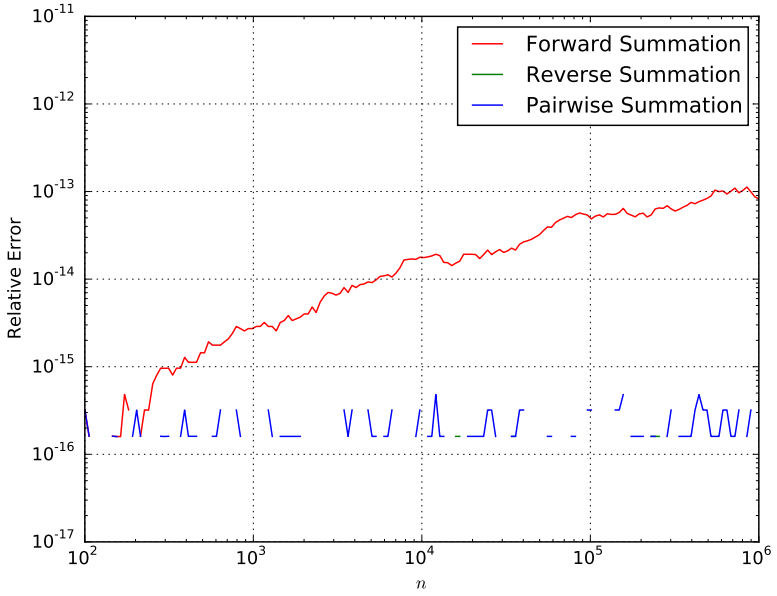


Figure 1.3: Comparison of the recursive and pairwise summation algorithms for $S_n = \sum_{i=1}^{n} \frac{(-1)^i}{i}$.

algorithm in finite precision, can be demonstrated to be the *exact* solution to a "nearby" problem, that is:

$$y^* = f(x^*), \tag{1.26}$$

where x^* is a point close to the given data x:

$$x^* = x + \Delta x, \tag{1.27}$$

and moreover, the distance $\|\Delta x\|_X$ of x^* to x can be estimated in terms of the machine precision. Therefore, if we know how strongly (or weakly) the map f reacts to a small perturbation, such as Δx, we can say something about the error $\|y^* - y\|_Y$ in the solution caused by this perturbation. This, indeed, is an important technique of error analysis – know as *backward error analysis* – which was pioneered in the 1950's by Givens, Lanczos, and Wilkinson.

Example 1.3.1. *1. For a given matrix $A \in \mathbb{R}^{m\times n}$, define the problem of matrix-vector multiplication $f : \mathbb{R}^n \to \mathbb{R}^m$ with the Euclidean norms on the input and output:*

$$f(x) = Ax. \tag{1.28}$$

This problem encodes the sensitivity of matrix-vector multiplication to perturbations in the vector.

2. For a given vector $x \in \mathbb{R}^n$, define the problem of matrix-vector multiplication $f : \mathbb{R}^{m\times n} \to \mathbb{R}^m$ with the (induced) 2-norms on the input and output:

$$f(A) = Ax. \tag{1.29}$$

This problem encodes the sensitivity of matrix-vector multiplication to perturbations in the matrix.

3. *Define the whole problem of matrix-vector multiplication* $f : \mathbb{R}^{m\times n} \times \mathbb{R}^n \to \mathbb{R}^m$*:*

$$f(A, x) = Ax. \tag{1.30}$$

Clearly, we can attach the 2*-norm to the output. For the input, we attach the norm:*

$$\|(A, x)\| = \max\{\|A\|_2, \|x\|_2\}. \tag{1.31}$$

This problem encodes the sensitivity of matrix-vector multiplication to perturbations in both *the matrix and the vector.*

4. *For a given vector* $x \in \mathbb{R}^n$*, define the problem of inverse matrix-vector multiplication (solving a linear system)* $f : \mathbb{R}^n \to \mathbb{R}^m$ *with the (induced)* 2*-norms on the input and output:*

$$f(x) = A^{-1}x. \tag{1.32}$$

This problem encodes the sensitivity of matrix inversion to perturbations in the vector.

Maps f between more general spaces (in particular, function spaces) have also been considered from the point of view of conditioning, but eventually, these spaces are reduced to finite-dimensional spaces for practical implementation.

We can measure the error using either *absolute* or *relative* error. The absolute error for the input x perturbed by Δx is:

$$\Delta y = f(x + \Delta x) - f(x), \tag{1.33}$$

while the relative error (if $f : X \to \mathbb{R}$ has one-dimensional output) is:

$$\frac{\Delta y}{y} = \frac{f(x + \Delta x) - f(x)}{f(x)}, \qquad y \neq 0. \tag{1.34}$$

Definition 1.3.2. *The* absolute condition number *of a problem is a measure of how much the absolute error in the input is magnified to cause absolute error in the output:*

$$\mathrm{cond}_{\mathrm{abs}}(f, \epsilon) := \sup_{\|\Delta x\|_X \leq \epsilon} \frac{\|f(x + \Delta x) - f(x)\|_Y}{\|\Delta x\|_X}. \tag{1.35}$$

We see that by this definition, the condition number gives us a bound on the absolute error subject to an ϵ-perturbation in the input:

$$\|f(x + \Delta x) - f(x)\|_Y \leq \mathrm{cond}_{\mathrm{abs}}(f, \epsilon) \|\Delta x\|_X . \tag{1.36}$$

Definition 1.3.3. *The* relative condition number *of a problem is a measure of how much the relative error in the input is magnified to cause relative error in the output:*

$$\mathrm{cond}_{\mathrm{rel}}(f, \epsilon) := \sup_{\|\Delta x\|_X \leq \epsilon} \frac{\|f(x + \Delta x) - f(x)\|_Y}{\|\Delta x\|_X} \frac{\|x\|_X}{\|f(x)\|_Y}. \tag{1.37}$$

We see that by this definition, the condition number gives us a bound on the relative error subject to an ϵ-perturbation in the input:

$$\frac{\|f(x + \Delta x) - f(x)\|_Y}{\|f(x)\|_Y} \leq \mathrm{cond}_{\mathrm{rel}}(f, \epsilon) \frac{\|\Delta x\|_X}{\|x\|_X}. \tag{1.38}$$

Example 1.3.4. *Consider the problem of evaluating the function $f \in C^1(D)$ at some point $x \in D \subset \mathbb{R}$. The relative condition number is:*

$$\mathrm{cond}_{\mathrm{rel}}(f,\epsilon) = \sup_{|\Delta x| \leq \epsilon} \frac{|f(x+\Delta x) - f(x)|}{|\Delta x|} \frac{|x|}{|f(x)|}.$$

If the perturbations are truly small, then the limiting value as $\epsilon \to 0$ can be taken as a sensible approximation of the relative condition number:

$$\mathrm{cond}_{\mathrm{rel}}(f,\epsilon) \approx \lim_{\epsilon \to 0} \mathrm{cond}_{\mathrm{rel}}(f,\epsilon) = \left| \frac{x f'(x)}{f(x)} \right|.$$

Definition 1.3.5. *The p-norm* condition number of a square matrix *is defined by:*

$$\mathrm{cond}_p(A) := \|A\|_p \left\|A^{-1}\right\|_p. \tag{1.39}$$

The condition number of a matrix gives a least upper bound on the relative condition number of the problems of matrix-vector multiplication and solution of linear systems described earlier.

Example 1.3.6. *1. For the problem of matrix-vector multiplication subject to perturbations in the vector, the relative condition number is:*

$$\mathrm{cond}_{\mathrm{rel}}(f,\epsilon) = \sup_{\|\Delta x\|_p \leq \epsilon} \frac{\|A(x+\Delta x) - Ax\|_p}{\|\Delta x\|_p} \frac{\|x\|_p}{\|Ax\|_p} = \frac{\|x\|_p}{\|Ax\|_p} \sup_{\|\Delta x\|_p \leq \epsilon} \frac{\|A\Delta x\|_p}{\|\Delta x\|_p} = \frac{\|x\|_p}{\|Ax\|_p} \|A\|_p.$$

The condition number is bounded by $\mathrm{cond}_p(A)$ since:

$$\mathrm{cond}_{\mathrm{rel}}(f,\epsilon) = \frac{\|A^{-1}Ax\|_p}{\|Ax\|_p} \|A\|_p \leq \frac{\|A^{-1}\|_p \|Ax\|_p}{\|Ax\|_p} \|A\|_p = \mathrm{cond}_p(A).$$

2. For the problem of solution of linear systems subject to perturbations in the vector, the relative condition number is:

$$\begin{aligned} \mathrm{cond}_{\mathrm{rel}}(f,\epsilon) &= \sup_{\|\Delta x\|_p \leq \epsilon} \frac{\|A^{-1}(x+\Delta x) - A^{-1}x\|_p}{\|\Delta x\|_p} \frac{\|x\|_p}{\|A^{-1}x\|_p}, \\ &= \frac{\|x\|_p}{\|A^{-1}x\|_p} \sup_{\|\Delta x\|_p \leq \epsilon} \frac{\|A^{-1}\Delta x\|_p}{\|\Delta x\|_p} = \frac{\|x\|_p}{\|A^{-1}x\|_p} \left\|A^{-1}\right\|_p. \end{aligned}$$

The condition number is still *bounded by $\mathrm{cond}_p(A)$ since:*

$$\mathrm{cond}_{\mathrm{rel}}(f,\epsilon) = \frac{\|AA^{-1}x\|_p}{\|A^{-1}x\|_p} \left\|A^{-1}\right\|_p \leq \frac{\|A\|_p \|A^{-1}x\|_p}{\|A^{-1}x\|_p} \left\|A^{-1}\right\|_p = \mathrm{cond}_p(A).$$

Example 1.3.7. *The Hilbert matrix is the square matrix of dimension n with entries $[H_n]_{i,j} = \dfrac{1}{i+j-1}$. This symmetric matrix has an extremely large 2-norm condition number. A result due to Szegő states that:*

$$\mathrm{cond}_2(H_n) \sim \frac{\left(\sqrt{2}+1\right)^{4n+4}}{2^{\frac{15}{4}}\sqrt{\pi n}}, \quad \text{as} \quad n \to \infty. \tag{1.40}$$

Table 1.2 shows some numerical values of the condition number of H_n with respect to the dimension n. In fact, the rapid growth of the condition number implies that beyond $n = 12$, neither the matrix-vector product nor the solution of a linear system can be computed with any reliability in 64-bit floating-point arithmetic.

Table 1.2: The 2-norm condition number of Hilbert matrices.

n	$\mathrm{cond}_2(H_n)$	Estimate (1.40)
5	$4.76607250242561 \times 10^{5}$	$2.88200032765989 \times 10^{7}$
10	$1.6026286870216889 \times 10^{13}$	$9.219189347346499 \times 10^{14}$
15	$6.116565791619838 \times 10^{20}$	$3.405342605196914 \times 10^{22}$
20	$2.4521565858153033 \times 10^{28}$	$1.334151505013483 \times 10^{30}$
25	$1.0096548421909462 \times 10^{36}$	$5.398384956972705 \times 10^{37}$
30	$4.227806388187525 \times 10^{43}$	$2.2293945465873128 \times 10^{45}$
35	$1.7910802642986633 \times 10^{51}$	$9.337427577656723 \times 10^{52}$
40	$7.652913898310316 \times 10^{58}$	$3.951345265521789 \times 10^{60}$

Note that this example illustrates an *extreme* scenario, where the condition number of the family of Hilbert matrices grows extremely rapidly with respect to the dimension n. Such problems are deemed *ill-conditioned*. In many cases of interest, the condition number will be low enough for us to obtain reasonable accuracy in floating-point arithmetic. These problems are deemed *well-conditioned*.

Remark 1.3.8. The condition number of the *problem* is different from the condition number of the *algorithm* we use to solve the problem. For example, suppose we wish to solve the linear system:

$$Ax = b.$$

One useful approach is to develop a factorization of $A = BC$, where the inverses of the matrices B and C can be applied to the vector b in fewer operations than a dense solve. In this case, the algorithm consists of:

1. Solve for $y := B^{-1}b$; and,
2. Solve for $x = C^{-1}y$.

Using matrix inequalities, we can compare condition numbers of the problem and the algorithm:

$$\begin{aligned}\mathrm{cond}_p(A) = \mathrm{cond}_p(BC) &= \|BC\|_p \left\|(BC)^{-1}\right\|_p = \|BC\|_p \left\|C^{-1}B^{-1}\right\|_p, \\ &\leq \|B\|_p \|C\|_p \left\|C^{-1}\right\|_p \left\|B^{-1}\right\|_p = \mathrm{cond}_p(B)\,\mathrm{cond}_p(C).\end{aligned}$$

Therefore, the condition number of the algorithm, consisting of the product $\mathrm{cond}_p(B)\,\mathrm{cond}_p(C)$, is *at least as large* as the condition number of the problem, but depending on our choice of the factorization BC, it could be much worse!

Algorithms that do not needlessly generate ill-conditioning, that is, algorithms for which the condition number of the problem and the condition number of algorithm do not differ egregiously, are said to be stable algorithms. While this concept is rather vague, in certain instances the stability of a given algorithm can be made precise.

1.4 Theorems of Calculus

Theorem 1.4.1 (Mean Value Theorem)**.** *If $f \in C([a,b])$, where $a < b$, and if $f \in C^1(a,b)$, then $\exists c \in (a,b)$ for which:*

$$f'(c) = \frac{f(b) - f(a)}{b - a}.$$

Theorem 1.4.2 (Integral Mean Value Theorem)**.** *If $f, g \in C([a,b])$, and $g(x) \geq 0$ on this interval, then $\exists c \in (a,b)$ for which:*

$$\int_a^b f(x)g(x)\,\mathrm{d}x = f(c)\int_a^b g(x)\,\mathrm{d}x.$$

Theorem 1.4.3 (Intermediate Value Theorem)**.** *If $f \in C([a,b])$ and c is any number between $f(a)$ and $f(b)$ then there is at least one number $\xi \in (a,b)$ such that $f(\xi) = c$.*

Theorem 1.4.4 (Taylor's Theorem)**.** *Let $k \in \mathbb{N}$ and let the function f be k times differentiable at the point a. Then Taylor's approximation is:*

$$f(x) = f(a) + f'(a)(x-a) + \cdots + \frac{f^{(k)}(a)}{k!}(x-a)^k + r_k(x),$$

where $r_k(x)$ is a remainder term. The formulæ for $r_k(x)$ are due to several others:

Peano form of the remainder *$r_k(x) = o(|x-a|^k)$, that is, as $x \to a$, the remainder tends to zero faster than the k^{th} power of the difference;*

Lagrange form of the remainder *if $f \in C^{k+1}(a,x)$ and if $f \in C^k([a,x])$, then $\exists c \in (a,x)$ for which:*

$$r_k(x) = \frac{f^{(k+1)}(c)}{(k+1)!}(x-a)^{k+1};\ \textit{and},$$

Cauchy form of the remainder *if $f \in C^{k+1}(a,x)$ and if $f \in C^k([a,x])$, then $\exists c \in (a,x)$ for which:*

$$r_k(x) = \frac{f^{(k+1)}(c)}{k!}(x-c)^k(x-a).$$

Each form of the remainder in Taylor's theorem is useful in a different context.

1.5 Asymptotic Analysis and Complexity

It is important to be able to describe the characteristics of numerical algorithms. While wall time of an algorithm is certainly important, mathematically it is interesting to understand how the algorithm scales with increasing, say, the problem dimension n. If it doubles,

- how much longer can we expect to wait?
- how much larger do the errors grow?
- how much more space must be allocated to carry out the extra computations?

To this end, we will find that notation from asymptotic analysis will be useful. This is an extension of our natural capabilities of estimation, and is even more effective when coupled with the p-norms.

Definition 1.5.1. *1. If there exists an $0 < M < \infty$ such that:*

$$|f(z)| \leq M|g(z)|, \quad \text{as} \quad z \to z_0,\ z \in D, \tag{1.41}$$

then we say $f(z) = \mathcal{O}(g(z))$ (f is on the order of g) as $z \to z_0$. Equivalently:

$$\lim_{\substack{z \to z_0 \\ z \in D}} \left| \frac{f(z)}{g(z)} \right| = M. \tag{1.42}$$

2. If for all $0 < M < \infty$:

$$|f(z)| < M|g(z)|, \quad \text{as} \quad z \to z_0,\ z \in D, \tag{1.43}$$

then we say $f(z) = o(g(z))$ as $z \to z_0$. Equivalently:

$$\lim_{\substack{z \to z_0 \\ z \in D}} \left| \frac{f(z)}{g(z)} \right| = 0. \tag{1.44}$$

3. If $f(z) = \mathcal{O}(g(z))$ with the constant $M = 1$, then we say $f(z) \sim g(z)$ (f is asymptotic to g) as $z \to z_0$. Equivalently:

$$\lim_{\substack{z \to z_0 \\ z \in D}} \left| \frac{f(z)}{g(z)} \right| = 1. \tag{1.45}$$

Example 1.5.2. *Using Taylor's theorem and the geometric series, we know that:*

1. $\sin x \sim x, \quad \text{as} \quad x \to 0$;
2. $\cos x = 1 + \mathcal{O}(x^2), \quad \text{as} \quad x \to 0$;
3. $\dfrac{e^x - 1}{x} \sim 1, \quad \text{as} \quad x \to 0$; *and,*
4. $\dfrac{1}{1 - x^2} = -\dfrac{1}{x^2(1 - 1/x^2)} = -\dfrac{1}{x^2} \displaystyle\sum_{n=0}^{\infty} \left(\frac{1}{x^2} \right)^n = -\frac{1}{x^2} + o(x^{-3}) \quad \text{as} \quad x \to \infty.$

1.6 Simple Numerical Algorithms

1.6.1 Horner's Rule

Horner's rule [6] is an algorithm for evaluating polynomials in the monomial basis. Let $p_n(x) = \sum_{k=0}^{n} c_k x^k$.

Algorithm 1.6.1 (Horner [6])**.**

1. *Set $S = c_n$;*
2. *For $k = n - 1, \ldots, 0$, set $S = c_k + xS$.*

S is the result of evaluating p_n at the point x.

This algorithm can be reasoned by re-expressing the polynomial p_n as:

$$p_n(x) = c_0 + x(c_1 + x(c_2 + \cdots + x(c_{n-1} + c_n x))).$$

Horner's rule requires:

- n additions and n multiplications, or $\mathcal{O}(n)$ flops; and,
- 1 unit of storage during execution, or $\mathcal{O}(1)$ storage.

It is an example of a perfectly fine algorithm for numerically evaluating polynomials in the monomial basis.

1.6.2 Matrix-Vector Multiplication

Matrix-vector multiplication allows us to determine the vector $b := Ax$, where $A \in \mathbb{F}^{m\times n}$, $x \in \mathbb{F}^n$, and $b \in \mathbb{F}^m$. Generically speaking, we find:

Algorithm 1.6.2.
for $i = 1, \ldots, m$, *set:*

$b_i = 0$

for $j = 1, \ldots, n$, *set:*

$b_i = b_i + A_{i,j}x_j$

end

end

Matrix-vector multiplication requires:

- mn additions and mn multiplications, or $\mathcal{O}(mn)$ flops; and,
- m units of storage for the output and 1 unit of storage during execution[8], or $\mathcal{O}(m)$ storage.

1.7 Problems

1. Vector and matrix p-norms are related by various inequalities, often involving the dimensions m or n. For each of the following, verify the inequality and give an example of a nonzero vector or matrix (for general m, n) for which equality is achieved. In this problem, x is an n-vector and A is an $m \times n$ matrix.

   (a) $\|x\|_\infty \leq \|x\|_2$;

[8] On hardware that supports fused multiply-add, the algorithm requires 0 units of storage during execution.

(b) $\|x\|_2 \leq \sqrt{n}\,\|x\|_\infty$;

(c) $\|A\|_\infty \leq \sqrt{n}\,\|A\|_2$; and,

(d) $\|A\|_2 \leq \sqrt{m}\,\|A\|_\infty$.

2. Let the matrix $A = uv^\top$ be the *outer product* of the vectors $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ such that $A_{i,j} = u_i v_j$. Use the Cauchy–Schwarz inequality to show that $\|A\|_2 = \|u\|_2\,\|v\|_2$.

3. The function $\mathrm{vec}(\cdot)$ maps a matrix $A \in \mathbb{F}^{m\times n}$ to the vector $a \in \mathbb{F}^{mn}$ such that $A_{i,j} = a_{i+m(j-1)}$. Show that the function $\|\cdot\|_F = \|\mathrm{vec}(\cdot)\|_2$ is a matrix norm. (Indeed, it is known as the Frobenius norm.) Prove the inequality $\|AB\|_F \leq \|A\|_F\,\|B\|_F$.

4. Convert:

   - the binary number $(0.111101)_2$ into decimal; and,
   - the octal number $(0.3416)_8$ into binary.

5. The condition number of the Hilbert matrices grows so rapidly that using a built-in computer function `cond` will *not* accurately compute the condition number past about $n = 12$. Instead, use the definition of $\mathrm{cond}_2(A) := \|A\|_2\,\|A^{-1}\|_2$ together with the inverse Hilbert matrices with elements given by:

$$[H_n^{-1}]_{i,j} = (-)^{i+j}(i+j-1)\binom{n+i-1}{n-j}\binom{n+j-1}{n-i}\binom{i+j-2}{i-1}^2,$$

   following these steps.

   (a) Use properties of binomial coefficients to simplify the expression to the following:

$$[H_n^{-1}]_{i,j} = \frac{(-)^{i+j}n^2}{i+j-1}\binom{n-1}{i-1}\binom{n-1}{j-1}\binom{n+i-1}{i-1}\binom{n+j-1}{j-1};$$

   (b) Derive the recurrences:

$$[H_n^{-1}]_{i,j} = -\frac{i+j-2}{i+j-1}\cdot\frac{n-i+1}{i-1}\cdot\frac{n+i-1}{i-1}\cdot[H_n^{-1}]_{i-1,j},$$

   and:

$$[H_n^{-1}]_{i,j} = -\frac{i+j-2}{i+j-1}\cdot\frac{n-j+1}{j-1}\cdot\frac{n+j-1}{j-1}\cdot[H_n^{-1}]_{i,j-1},$$

   to be able to populate the lower half of the inverse recursively starting from $[H_n^{-1}]_{1,1} = n^2$; and,

   (c) Use symmetry to populate the upper half, and compute the 2-norm condition number by its definition. Furthermore, use the fact that the entries of H_n^{-1} are integers to organize the arithmetic to minimize the number of rounding errors. Replicate Table 1.2. What is the largest n for which you can compute $\mathrm{cond}_2(H_n)$?

6. The integrals $E_n(z) := \displaystyle\int_z^\infty \frac{e^{-x}}{x^n}\,\mathrm{d}x$ are known as the family of *exponential integrals*. Derive the recurrence relation:

$$E_n(z) = \frac{e^{-z}}{z^n} - nE_{n+1}(z).$$

Using an inequality on the family of integrals, derive:

$$E_n(z) = \frac{e^{-z}}{z^n} + \mathcal{O}\left(\frac{e^{-z}}{z^{n+1}}\right), \quad \text{as} \quad z \to \infty.$$

Use the recurrence relation to derive the first few terms of the asymptotic expansion:

$$ze^z E_1(z) \sim 1 - \frac{1}{z} + \frac{2}{z^2} - \frac{6}{z^3} + \cdots + = \sum_{n=0}^{\infty} \frac{n!}{(-z)^n}, \quad \text{as} \quad z \to \infty.$$

Chapter 2

Linear Systems of Equations

Given a nonsingular matrix $A \in \mathbb{C}^{n\times n}$ and a vector $b \in \mathbb{C}^n$, the goal is to find $x \in \mathbb{C}^n$ such that $Ax = b$. Equivalently, find $x = (x_1, x_2, \ldots, x_n)^\top$ for which:

$$\begin{array}{rcl} A_{1,1}x_1 + A_{1,2}x_2 + \cdots + A_{1,n}x_n &=& b_1, \\ A_{2,1}x_1 + A_{2,2}x_2 + \cdots + A_{2,n}x_n &=& b_2, \\ \vdots \quad\quad \vdots \quad\quad \cdots \quad\quad \vdots &&\quad\quad \vdots \\ A_{n,1}x_1 + A_{n,2}x_2 + \cdots + A_{n,n}x_n &=& b_n. \end{array} \tag{2.1}$$

This is without a doubt the most important problem in scientific computing; many problems in science and engineering are often restated as a linear system of equations. For instance, after discretization, a system of linear partial differential equations becomes a system of linear equations.

We shall examine two classes of methods for solving linear systems. The first class of methods are direct methods, with Gaussian elimination as the poster child. Direct methods are designed to return the exact solution (modulo rounding errors) in a finite number of steps. The second class of methods are iterative methods, which are designed to converge only after an infinite number of steps. Each class of methods is useful for different classes of linear systems. In particular, iterative methods are primarily used for the solution of large (data-)sparse linear systems[1].

We shall also consider the solution of overdetermined systems $Ax = b$, where $A \in \mathbb{C}^{m\times n}$ where $m > n$ and $x \in \mathbb{C}^n$ and $b \in \mathbb{C}^m$. The so-called least squares solution of the system can be obtained by solving the square system $A^*Ax = A^*b$. While directly multiplying both sides of the equation by A^* roughly doubles the inaccuracy, we will explore matrix factorizations that provide well-conditioned algorithms for least squares solutions.

2.1 Gaussian Elimination

Gaussian Elimination (GE) is the prototypical direct method for small to medium systems ($n < 1000$). In this section, we shall start the discussion with the basic version, which does not work for all matrices. Then, the full version will be given in the following section.

[1] Sparsity in a matrix means *many* of its entries are zero. Data sparsity in a matrix means that while many entries may *not* be zero, the entries are prescribed by a formula involving a very small number of degrees of freedom. For example, in the data-sparse outer-product $A = uv^\top$, the mn entries of A are all determined by only $m + n$ entries in u and v.

2.1.1 Without Pivoting

Example 2.1.1. *Consider the system:*

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

In the basic version of GE, we augment the matrix with the righthand side as:

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 2 & 1 & -2 & 3 \\ -3 & 1 & 1 & -6 \end{bmatrix}.$$

Let r_i denote the i^{th} row of the above augmented matrix. The notation $r_i \leftarrow r_i + ar_j$ is used to denote replacing row i by row i plus a times row j. GE performs a sequence of row operations to reduce the augmented matrix to upper triangular form. For this example, performing $r_2 \leftarrow -2r_1 + r_2$ and $r_3 \leftarrow 3r_1 + r_3$ results in:

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 7 & -2 & 3 \end{bmatrix}.$$

Notice that after these operations, the first column is zero except for the first entry. Next, we perform $r_3 \leftarrow \frac{7}{3}r_2 + r_3$ and we obtain:

$$\begin{bmatrix} 1 & 2 & -1 & 3 \\ 0 & -3 & 0 & -3 \\ 0 & 0 & -2 & -4 \end{bmatrix}.$$

Note that the new matrix is now upper triangular. The solution can easily be calculated by back substitution: from the last equation, $-2z = -4$ or $z = 2$; from the second equation, $-3y = -3$ or $y = 1$; and, from the first equation:

$$x + 2y - z = 3,$$

or using our values for y and z, we find $x = 3$.

The full algorithm of Gaussian Elimination can be described in terms of overwriting the entries of the augmented matrix.

Algorithm 2.1.2 (Gaussian Elimination)**.**
for columns $j = 1, 2, \ldots, n-1$,
    for rows $i = j+1, j+2, \ldots, n$,

$$r_i \leftarrow r_i - \frac{A_{i,j}}{A_{j,j}} r_j, \quad \text{and} \quad b_i \leftarrow b_i - \frac{A_{i,j}}{A_{j,j}} b_j.$$

    end
end

Let us calculate the complexity of GE. Since the notation $r_i \leftarrow r_i - \dfrac{A_{i,j}}{A_{j,j}} r_j$ is equivalent to:

for columns $k = j+1, j+2, \ldots, n$,

$$A_{i,k} \leftarrow A_{i,k} - \frac{A_{i,j}}{A_{j,j}} A_{j,k},$$

end

This is approximately $2(n-j)$ flops as the multiplier $\dfrac{A_{i,j}}{A_{j,j}}$ is calculated only once. N.B. $A_{j,j}$ is called the *pivot*. Overall, therefore, the complexity of GE is approximately:

$$\sum_{j=1}^{n-1} 2(n-j)^2 = 2\sum_{l=1}^{n-1} l^2 = 2\frac{n(n-1)(2n-1)}{6} = \frac{2}{3}n^3 + \mathcal{O}(n^2),$$

flops. The calculations involving the vector b are:

$$\sum_{j=1}^{n-1} 2(n-j) = 2\sum_{l=1}^{n-1} l = 2\frac{n(n-1)}{2} = n^2 + \mathcal{O}(n),$$

flops.

Example 2.1.3. *We wish to solve:*

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix} : \textit{is represented as } \begin{bmatrix} 3 & -1 & 2 & 12 \\ 1 & 2 & 3 & 11 \\ 2 & -2 & -1 & 2 \end{bmatrix}.$$

$$\begin{matrix} r_2 \leftarrow r_2 - \frac{1}{3}r_1 \\ r_3 \leftarrow r_3 - \frac{2}{3}r_1 \end{matrix} \Rightarrow \begin{bmatrix} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & -\frac{4}{3} & -\frac{7}{3} & -6 \end{bmatrix}$$

$$r_3 \leftarrow r_3 + \tfrac{4}{7}r_2 \Rightarrow \begin{bmatrix} 3 & -1 & 2 & 12 \\ 0 & \frac{7}{3} & \frac{7}{3} & 7 \\ 0 & 0 & -1 & -2 \end{bmatrix}$$

We then use back substitution to obtain:

$$\begin{aligned} x_3 &= 2, \\ x_2 &= \frac{3}{7}\left(7 - \frac{7}{3}\cdot 2\right) = 1, \\ x_1 &= \frac{1}{3}\left(12 - (-1)\cdot 1 - 2\cdot 2\right) = 3. \end{aligned}$$

2.1.2 With Partial Pivoting

Naïve GE does not always work! Consider the system:

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \tag{2.2}$$

whose solution is clearly $x = (2,1)^\top$. If we apply the generic GE algorithm, it will fail at the first step due to division by zero. We are free, however, to re-order the equations (i.e. the rows) into any order that suits us. Clearly, a better ordering of the system (2.2) is:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \tag{2.3}$$

On paper, then, it will suffice to ensure that, at every step in the GE algorithm, we re-order the remaining equations to ensure we are not pivoting with 0. However, we can take this argument further and consider the system with a very small parameter $\epsilon \neq 0$:

$$\begin{bmatrix} \epsilon & 1 \\ 1 & \epsilon \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \tag{2.4}$$

whose solution is $x = \left(\frac{2-\epsilon}{1-\epsilon^2}, \frac{1-2\epsilon}{1-\epsilon^2}\right)^\top \approx (2,1)^\top$ for small ϵ. If we apply the generic GE algorithm, then we find:

$$\begin{bmatrix} \epsilon & 1 \\ 0 & \epsilon - \epsilon^{-1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 - \epsilon^{-1} \end{bmatrix}. \tag{2.5}$$

In double precision, $\epsilon - \epsilon^{-1}$ is indistinguishable from $-\epsilon^{-1}$ when $|\epsilon| \leq 7.450580596923828 \times 10^{-9}$. Therefore, in this régime, the matrix is numerically equivalent to:

$$\mathrm{fl}\left(\begin{bmatrix} \epsilon & 1 \\ 0 & \epsilon - \epsilon^{-1} \end{bmatrix}\right) = \begin{bmatrix} \epsilon & 1 \\ 0 & -\epsilon^{-1} \end{bmatrix}. \tag{2.6}$$

The solution of the numerically equivalent system is $x_2 = 1 - 2\epsilon$, and $x_1 = \epsilon^{-1}(1 - x_2)$. Clearly, there is no issue in computing x_2, and its numerical value even approaches $\frac{1-2\epsilon}{1-\epsilon^2}$ as $\epsilon \to 0$. However, computing x_1 via this algorithm results in a loss of accuracy on the order of $-\log_{10}(\epsilon)$ digits, due to the subtractive cancellation involved in $1 - x_2$.

To mitigate the accumulation and amplification of disastrous rounding errors, we can interchange rows at every intermediate step in the GE algorithm to ensure we are pivoting with the entry with the largest magnitude at step j, $j_{\max} = \arg\max_{j \leq i \leq n} |A_{i,j}|$ and $r_j \leftrightarrow r_{j_{\max}}$. This will ensure that we are not multiplying/dividing by exceptionally small/large values, and thus the amplification of rounding errors is minimized.

Example 2.1.4. *Apply GE with partial pivoting to* $\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$. *First, we permute rows:*

$$r_3 \leftrightarrow r_1 \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{bmatrix}.$$

Then, we eliminate:

$$\begin{array}{l} r_2 \leftarrow r_2 - \frac{1}{2} r_1 \\ r_3 \leftarrow r_3 - \frac{1}{4} r_1 \\ r_4 \leftarrow r_4 - \frac{3}{4} r_1 \end{array} \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ 0 & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{bmatrix}.$$

Again, we permute rows:

$$r_2 \leftrightarrow r_4 \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{bmatrix},$$

and we eliminate:

$$\begin{matrix} r_3 \leftarrow r_3 + \frac{3}{7} r_2 \\ r_4 \leftarrow r_4 + \frac{2}{7} r_2 \end{matrix} \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{2}{7} & \frac{4}{7} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \end{bmatrix}.$$

Finally, we permute rows:

$$r_3 \leftrightarrow r_4 \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & -\frac{2}{7} & \frac{4}{7} \end{bmatrix},$$

and we eliminate:

$$r_4 \leftarrow r_4 - \tfrac{1}{3} r_3 \Rightarrow \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & 0 & \frac{2}{3} \end{bmatrix}.$$

In practice, GE with partial pivoting is a numerically stable algorithm, though there are academic examples where rounding errors increase exponentially quickly as a function of the number of unknowns in the linear system. Such examples, fortunately, rarely occur in practice. There is a more numerically stable algorithm, known as GE with full pivoting. In this algorithm, at the j^{th} pass of the process, we choose the largest entry in the $(n-j+1) \times (n-j+1)$ submatrix to be the pivot. GE with full pivoting is rarely used in practice due to the extra complexity involved in searching for the pivot, especially since partial pivoting is usually quite adequate.

2.2 Taking Advantage of Structure I

2.2.1 Triangular Matrices

Definition 2.2.1. *A* lower (upper) triangular matrix *is a square matrix whose entries above (below) the main diagonal are all zero. If $L \in \mathbb{C}^{n\times n}$ is a lower triangular matrix and $U \in \mathbb{C}^{n\times n}$ is an upper triangular matrix:*

$$L = \begin{bmatrix} L_{1,1} & & & 0 \\ L_{2,1} & L_{2,2} & & \\ \vdots & \vdots & \ddots & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} \end{bmatrix}, \quad \text{and} \quad U = \begin{bmatrix} U_{1,1} & U_{1,2} & \cdots & U_{1,n} \\ & U_{2,2} & \cdots & U_{2,n} \\ & & \ddots & \vdots \\ 0 & & & U_{n,n} \end{bmatrix}.$$

A lower (upper) triangular matrix is strictly lower (upper) triangular *if the diagonal entries are zero. Any matrix that is simultaneously upper and lower triangular is* diagonal.

Triangular matrices have a many properties that make them convenient to work with. For example, the (conjugate) transpose of a lower (upper) triangular matrix is an upper (lower) triangular matrix. As well, recall that the determinant of a triangular matrix is the product of the diagonal entries of the matrix. Similarly, the diagonal entries of a triangular matrix are the eigenvalues of the matrix.

Algorithm 2.2.2 (Forward substitution)**.** *Let $L \in \mathbb{C}^{n\times n}$ be a lower triangular matrix, and let $y, b \in \mathbb{C}^n$. If $Ly = b$, then:*

1. $y_1 = L_{1,1}^{-1} b_1$*; and for* $i = 2, \ldots, n$,
2. $y_i = L_{i,i}^{-1}\left(b_i - \sum_{j=1}^{i-1} L_{i,j} y_j\right)$.

Algorithm 2.2.3 (Backward substitution)**.** *Let $U \in \mathbb{C}^{n\times n}$ be an upper triangular matrix, and let $y, b \in \mathbb{C}^n$. If $Uy = b$, then:*

1. $y_n = U_{n,n}^{-1} b_n$*; and for* $i = n-1, \ldots, 1$,
2. $y_i = U_{i,i}^{-1}\left(b_i - \sum_{j=i+1}^{n} U_{i,j} y_j\right)$.

Using forward (backward) substitution, the inverse of a lower (upper) triangular matrix can be applied to a vector in $n^2 + \mathcal{O}(n)$ operations, which is faster than Gaussian elimination for an unstructured matrix. Clearly, a diagonal matrix can be inverted in n operations.

Example 2.2.4. *If* $\begin{bmatrix} 1 & 0 & 0 \\ 2 & 3 & 0 \\ 4 & 8 & 3 \end{bmatrix} y = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$*, then:*

$$\begin{aligned} y_1 &= 1, \\ y_2 &= \frac{1}{3}(2 - 2\cdot 1) = 0, \\ y_3 &= \frac{1}{3}(3 - 4\cdot 1 - 8\cdot 0) = -\frac{1}{3}. \end{aligned}$$

Definition 2.2.5. *A* unit *lower (upper) triangular matrix is a lower (upper) triangular matrix whose diagonal entries are all* 1.

2.2.2 Permutation Matrices

Definition 2.2.6. *Let p be a permutation of the set $\{1, \ldots, n\}$. A* row permutation *matrix $P_n^r \in \mathbb{R}^{n\times n}$ is a matrix whose rows are the p^{th} rows of the identity $P_n^r = I_n[p, :]$. Similarly, a* column permutation *matrix $P_n^c \in \mathbb{R}^{n\times n}$ is a matrix whose columns are the p^{th} columns of the identity $P_n^c = I_n[:, p]$.*

Row permutation matrices applied to the left of a matrix $A \in \mathbb{C}^{n\times n}$ permute the rows of the matrix A according to the permutation p. Column permutation matrices applied to the right of a matrix permute the columns of the matrix A according to the permutation p. In other words:

$$P_n^r A = A[p, :], \quad \text{and} \quad A P_n^c = A[:, p].$$

Row and column permutation matrices corresponding to the same permutation p are inverses, and indeed, $P_n^{r,-1} = P_n^{r,\top} = P_n^c$ and $P_n^{c,-1} = P_n^{c,\top} = P_n^r$.

2.2.3 Unitary (and Orthogonal) Matrices

Definition 2.2.7. *A* unitary matrix $Q \in \mathbb{C}^{n\times n}$ *is a matrix whose rows and columns are all mutually orthonormal. In other words:*

$$QQ^* = Q^*Q = I.$$

An orthogonal matrix $Q \in \mathbb{R}^{n\times n}$ *is a unitary matrix with real entries. In this case,* $QQ^\top = Q^\top Q = I$.

Unitary matrices are an important class of matrices because they will enable us to derive new types of matrix factorizations with better conditioning. Indeed, unitary matrices can be viewed as the class of matrices whose conjugate transpose is the inverse, $Q^{-1} = Q^*$, and specializing on the class of orthogonal matrices, $Q^{-1} = Q^\top$. This simple identification allows for $\mathcal{O}(n^2)$ (fast) inversion via matrix-vector multiplication.

Lemma 2.2.8. *For any unitary matrix* $Q \in \mathbb{C}^{n\times n}$, $\|Q\|_2 = 1$.

Proof. We prove that for any vector $x \in \mathbb{C}^n$, $\|Qx\|_2 = \|x\|_2$. The result then follows by the definition of the induced 2-norm:

$$\|Q\|_2 = \sup_{\substack{x\in\mathbb{C}^n \\ x\neq 0}} \frac{\|Qx\|_2}{\|x\|_2}.$$

Since the space ℓ^2 is an inner product space, we identify the square of the 2-norm with the inner product:

$$\|Qx\|_2^2 = \langle Qx, Qx\rangle = (Qx)^* Qx = x^*Q^*Qx = x^*Ix = x^*x = \langle x, x\rangle = \|x\|_2^2.$$

□

This lemma is very powerful because it allows us to prove the following theorem regarding the 2-norm condition number of unitary matrices.

Theorem 2.2.9. *For any unitary matrix* $Q \in \mathbb{C}^{n\times n}$, $\mathrm{cond}_2(Q) = 1$.

Proof. By definition:

$$\mathrm{cond}_2(Q) = \|Q\|_2 \left\|Q^{-1}\right\|_2 = \|Q\|_2 \|Q^*\|_2.$$

Since Q^* is also a unitary matrix, it follows from Lemma 2.2.8 that $\|Q\|_2 = 1$ and $\|Q^*\|_2 = 1$. □

Unitary matrices have a host of other properties which make them excellent to work with. Lemma 2.2.8 has a geometrical interpretation of unitary matrices. Since $\|Qx\|_2 = \|x\|_2$, the vector x is *neither* expanded nor contracted by the transformation Q. This allows us to identify Q with a change of basis that is simply a rotation of the coordinates in $\mathbb{C}^n$.

Two classes of unitary matrices can be derived from this geometrical interpretation. The first class is known as Givens rotation matrices. The second class is known as Householder reflector matrices. They are named after the mathematicians who identified them first.

Example 2.2.10. *Let $\theta \in [0, 2\pi)$, let $c = \cos\theta$ and $s = \sin\theta$, and let $i, j \in \{1, \ldots, n\}$. The $n \times n$* Givens matrix*:*

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

where the first c appears at the intersection of the i^{th} row and column, and the second c appears at the intersection of the j^{th} row and column, is an orthogonal matrix. This can be observed by identifying $G(i,j,\theta)$ as a clockwise rotation of a vector in the i and j coordinates by the angle θ. It can also be proved by multiplying by $G(i,j,\theta)^\top$ and recovering the identity.

Example 2.2.11. *Let $0 \neq w \in \mathbb{R}^n$. The $n \times n$* Householder matrix*:*

$$H(w) := I - \frac{2}{\langle w, w\rangle} w w^\top,$$

is an orthogonal matrix. This follows from:

$$\begin{aligned} H(w)H(w)^\top &= \left(I - \frac{2}{\langle w,w\rangle} w w^\top\right)\left(I - \frac{2}{\langle w,w\rangle} w w^\top\right), \\ &= I - \frac{4}{\langle w,w\rangle} w w^\top + \frac{4}{\langle w,w\rangle^2} w(w^\top w) w^\top, \\ &= I. \end{aligned}$$

Householder matrices can be identified as reflections about the hyperplane defined by $w \cdot x = 0$.

Why are the Givens and Householder matrices so important? Due to the following theorem, we can express *any* orthogonal matrix Q as a product of Givens and/or Householder matrices.

Theorem 2.2.12. *The product of unitary (orthogonal) matrices is unitary (orthogonal).*

2.3 Matrix Factorizations

2.3.1 LU Factorization

The basic operation of the algorithm of Gaussian elimination is the replacement of row i with row i plus some constant multiple of row j. The computational complexity of Gaussian elimination consists of $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ for the operations on the matrix A, and $n^2 + \mathcal{O}(n)$ for the operations on the vector b. Suppose we are required to calculate the solution of both $Ax = b$ and $Ax = c$. It would be ideal to save as much work as possible in the first "solve" so that successive solves can be computed by only operating

on the input data $(b, c, \ldots)$. This is achieved by the LU factorization of the matrix A, where L is a lower triangular matrix, and U is an upper triangular matrix:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} = \begin{bmatrix} L_{1,1} & & & 0 \\ L_{2,1} & L_{2,2} & & \\ \vdots & \vdots & \ddots & \\ L_{n,1} & L_{n,2} & \cdots & L_{n,n} \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & \cdots & U_{1,n} \\ & U_{2,2} & \cdots & U_{2,n} \\ & & \ddots & \vdots \\ 0 & & & U_{n,n} \end{bmatrix}. \tag{2.7}$$

As there are $\sum_{l=1}^{n} l = \frac{n(n+1)}{2}$ unknowns in L and similarly in U, the LU factorization of a matrix A is not unique. In order to enforce uniqueness, we factorize A into a *unit* lower triangular matrix L and an upper triangular matrix U, removing the ambiguity:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & A_{n,n} \end{bmatrix} = \begin{bmatrix} 1 & & & 0 \\ L_{2,1} & 1 & & \\ \vdots & \vdots & \ddots & \\ L_{n,1} & L_{n,2} & \cdots & 1 \end{bmatrix} \begin{bmatrix} U_{1,1} & U_{1,2} & \cdots & U_{1,n} \\ & U_{2,2} & \cdots & U_{2,n} \\ & & \ddots & \vdots \\ 0 & & & U_{n,n} \end{bmatrix}. \tag{2.8}$$

By expanding the matrix-matrix product, it is clear that $A_{1,1} = 1 \cdot U_{1,1}$, and thus $U_{1,1}$ is readily determined. We could continue this process, but we want to draw analogy to GE. Note that in each step of GE, we seek row transformations in order to introduce zeros below the column pivot. For every column, we can accumulate all this work into a special unit lower triangular matrix. Let $v \in \mathbb{C}^{n-j}$, and consider the matrix:

$$M_n(v) := \begin{bmatrix} 1 & & 0 & & & 0 \\ & \ddots & & & & \\ 0 & & 1 & & & \\ & & & 1 & & \\ & & & v_1 & 1 & 0 \\ & & & \vdots & & \ddots \\ 0 & & & v_{n-j} & 0 & 1 \end{bmatrix}. \tag{2.9}$$

This matrix is the unit lower triangular matrix with the entries of the vector v in the j^{th} column below the main diagonal. This class of matrices is useful in storing the multipliers for every row in each iteration of GE.

Example 2.3.1. *In Example 2.1.3, we solved the system:*

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix},$$

by GE. The row operations were:

$$\begin{aligned} r_2 &\leftarrow r_2 - \tfrac{1}{3} r_1 \\ r_3 &\leftarrow r_3 - \tfrac{2}{3} r_1 \end{aligned},$$

followed by:

$$r_3 \leftarrow r_3 + \tfrac{4}{7} r_2.$$

These row operations can be represented with the M matrices as:

$$M_3((-\tfrac{1}{3},-\tfrac{2}{3})^\top) = \begin{bmatrix} 1 & & 0 \\ -\frac{1}{3} & 1 & \\ -\frac{2}{3} & & 1 \end{bmatrix},$$

and:

$$M_3((\tfrac{4}{7})^\top) = \begin{bmatrix} 1 & & 0 \\ & 1 & \\ & \frac{4}{7} & 1 \end{bmatrix}.$$

The class of M matrices satisfies a couple powerful properties.

Lemma 2.3.2. *Let $v \in \mathbb{C}^{n-j}$. Then:*

$$M_n(v)^{-1} = M_n(-v). \tag{2.10}$$

Lemma 2.3.3. *Let $u \in \mathbb{C}^{n-j}$ and let $v \in \mathbb{C}^{n-k}$ where $j < k$. Then:*

$$M_n(u)M_n(v) = M_n(u) + M_n(v) - I. \tag{2.11}$$

These properties allow us to accumulate the multipliers in GE into a product of unit lower triangular matrices, whose inverses are known. The products of these inverses are then accumulated into a single unit lower triangular matrix.

Example 2.3.4. *In Example 2.1.3, we solved the system:*

$$\begin{bmatrix} 3 & -1 & 2 \\ 1 & 2 & 3 \\ 2 & -2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 12 \\ 11 \\ 2 \end{bmatrix},$$

by GE. The row operations were:

$$M_3((-\tfrac{1}{3},-\tfrac{2}{3})^\top) = \begin{bmatrix} 1 & & 0 \\ -\frac{1}{3} & 1 & \\ -\frac{2}{3} & & 1 \end{bmatrix},$$

followed by:

$$M_3((\tfrac{4}{7})^\top) = \begin{bmatrix} 1 & & 0 \\ & 1 & \\ & \frac{4}{7} & 1 \end{bmatrix}.$$

Thus,

$$M_3((\tfrac{4}{7})^\top)M_3((-\tfrac{1}{3},-\tfrac{2}{3})^\top)A = \begin{bmatrix} 3 & -1 & 2 \\ 0 & \frac{7}{3} & \frac{7}{3} \\ 0 & 0 & -1 \end{bmatrix} = U.$$

By successively inverting the M matrices and accumulating the result:

$$A = \begin{bmatrix} 1 & & \\ \frac{1}{3} & 1 & \\ \frac{2}{3} & -\frac{4}{7} & 1 \end{bmatrix} \begin{bmatrix} 3 & -1 & 2 \\ & \frac{7}{3} & \frac{7}{3} \\ & & -1 \end{bmatrix} = LU.$$

The LU factorization of A carries the same complexity[2] as GE with respect to operations on the matrix A, but does not include any operations with an input vector b. When we want to solve a system $Ax = b$, we can:

1. Compute the factorization $A = LU$ in $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ operations;
2. Solve the system $Ly = b$ via forward substitution in $n^2 + \mathcal{O}(n)$ operations; and,
3. Solve the system $Ux = y$ via backward substitution in $n^2 + \mathcal{O}(n)$ operations.

We know that GE is numerically stable with partial pivoting, and in this case, we would also like to store pivot information in the LU factorization.

Example 2.3.5. *In Example 2.1.4, we applied GE with partial pivoting to* $\begin{bmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{bmatrix}$. *The row permutations and eliminations can be written succinctly as:*

$$M_4((-\tfrac{1}{3})^\top) I_4[(1,2,4,3)^\top, :] M_4((\tfrac{3}{7}, \tfrac{2}{7})^\top) I_4[(1,4,3,2)^\top, :]$$
$$\times M_4((-\tfrac{1}{2}, -\tfrac{1}{4}, -\tfrac{3}{4})^\top) I_4[(3,2,1,4)^\top, :] A = \begin{bmatrix} 8 & 7 & 9 & 5 \\ 0 & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ 0 & 0 & -\frac{6}{7} & -\frac{2}{7} \\ 0 & 0 & 0 & \frac{2}{3} \end{bmatrix} = U.$$

If we invert the multiplication and permutation matrices, we find:

$$A = I_4[(3,2,1,4)^\top, :]^\top M_4((\tfrac{1}{2}, \tfrac{1}{4}, \tfrac{3}{4})^\top) I_4[(1,4,3,2)^\top, :]^\top M_4((-\tfrac{3}{7}, -\tfrac{2}{7})^\top) I_4[(1,2,4,3)^\top, :]^\top M_4((\tfrac{1}{3})^\top) U$$

If we carry out the matrix-matrix multiplications, then we find:

$$A = \begin{bmatrix} \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \\ \frac{1}{2} & -\frac{2}{7} & 1 & 0 \\ 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \end{bmatrix} U,$$

The rows of the matrix multiplying U can then be permuted[3] *such that it is unit lower triangular:*

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \\ \frac{1}{2} & -\frac{2}{7} & 1 & 0 \\ \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \end{bmatrix}}_{L} = \underbrace{\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}}_{P} \begin{bmatrix} \frac{1}{4} & -\frac{3}{7} & \frac{1}{3} & 1 \\ \frac{1}{2} & -\frac{2}{7} & 1 & 0 \\ 1 & 0 & 0 & 0 \\ \frac{3}{4} & 1 & 0 & 0 \end{bmatrix}.$$

Thus, $PA = LU$.

Similarly to the LU factorization, the LUP factorization can be used to solve linear systems as follows:

1. Compute the factorization $PA = LU$ in $\frac{2}{3}n^3 + \mathcal{O}(n^2)$ operations;
2. Solve the system $Ly = Pb$ via forward substitution in $n^2 + \mathcal{O}(n)$ operations; and,
3. Solve the system $Ux = y$ via backward substitution in $n^2 + \mathcal{O}(n)$ operations.

[2] After all, it is just an organization of the arithmetic.

[3] This much is straightforward: one need only organize the ones such that they are on the diagonal.

2.3.2 QR Factorization

We have seen that unitary and orthogonal matrices have interesting properties that make them useful. In this section, we combine the fact that the product of unitary (orthogonal) matrices is unitary (orthogonal) and the fact that unitary (orthogonal) matrices have an $\mathcal{O}(n^2)$ inverse to consider another factorization of a square matrix $A \in \mathbb{C}^{n\times n}$. Such a factorization is the QR factorization, where Q is a unitary matrix and R is an upper triangular matrix. Due to the added degrees of freedom, QR factorizations are not unique[4]; however, we will assert existence of the factorization by constructions. The main advantage of the QR factorization is that, viewed as an algorithm for solving linear systems, it is a well-conditioned algorithm since unitary (orthogonal) matrices have 2-norm condition number 1.

Theorem 2.3.6. *Given a square matrix $A \in \mathbb{C}^{n\times n}$, there exists a unitary matrix Q and an upper triangular matrix R such that:*

$$A = QR.$$

Without loss of generality, we will focus on real square matrices $A \in \mathbb{R}^{n\times n}$ with orthogonal QR factorizations[5].

Construction via Givens Matrices

Lemma 2.3.7. *Given $u \in \mathbb{R}^n$, there exists a $\theta \in [0, 2\pi)$ such that:*

$$G(i,j,\theta)u = \begin{bmatrix} u_1 \\ \vdots \\ u_{i-1} \\ r \\ u_{i+1} \\ \vdots \\ u_{j-1} \\ 0 \\ u_{j+1} \\ \vdots \\ u_n \end{bmatrix} =: v,$$

say, where $r = \sqrt{u_i^2 + u_j^2}$.

[4]Non-uniqueness is irrelevant for the QR factorization.

[5]The technical details of the unitary QR factorization for $A \in \mathbb{C}^{n\times n}$ needlessly complicate the matter.

Proof. Recall that the Givens matrix is:

$$G(i,j,\theta) = \begin{bmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & & \vdots & & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & & \vdots & \ddots & \vdots & & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & & \vdots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{bmatrix},$$

Multiplying through, we note that the Givens matrix leaves every entry of u untouched except the i^{th} and the j^{th} entries:

$$G(i,j,\theta)u = \begin{bmatrix} u_1 \\ \vdots \\ u_{i-1} \\ cu_i + su_j \\ u_{i+1} \\ \vdots \\ u_{j-1} \\ -su_i + cu_j \\ u_{j+1} \\ \vdots \\ u_n \end{bmatrix} = \begin{bmatrix} u_1 \\ \vdots \\ u_{i-1} \\ r \\ u_{i+1} \\ \vdots \\ u_{j-1} \\ 0 \\ u_{j+1} \\ \vdots \\ u_n \end{bmatrix}.$$

From the j^{th} equation, we know that $\tan\theta = \dfrac{u_j}{u_i}$, or equivalently:

$$s = \frac{u_j}{\sqrt{u_i^2 + u_j^2}}, \quad \text{and} \quad c = \frac{u_i}{\sqrt{u_i^2 + u_j^2}}.$$

□

Since there *always* exists an angle θ by which we can "rotate" the vector u in order to introduce a zero in entry j, we will use the notation $G(i,j)$ to denote the Givens matrix that uses entry i to introduce a zero in entry j.

Now, if u denotes the first column of $A \in \mathbb{R}^{n\times n}$, apply the Givens rotation to "zero" the $n,1$ entry of A using the $n-1,1$ entry:

$$G(n-1,n)A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{A_{n-1,1}^2 + A_{n,n}^2} & \times & \cdots & \times \\ 0 & \times & \cdots & \times \end{bmatrix},$$

where the $\times$ denotes a general, modified entry. Continue by introducing a zero in the $n-1,1$ entry of the modified matrix using the $n-2,1$ entry:

$$G(n-2,n-1)G(n-1,n)A = \begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ \sqrt{A_{n-2,1}^2 + A_{n-1,1}^2 + A_{n,n}^2} & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ 0 & \times & \cdots & \times \end{bmatrix}.$$

Thus it is clear that we may introduce zeros in the $n-1$ entries below the $1,1$ entry, zeros in the $n-2$ entries below the $2,2$ entry, and so on and so forth until we have introduced zeros in every position below the main diagonal:

$$\underbrace{G(n-1,n)\cdot\cdots\cdot G(2,3)\cdots G(n-1,n)G(1,2)\cdots G(n-1,n)}_{Q^\top} A = \begin{bmatrix} \times & \times & \cdots & \times \\ 0 & \times & \cdots & \times \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \times \end{bmatrix} =: R.$$

Construction via Householder Matrices

Lemma 2.3.8. *Given $u \in \mathbb{R}^n$, there exists a $w \in \mathbb{R}^n$ such that:*

$$H(w)u = \begin{bmatrix} \alpha \\ 0 \\ \vdots \\ 0 \end{bmatrix} =: v,$$

say, where $\alpha = \pm\sqrt{\langle u, u\rangle}$.

Proof. Let $w = \gamma(u-v)$ where $\gamma \neq 0$. Recall that $\langle u, u\rangle = \langle v, v\rangle$. Thus:

$$\begin{aligned} w^\top w &= \gamma^2(u-v)^\top(u-v) = \gamma^2(u^\top u - u^\top v - v^\top u + v^\top v), \\ &= \gamma^2(2u^\top u - 2u^\top v) = 2\gamma^2 u^\top(u-v) = 2\gamma u^\top w = 2\gamma w^\top u. \end{aligned}$$

Let us apply $H(w)$ to u:

$$\begin{aligned} H(w)u &= \left(I - \frac{2}{w^\top w} w w^\top\right) u, \\ &= u - \frac{2}{2\gamma w^\top u} w w^\top u. \end{aligned}$$

Here, we can cancel the factor $2w^\top u$ from the numerator and denominator to obtain:

$$H(w)u = u - \frac{1}{\gamma} w = u - (u-v) = v.$$

□

Lemma 2.3.8, is useful in showing that Householder matrices can be used to introduce as many as $n-1$ *zeros* in a vector of length n. Now, if u is the first column of the n-by-n matrix A, then let:

$$H(w_1)A = \left[\begin{array}{c|ccc} \alpha & \times & \cdots & \times \\ \hline 0 & & & \\ \vdots & & B & \\ 0 & & & \end{array}\right].$$

The resulting matrix has zeros introduced below the first entry, which is now α, $\times$ denotes a general entry of the matrix, and where $B \in \mathbb{R}^{n-1\times n-1}$. Similarly for B, we can find $\hat{w} \in \mathbb{R}^{n-1}$ such that:

$$H(\hat{w})B = \left[\begin{array}{c|ccc} \beta & \times & \cdots & \times \\ \hline 0 & & & \\ \vdots & & C & \\ 0 & & & \end{array}\right].$$

Since $H(\hat{w})$ is orthogonal, we can embed it in the identity matrix of size n-by-n:

$$\left[\begin{array}{c|ccc} 1 & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & H(\hat{w}) & \\ 0 & & & \end{array}\right] =: H(w_2), \quad \text{where} \quad w_2 = \begin{bmatrix} 0 \\ \hat{w} \end{bmatrix}.$$

Then:

$$H(w_2)H(w_1)A = \left[\begin{array}{cc|ccc} \alpha & \times & \times & \cdots & \times \\ 0 & \beta & \times & \cdots & \times \\ \hline 0 & 0 & & & \\ \vdots & \vdots & & C & \\ 0 & 0 & & & \end{array}\right].$$

Continuing in this manner for the $n-1$ steps, we obtain:

$$\underbrace{H(w_{n-1})\cdots H(w_2)H(w_1)}_{Q^\top} A = \begin{bmatrix} \alpha & \times & \cdots & \times \\ 0 & \beta & \cdots & \times \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \omega \end{bmatrix} =: R, \tag{2.12}$$

an upper triangular matrix! Since the product of orthogonal matrices is also an orthogonal matrix, we have constructed the QR factorization of $A = QR$.

Givens Matrices vs. Householder Matrices

In certain scenarios, there are advantages to using either the Givens matrices or the Householder matrices to form the QR factorization of A. Givens rotations are efficient to use on a parallel computer because every Givens rotation only modifies two rows of the matrix A. Therefore, a large matrix can be distributed between processors and Givens rotations can be performed in parallel. In contrast, Householder matrices are very efficient on a serial computer due to low-level optimization of code such as inner products in Basic Linear Algebra Subprograms (BLAS).

Remark 2.3.9. Recall that the 2-norm condition number of a square matrix $A \in \mathbb{C}^{n\times n}$, $\text{cond}_2(A) = \|A\|_2 \|A^{-1}\|_2$, provides a least upper bound on the relative errors we can expect in matrix-vector multiplication and solution of linear systems. Furthermore, in Remark 1.3.8, we distinguished between the condition number of the *problem* and the condition number of the *algorithm* we use to solve the problem. We considered a factorization $A = BC$ as a useful algorithm for linear systems, only if the matrices B and C can be used efficiently to:

1. Solve for $y := B^{-1}b$; and,
2. Solve for $x = C^{-1}y$.

Using matrix inequalities, we compared condition numbers of the problem and the algorithm:

$$\begin{aligned}\text{cond}_p(A) = \text{cond}_p(BC) &= \|BC\|_p \left\|(BC)^{-1}\right\|_p = \|BC\|_p \left\|C^{-1}B^{-1}\right\|_p, \\ &\leq \|B\|_p \|C\|_p \left\|C^{-1}\right\|_p \left\|B^{-1}\right\|_p = \text{cond}_p(B)\,\text{cond}_p(C),\end{aligned}$$

concluding that the condition number of the algorithm is *at least as large* as the condition number of the problem. This is certainly true for the LU factorization of a matrix, but for the QR, we can do better. Consider comparing the following condition numbers:

$$\begin{aligned}\text{cond}_2(R) = \text{cond}_2(Q^*A) &= \|Q^*A\|_2 \left\|(Q^*A)^{-1}\right\|_2 = \|Q^*A\|_2 \left\|A^{-1}Q\right\|_2, \\ &\leq \|Q^*\|_2 \|A\|_2 \left\|A^{-1}\right\|_2 \|Q\|_2 = \|A\|_2 \left\|A^{-1}\right\|_2 = \text{cond}_2(A).\end{aligned}$$

On the one hand it follows from the generic inequality for *any* factorization that:

$$\text{cond}_2(A) \leq \text{cond}_2(R),$$

and from the inequality just derived:

$$\text{cond}_2(R) \leq \text{cond}_2(A).$$

This can be true if and only if $\text{cond}_2(A) = \text{cond}_2(R)$. Since the QR factorization has exactly the same condition number as the problem, it is a well-conditioned algorithm for solving linear systems.

Least Squares Approximation

By the same upper triangularization procedures of Givens and Householder, given a matrix $A \in \mathbb{C}^{m\times n}$ with $m > n$, it should not be inconceivable that we can also find a factorization:

$$A = QR,$$

where $Q \in \mathbb{C}^{m\times n}$ has n orthonormal columns and where $R \in \mathbb{C}^{n\times n}$ is upper triangular. This is known as the *reduced* QR factorization.

The reduced QR factorization can be used to solve the least squares problem $Ax = b$, where $A \in \mathbb{C}^{m\times n}$ with $m > n$ is the matrix of an over-determined system. Recall that the least squares solution is given by:

$$A^*Ax = A^*b.$$

However, forming the matrix-matrix product A^*A is costly and can double the relative size of the matrix, risking doubling the loss of accuracy. Alternatively, let $A = QR$ be the reduced QR factorization of A. Then:

$$\begin{aligned}(QR)^*QRx &= (QR)^*b,\\ R^*Q^*QRx &= R^*Q^*b,\\ Q^*QRx &= Q^*b.\end{aligned}$$

Since Q has orthonormal columns, $Q^*Q = I_n$, and thus the solution of the linear system via the reduced QR factorization, $x = R^{-1}Q^*b$, is *equivalent* to the least squares approximation, and avoids the formation of A^*A.

2.3.3 Spectral Decomposition

A square matrix $A \in \mathbb{C}^{n\times n}$ may transform some vectors $v \in \mathbb{C}^n$ only by stretching them, that is:

$$Av = \lambda v, \tag{2.13}$$

for some $\lambda \in \mathbb{C}$. Such vectors are known as *eigenvectors* of A and the factors λ are known as *eigenvalues*. To find all the eigenvalues of a matrix, we rewrite (2.13) as the homogeneous system:

$$(A - \lambda I)v = 0. \tag{2.14}$$

To ensure that we do not obtain the trivial "zero" eigenvector solution, we assert that $\det(A - \lambda I) = 0$, guaranteeing that the linear system is singular. The determinant then defines the degree-n *characteristic polynomial* of A:

$$p(\lambda) := \det(A - \lambda I).$$

By the fundamental theorem of algebra, we know that $p(\lambda)$ has exactly n roots in the complex plane, though they may not be distinct. Let $\{\lambda_i\}_{i=1}^n$ denote the set of solutions of the characteristic equation $p(\lambda) = 0$. The set $\{\lambda_i\}_{i=1}^n$ is also known as the *spectrum* of the matrix A.

Certain matrices allow for a canonical decomposition in terms of all of its eigenvalues and eigenvectors.

Definition 2.3.10. *Let $A \in \mathbb{C}^{n\times n}$ be a square matrix with n linearly independent eigenvectors $v_1, \ldots, v_n$. Then A can be factored as:*

$$A = V\Lambda V^{-1},$$

where the columns of V are the eigenvectors $v_1, \ldots, v_n$ and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$.

If A admits the *spectral decomposition*, or *eigendecomposition*, described above then we say that A is *diagonalizable*. Generally speaking, not all matrices are diagonalizable. However, under certain conditions on A, the spectral decomposition can be described in terms of orthogonal or unitary matrices V.

Definition 2.3.11. *A* symmetric *matrix $A \in \mathbb{R}^{n\times n}$ is a matrix that satisfies $A = A^\top$, or $A_{i,j} = A_{j,i}$ for every $i, j = 1, \ldots, n$.*

Definition 2.3.12. *An* Hermitian *matrix $A \in \mathbb{C}^{n\times n}$ is a matrix that satisfies $A = A^*$, or $A_{i,j} = \overline{A_{j,i}}$ for every $i, j = 1, \ldots, n$.*

Lemma 2.3.13. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix:*

1. *Eigenvalues of A are real;*
2. *Eigenvectors of A are real; and,*
3. *Eigenvectors of distinct eigenvalues of A are orthogonal.*

Proof. We prove the results in order.

1. Let $v \in \mathbb{C}^n$ be an eigenvector and let $\lambda \in \mathbb{C}$ be an eigenvalue, $Av = \lambda v$. Taking complex conjugates:

$$\begin{aligned} \overline{Av} &= \overline{\lambda v}, \\ &= \overline{A}\overline{v} = A\overline{v} = \overline{\lambda}\overline{v}. \end{aligned} \tag{2.15}$$

   Next, apply $\overline{v}^\top$ to the eigenvalue equation to obtain:

$$\begin{aligned} \overline{v}^\top Av &= \overline{v}^\top \lambda v, \\ = \underbrace{(A^\top \overline{v})^\top v}_{\text{by } x^\top A = (A^\top x)^\top} = \underbrace{(A\overline{v})^\top v}_{\text{by symmetry}} = \underbrace{(\overline{\lambda}\overline{v})^\top v}_{\text{by (2.15)}} &= \overline{\lambda}\overline{v}^\top v = \lambda \overline{v}^\top v. \end{aligned}$$

   We conclude that:

$$(\lambda - \overline{\lambda})\langle v, v\rangle = 0.$$

   Since v is an eigenvector of A, its inner product with itself cannot be zero and thus $\lambda = \overline{\lambda}$.

2. The eigenvector $v \in \ker(A - \lambda I)$, where $A - \lambda I \in \mathbb{R}^{n\times n}$. Vectors in the kernel of real homogeneous linear systems are real.

3. Let $Av_1 = \lambda_1 v_1$ and $Av_2 = \lambda_2 v_2$, with λ_1, λ_2 distinct. Apply $v_1^\top$ to the second eigenvalue equation to obtain:

$$\begin{aligned} v_1^\top Av_2 &= v_1^\top \lambda_2 v_2, \\ = (A^\top v_1)^\top v_2 = (Av_1)^\top v_2 = (\lambda_1 v_1)^\top v_2 = \lambda_1 v_1^\top v_2 &= \lambda_2 v_1^\top v_2. \end{aligned}$$

   We conclude that:

$$(\lambda_1 - \lambda_2)\langle v_1, v_2\rangle = 0.$$

   Since $\lambda_1 \neq \lambda_2$, $\langle v_1, v_2\rangle = 0$.

□

Theorem 2.3.14. *Let $A \in \mathbb{R}^{n\times n}$ be a symmetric matrix. Then, the spectral decomposition of A can be described in terms of orthonormal eigenvectors $q_1, \ldots, q_n$ and real eigenvalues:*

$$A = Q\Lambda Q^\top,$$

where $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix whose columns are the orthonormal eigenvectors $q_1, \ldots, q_n$ and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$.

Proof. By Lemma 2.3.13, it is clear that eigenvalues and eigenvectors of A are real, and eigenvectors of distinct eigenvalues are orthogonal. If an eigenvalue λ has multiplicity m, we can always find a set of m orthogonal eigenvectors associated with λ. The result follows from orthonormalization. □

The following theorem is the complex analogue of Theorem 2.3.14.

Theorem 2.3.15. *Let $A \in \mathbb{C}^{n\times n}$ be an Hermitian matrix. Then, the spectral decomposition of A can be described in terms of orthonormal eigenvectors $q_1, \ldots, q_n$ and real eigenvalues:*

$$A = Q\Lambda Q^*,$$

where $Q \in \mathbb{C}^{n\times n}$ is a unitary matrix whose columns are the orthonormal eigenvectors $q_1, \ldots, q_n$, and $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n) \in \mathbb{R}^{n\times n}$.

2.3.4 Singular Value Decomposition

Not every square matrix is diagonalizable. In fact, it is not so complicated to construct a counter-example:

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

This matrix has the eigenvalue 1 with multiplicity 2, and it is *impossible* to construct linearly independent eigenvectors.

However, there are certain instances in which *another* canonical decomposition is useful. This other decomposition is also applicable more generally to rectangular matrices as well.

Theorem 2.3.16. *Let $A \in \mathbb{C}^{m\times n}$. Then, the* singular value *decomposition of A is:*

$$A = U\Sigma V^*,$$

where $U \in \mathbb{C}^{m\times m}$ and $V \in \mathbb{C}^{n\times n}$ are unitary matrices and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with non-increasing singular values *$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min\{m,n\}}$. The columns of U are known as left singular vectors, while the columns of V are known as right singular vectors.*

Sketch of Proof. The matrix $AA^* \in \mathbb{C}^{m\times m}$ is Hermitian, i.e. $(AA^*)^* = A^{**}A^* = AA^*$. Therefore, by Theorem 2.3.15, it has the spectral decomposition:

$$AA^* = U\Lambda_1 U^*.$$

Similarly, $A^*A \in \mathbb{C}^{n\times n}$ is also Hermitian and has the spectral decomposition:

$$A^*A = V\Lambda_2 V^*.$$

Making no assumptions on the form of Σ, if we take $A = U\Sigma V^*$, then:

$$AA^* = U\Sigma V^*(V\Sigma^* U^*) = U\Sigma\Sigma^* U^*,$$
$$A^*A = V\Sigma^* U^*(U\Sigma V^*) = V\Sigma^*\Sigma V^*.$$

Thus, $\Sigma\Sigma^* = \Lambda_1$ and $\Sigma^*\Sigma = \Lambda_2$ are both diagonal matrices. We can choose Σ to be a diagonal matrix, and it turns out that[6] the nonzero eigenvalues of AA^* and A^*A are the *same*, and thus the singular values are just the positive square roots of the nonzero real eigenvalues Λ_1 and Λ_2. □

[6] Omitting parts of the proof beyond the scope of this course.

Theorem 2.3.17. *Let $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$ be the real singular values of $A \in \mathbb{C}^{n\times n}$. Then:*

1. $\|A\|_2 = \sigma_1$;
2. $\left\|A^{-1}\right\|_2 = 1/\sigma_n$; *and,*
3. $\mathrm{cond}_2(A) = \sigma_1/\sigma_n$.

2.4 Taking Advantage of Structure II

Normally, a matrix $A \in \mathbb{C}^{m\times n}$ arising from a particular type of problem will also be endowed with extra structure that will allow either lower storage complexity or faster matrix operations or both. In these cases, it is beneficial to fully describe the structure of the matrix A. Below, we enumerate certain matrix structures that allow for increased performance and solution of linear systems much larger than via dense factorizations.

2.4.1 Symmetric Positive-Definite Matrices

We have already seen the property of symmetry (more generally, the Hermitian property) in proving the spectral theorem for symmetric matrices. When we also add *positive-definiteness* to the matrix A, then even more favourable properties can be derived.

Definition 2.4.1. *Let $A \in \mathbb{C}^{n\times n}$ be an Hermitian matrix. We say that A is* symmetric positive-definite *if $z^*Az > 0$ for every nonzero vector $z \in \mathbb{C}^n$.*

Note that if we restrict our attention to real symmetric matrices, then these are symmetric positive-definite if $x^\top Ax > 0$ for every nonzero vector $x \in \mathbb{R}^n$.

The term positive-definite is derived from the fact that $x^*Ay = \langle x, Ay\rangle =: \langle x, y\rangle_A$ can be used to define an inner product, which has the positive-definite property.

Symmetric positive-definite (SPD) matrices have a unique *Cholesky* factorization, which is a symmetric version of the LU factorization. Thus, $A \in \mathbb{R}^{n\times n}$ can be factored into $LL^\top$ and $A \in \mathbb{C}^{n\times n}$ can be factored into LL^*. The beauty of the SPD property is that *no pivoting*[7] *is required for numerical stability!*

2.4.2 Sparse Matrices

Often, large matrices, such as matrices with dimensions greater than $10,000$, only have a small fraction of their entries nonzero. In this case, it becomes efficient to store three vectors consisting of the indices i, j such that $A_{i,j} \neq 0$ and the nonzero entries themselves. These matrices have fast matrix-vector products and are therefore generally amenable to solution via iterative methods.

[7] In fact, row and/or column pivoting generally destroy the SPD property, so the numerical stability without pivoting is nothing short of a mathematical miracle.

2.4.3 Banded Matrices

Definition 2.4.2. *A matrix $A \in \mathbb{C}^{m\times n}$ is* banded *with lower bandwidth p and upper bandwidth q if:*

$$A_{i,j} = 0, \quad \text{for} \quad i - j > p, \quad \text{or} \quad i - j < -q.$$

In other words:

$$A = \begin{bmatrix} A_{1,1} & \cdots & A_{1,q+1} & 0 & \cdots & 0 \\ \vdots & & & \ddots & \ddots & \vdots \\ A_{p+1,1} & & & & \ddots & 0 \\ 0 & \ddots & & & & A_{m-q,n} \\ \vdots & \ddots & \ddots & & & \vdots \\ 0 & \cdots & 0 & A_{m,n-p} & \cdots & A_{m,n} \end{bmatrix}.$$

A banded *matrix A has bandwidth $r = \max\{p, q\}$.*

Banded matrices can be stored more compactly by only storing the nonzero terms within the bandwidth. Banded matrices also have banded LU and QR factorizations; therefore solution of linear systems of square banded matrices can be performed in $\mathcal{O}(r^2 n)$ operations. Banded matrices are a special class of sparse matrices, where the sparsity pattern is known *a priori*.

Example 2.4.3. *The banded matrix:*

$$A = \begin{bmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ \frac{1}{2} & -1 & 2 & -1 & \\ & \frac{1}{2} & -1 & 2 & -1 \\ & & \frac{1}{2} & -1 & 2 \end{bmatrix},$$

has the banded LU factorization:

$$L = \begin{bmatrix} 1 & & & & \\ -\frac{1}{2} & 1 & & & \\ \frac{1}{4} & -\frac{1}{2} & 1 & & \\ & \frac{1}{3} & -\frac{4}{9} & 1 & \\ & & \frac{1}{3} & -\frac{3}{7} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 2 & -1 & & & \\ & \frac{3}{2} & -1 & & \\ & & \frac{3}{2} & -1 & \\ & & & \frac{14}{9} & -1 \\ & & & & \frac{11}{7} \end{bmatrix}.$$

2.5 Iterative Solvers

Dense factorizations calculate the exact solution (modulo rounding errors) of a linear system in a finite number of steps. A second class of methods, called iterative solvers, usually converge to the exact solution after infinitely many iterations. At first, it may seem odd why one would want to consider iterative solvers at all. But in actuality, there are several good reasons. In many applications requiring the solution of partial differential equations, the linear systems involved are extremely large, say $10^6 \times 10^6$ or larger. A naïve approach involving dense factorizations such as LU and QR requires storage of all $\mathcal{O}(10^{12})$ entries of the entire matrix, since during the factorization process, zero entries in the matrix do not imply zero entries

in the factorization. Currently, no personal computer can store 10^{12} floating-point numbers in double precision in random access memory (RAM), and thus one would need to rely on secondary memory (hard drive storage) which drastically slows down the solution process. Another consideration is that the flop count of the LU factorization is approximately $\mathcal{O}(10^{18})$. Assuming that a personal computer runs at its peak flop rate for the entire calculation, which is roughly $1.5 \times 10^{11}\,\mathrm{flops/s}$, then an LU factorization would require $\mathcal{O}(7 \times 10^6)$ seconds, or approximately 77 days.

Clearly, the above pitfalls concerning the naïve dense factorization of a large linear system are enough to warrant an investigation into iterative solvers. Typically, iterative solvers just need access to a matrix-vector product, which is ideal for the class of sparse matrices since only the nonzero entries need to be stored and the matrix-vector product is fast. Another reason for using iterative solvers is that, sometimes we already have a good guess to the solution. In this case, there is a good chance that iterative solvers will converge to a better solution with just a few corrections. Iterative solvers, however, may not converge for arbitrary non-singular matrices. However, for the class of symmetric positive-definite matrices, it has been shown that certain iterative solvers are *guaranteed* to converge.

Furthermore, to remedy the problem-dependence of iterative solvers, a technique known as *preconditioning* has significantly preoccupied numerical linear algebraists since the first modern iterative solvers were derived.

2.5.1 Jacobi and Gauss–Seidel

Given the linear system $Ax = b$, let B be any non-singular matrix. The system is equivalent to $Bx + (A - B)x = b$, or:

$$x = (I - B^{-1}A)x + B^{-1}b. \tag{2.16}$$

Base on this equation, it is natural to define an iterative method as:

$$x^{(k+1)} = Gx^{(k)} + B^{-1}b, \tag{2.17}$$

where the matrix $G := I - B^{-1}A$ is called the *iteration matrix*. For an efficient iterative method, B should be easy to invert and should be as "close" to A as possible such that $\|G\|$ is small. Unfortunately, these two criteria are conflicting. Considering one extreme scenario where $B = A$, then $I - B^{-1}A = 0$ and only one iteration is necessary to compute the exact solution. However, in that iteration, we solved to original linear system! At the other extreme, if we choose $B = I$, linear solves involving B are trivial, but $I - B^{-1}A = I - A$, is in general the same size as A and the iteration may not converge.

Let $e^{(k)} := x^{(k)} - x$ and let $\rho(G)$ denote the *spectral radius* of the matrix G:

$$\rho(G) := \max_{1 \le i \le n} |\lambda_i|,$$

where λ_i are the eigenvalues of G.

Theorem 2.5.1. *Let G be a diagonalizable matrix, $G = V\Lambda V^{-1}$. Let $x^{(0)} \in \mathbb{R}^n$ be any initial iterate. Then $e^{(k)} \to 0$ iff $\rho(G) < 1$.*

Proof. Subtracting (2.17) from (2.16), we have $e^{(k+1)} = Ge^{(k)}$, or $e^{(k)} = G^k e^{(0)}$. If $\rho(G) \ge 1$, let $e^{(0)}$ be a normalized eigenvector of G with eigenvalue of magnitude $\rho(G)$. Clearly, $e^{(k)}$ does not converge to 0.

On the other hand, suppose $\rho(G) < 1$. Then $G^k = V\Lambda^k V^{-1}$. Since every eigenvalue of Λ has magnitude less than one, $\Lambda^k \to 0$ and hence $G^k \to 0$ which implies that $e^{(k)} \to 0$. □

We are now prepared to discuss two classical iterative methods. Suppose the matrix $A = L + D + U$ is decomposed into a diagonal matrix D, and L and U are strictly lower and upper triangular matrices[8] with entries corresponding to A.

The *Jacobi* iteration takes $B = D$ and thus:

$$x^{(k+1)} = -D^{-1}(L + U)x^{(k)} + D^{-1}b.$$

Suppose $x^{(k)}$ is known. Applying $L + U$ to $x^{(k)}$ requires $\mathcal{O}(n^2)$ operations generically, and applying D^{-1} requires $\mathcal{O}(n)$ operations, requiring $\mathcal{O}(n^2)$ operations per iteration, generically:

$$x_i^{(k+1)} = \frac{1}{A_{i,i}} \left(b_i - \sum_{j \neq i} A_{i,j} x_j^{(k)} \right), \qquad 1 \leq i \leq n.$$

The *Gauss–Seidel* iteration takes $B = L + D$ and thus:

$$x^{(k+1)} = -(L + D)^{-1}Ux^{(k)} + (L + D)^{-1} = (L + D)^{-1}(b - Ux^{(k)}).$$

This scheme is similar to the Jacobi iteration except that the most up-to-date components are used as one sweeps from the first equation to the last equation. Since matrix-vector multiplication and solution of triangular linear systems is required, the complexity is again $\mathcal{O}(n^2)$ operations per iteration, generically:

$$x_i^{(k+1)} = \frac{1}{A_{i,i}} \left(b_i - \sum_{j<i} A_{i,j} x_j^{(k+1)} - \sum_{j>i} A_{i,j} x_j^{(k)} \right), \qquad 1 \leq i \leq n.$$

The complexity of Jacobi and Gauss–Seidel improve when iterating with sparse matrices.

Characterizing convergence of an iterative method by the spectral radius is expensive; therefore, an alternative characterization in terms of *strict diagonal dominance* can determine whether the Jacobi or Gauss–Seidel iterations will converge.

Definition 2.5.2. *A matrix $A \in \mathbb{R}^{n \times n}$ is said to be* strictly diagonally dominant *if for every $1 \leq i \leq n$:*

$$|A_{i,i}| > \sum_{j \neq i} |A_{i,j}| .$$

Theorem 2.5.3. *If $A \in \mathbb{R}^{n \times n}$ is strictly diagonally dominant, then the Jacobi and Gauss–Seidel iterations converge for any initial iterate.*

2.5.2 Conjugate Gradients

Modern iterative methods are not built on the fictitious factorization of A into $B + (A - B)$ as in the classical Jacobi and Gauss–Seidel iterations. In fact, modern iterative methods are built on the fact that matrix-vector multiplication is fast when A is (data-)sparse.

Definition 2.5.4. *Let $r^{(0)} := b - Ax^{(0)}$ be the residual of the initial starting vector $x^{(0)}$. The* Krylov subspace *is defined to be:*

$$K_k(r^{(0)}, A) := \operatorname{span}\{r^{(0)}, Ar^{(0)}, \ldots, A^{k-1}r^{(0)}\}, \quad \text{for} \quad k = 1, 2, \ldots.$$

[8] These L and U should not be confused with the factors in the LU factorization of A.

Krylov subspace methods produce a sequence of vectors $x^{(k)} \in x_0 + K_k(r^{(0)}, A)$, and the amount of work per iteration $k \to k+1$ is roughly equal to that of matrix-vector multiplication.

The first method of this kind, known as the *conjugate gradient* (cg-) method, was proposed by Hestenes and Stiefel [7] for linear systems with a symmetric positive definite matrix. These matrices define a norm:

$$\|z\|_A := \sqrt{z^\top A z},$$

and the cg-method generates a sequence in the Krylov subspace with the minimality property:

$$\left\|x^{(k)} - x\right\|_A = \min_{z \in x^{(0)} + K_k(r^{(0)}, A)} \|z - x\|_A \,, \tag{2.18}$$

where x is the true solution.

While beyond the scope of this course, we list the cg-method here for comparison with the Jacobi and Gauss–Seidel methods. A good reference on the cg- and other Krylov subspace methods is [8, Chap. 8].

Algorithm 2.5.5 (Conjugate-Gradient Method)**.** *Initialize $x^{(0)} \in \mathbb{R}^n$ and set $p^{(0)} := r^{(0)} := b - Ax^{(0)}$.*

For $k = 0, 1, \ldots$:

1. *If $\left\|r^{(k)}\right\|_2 < \epsilon$, set $m := k$ and stop: $x^{(k)}$ is the approximate solution of $Ax = b$. Otherwise,*
2. *Compute:*

$$\begin{aligned}
\alpha_k &:= \frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle p^{(k)}, Ap^{(k)} \rangle}, \\
x^{(k+1)} &:= x^{(k)} + \alpha_k p^{(k)}, \\
r^{(k+1)} &:= r^{(k)} - \alpha_k A p^{(k)}, \\
\beta_k &:= \frac{\langle r^{(k+1)}, r^{(k+1)} \rangle}{\langle r^{(k)}, r^{(k)} \rangle}, \\
p^{(k+1)} &:= r^{(k+1)} + \beta_k p^{(k)}.
\end{aligned}$$

For the implementation of the cg-method, only four vectors, $x^{(k)}$, $r^{(k)}$, $p^{(k)}$, and $Ap^{(k)}$, need to be stored. At each step in the iteration, only one matrix-vector multiplication, $Ap^{(k)}$, is required; the remaining work amounts to the calculation of six inner products in $\mathbb{R}^n$. The total computational effort, for sparse matrices, is therefore modest.

2.5.3 Preconditioning

The convergence rate of the cg-method is highly problem-dependent. Using the minimality property (2.18), it can be shown that the convergence rate of the cg-method can be bounded by a function of the condition number of the matrix.

Theorem 2.5.6. *Let A be a symmetric positive definite matrix. For $k = 0, 1, \ldots$, let $e^{(k)} := x^{(k)} - x$ be the difference between the k^{th} iterate in the cg-method and the solution to $Ax = b$. Then:*

$$\frac{\left\|e^{(k)}\right\|_A}{\left\|e^{(0)}\right\|_A} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^k,$$

where $\kappa = \operatorname{cond}_2(A)$.

Therefore, the convergence of the cg-method is exponential and the constant is determined by the square-root of the 2-norm condition number of the matrix.

This behaviour may be exploited by so-called preconditioners, in order to accelerate the cg-method and, at best, try to prove a bound on the total number of iterations for a certain class of problems. To precondition A, we find another symmetric positive definite matrix P, the *preconditioner*, such that $P^{-1}A$ is a good approximation of the identity[9]. However, as we need to be able to apply the inverse, a balance must again be struck.

Algorithm 2.5.7 (Preconditioned Conjugate-Gradient Method)**.** *Initialize $x^{(0)} \in \mathbb{R}^n$ and set $r^{(0)} := b - Ax^{(0)}$, $q^{(0)} := P^{-1}r^{(0)}$ and set $p^{(0)} := q^{(0)}$.*

For $k = 0, 1, \ldots$:

1. *If $\sqrt{\langle r^{(k)}, q^{(k)}\rangle} < \epsilon$, set $m := k$ and stop: $x^{(k)}$ is the approximate solution of $Ax = b$. Otherwise,*
2. *Compute:*

$$\begin{aligned}
\alpha_k &:= \frac{\langle r^{(k)}, q^{(k)}\rangle}{\langle p^{(k)}, Ap^{(k)}\rangle},\\
x^{(k+1)} &:= x^{(k)} + \alpha_k p^{(k)},\\
r^{(k+1)} &:= r^{(k)} - \alpha_k Ap^{(k)},\\
q^{(k+1)} &:= P^{-1}r^{(k+1)},\\
\beta_k &:= \frac{\langle r^{(k+1)}, q^{(k+1)}\rangle}{\langle r^{(k)}, q^{(k)}\rangle},\\
p^{(k+1)} &:= q^{(k+1)} + \beta_k p^{(k)}.
\end{aligned}$$

Essentially, the only difference between the cg- and the preconditioned cg- (pcg-) methods is that we have to solve at each step an additional linear system $Pq = r$ with the preconditioner.

2.6 Problems

1. Prove Lemmas 2.3.2 and 2.3.3 regarding the multiplication matrices $M_n(v)$.
2. The Cholesky factorization of a real symmetric positive definite matrix is the unique factorization $A = LL^\top$, where L is a lower triangular matrix. Derive the relations for the entries of the matrix L. Since the algorithm fails if A is not SPD, the Cholesky factorization is an algorithm to *test* for positive-definiteness. Use your algorithm to check whether the following matrices are SPD:

$$A_1 = \begin{bmatrix} 2 & -1 & \frac{1}{2} & 0\\ -1 & 3 & -1 & -1\\ \frac{1}{2} & -1 & 4 & -1\\ 0 & -1 & -1 & 5 \end{bmatrix}, \quad \text{and} \quad A_2 = \begin{bmatrix} 2 & -1 & -\frac{3}{4} & 0\\ -1 & 2 & -1 & -\frac{3}{4}\\ -\frac{3}{4} & -1 & 2 & -1\\ 0 & -\frac{3}{4} & -1 & 2 \end{bmatrix}.$$

3. (a) Show that Givens matrices $G(i, j, \theta)$ have the eigenvalues $e^{\pm \mathrm{i}\theta}$ and $n - 2$ eigenvalues $+1$;

[9] The name "preconditioner" is derived from the fact that the identity matrix has unit condition number and $P^{-1}A \approx I$.

(b) Show that Householder matrices have one eigenvalue -1 and $n-1$ eigenvalues $+1$. *Hint: clearly $H(w)w = -w$; then, one must identify $n-1$ vectors for which $H(w)v = v$;*

(c) Use Lemma 2.2.8 to show that the eigenvalues of unitary matrices are all on the unit circle;

(d) Conclude from part (c) that if Q is a unitary matrix, $|\det(Q)| = 1$; and,

(e) Conclude from the singular value decomposition of $A \in \mathbb{C}^{n\times n}$ that $|\det(A)| = \prod_{i=1}^{n} \sigma_i$.

4. Formally, let $f(x) = a_0 + a_1 x + a_2 x^2 + \cdots +$ be a power series in x. Show that if $A \in \mathbb{C}^{n\times n}$ is diagonalizable, i.e. $A = V\Lambda V^{-1}$, then $f(A) = Vf(\Lambda)V^{-1}$ where $f(\Lambda) = \operatorname{diag}(f(\lambda_1), \ldots, f(\lambda_n))$.

5. Let $A \in \mathbb{C}^{n\times n}$ and consider its QR factorization. After showing that $\operatorname{cond}_2(A) = \operatorname{cond}_2(Q)\operatorname{cond}_2(R) = \operatorname{cond}_2(R)$, we concluded that the QR factorization is a well-conditioned method for solving linear systems, when viewed as an algorithm. Show that:

   (a) the eigendecomposition of an Hermitian matrix $A \in \mathbb{C}^{n\times n}$; and,

   (b) the singular value decomposition of $A \in \mathbb{C}^{n\times n}$;

   are also well-conditioned methods for solving linear systems.

6. Prove Theorem 2.3.17. *Hint: combine Example 1.1.18 and Lemma 2.2.8.*

7. Prove Theorem 2.5.3. *Hint: Use as an initial guess the normalized eigenvector with corresponding eigenvalue equal in absolute value to the spectral radius. Show that strict diagonal dominance bounds the spectral radius by* 1.

8. Let $n = 10,000$, let:

$$T = \begin{bmatrix} 3 & -1 & & & \\ -1 & 3 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 3 & -1 \\ & & & -1 & 3 \end{bmatrix} \in \mathbb{R}^{n\times n},$$

   be the symmetric tridiagonal matrix of threes on the diagonal and ones on the sub- and super-diagonals, and let $u = (1/1, \ldots, 1/n)^\top$ and $b = (0, \ldots, 0, 1)^\top$. The matrix A is formed by adding the symmetric tridiagonal matrix T and the symmetric outer product $uu^\top$:

$$A = T + uu^\top \in \mathbb{R}^{n\times n}.$$

   The matrix A is the data-sparse result of discretizing an integro-differential equation. It can be taken for granted that for every $n \in \mathbb{N}$, A is also positive-definite.

   Solve the system $Ax = b$ using Jacobi and Gauss–Seidel iterations and the pcg-method with a preconditioner of your choice. Compare and contrast the methods and your solutions. How many steps does each method take for residual error $\left\|Ax^{(k)} - b\right\|_2 < 10^{-10}$?

Chapter 3

Interpolation and Approximation

Given data points $(x_0, f_0), \ldots, (x_n, f_n)$ with distinct abscissæ ($x_i \neq x_j \; \forall i \neq j$), we wish to find a function which $f(x)$ which interpolates the data points: $f(x_i) = f_i$ for $i = 0, \ldots, n$. The simplest functions that satisfy our ability to interpolate the data are degree-n polynomials $p_n(x)$ which interpolate the data points: $p_n(x_i) = f_i$ for $i = 0, \ldots, n$.

Let $\mathbb{P}_n$ denote the space of all polynomials of degree at most n.

3.1 Lagrange Interpolation

3.1.1 Existence and Uniqueness

Theorem 3.1.1. *There exists a polynomial $p_n \in \mathbb{P}_n$ such that $p_n(x_i) = f_i$ for $i = 0, \ldots, n$.*

Proof. Consider, for $k = 0, \ldots, n$, the Lagrange basis polynomials:

$$\ell_{n,k}(x) = \frac{(x - x_0) \cdots (x - x_{k-1})(x - x_{k+1}) \cdots (x - x_n)}{(x_k - x_0) \cdots (x_k - x_{k-1})(x_k - x_{k+1}) \cdots (x_k - x_n)} \in \mathbb{P}_n. \tag{3.1}$$

Then:

$$\ell_{n,k}(x_i) = \begin{cases} 0 & \text{for} \quad i = 0, \ldots, k-1, k+1, \ldots, n, \\ 1 & \text{for} \quad i = k. \end{cases} \tag{3.2}$$

So now define:

$$p_n(x) = \sum_{k=0}^{n} f_k \ell_{n,k}(x) \in \mathbb{P}_n. \tag{3.3}$$

From (3.2), we know that $p_n(x_i) = f_i$ for $i = 0, \ldots, n$. □

The polynomial (3.3) is the **Lagrange interpolating polynomial**.

Theorem 3.1.2. *The interpolating polynomial of degree $\leq n$ is unique.*

Proof. Consider two interpolating polynomials $p_n, q_n \in \mathbb{P}_n$. Their difference $d_n = p_n - q_n \in \mathbb{P}_n$ satisfies $d_n(x_k) = 0$ for $k = 0, \ldots, n$. In other words, d_n is a polynomial of degree at most n but has at least $n + 1$ distinct roots. The Fundamental Theorem of Algebra states that $d_n \equiv 0 \Rightarrow p_n = q_n$. □

Suppose that $f \in C^{n+1}(x_0, x_n)$. Let $f_k = f(x_k)$ for $k = 0, \ldots, n$, and let p_n be the Lagrange interpolating polynomial for the data (x_k, f_k) for $k = 0, \ldots, n$. How large can the error $f(x) - p_n(x)$ be on the interval $[x_0, x_n]$? Let:

$$\ell(x) := \prod_{i=0}^{n}(x - x_i) = (x - x_0)(x - x_1)\cdots(x - x_n) \in \mathbb{P}_{n+1}, \tag{3.4}$$
$$= x^{n+1} - \left(\sum_{i=0}^{n} x_i\right) x^n + \cdots + (-1)^{n+1} x_0 \cdots x_n.$$

Theorem 3.1.3. *For every $x \in [x_0, x_n]$, there exists $\xi = \xi(x) \in (x_0, x_n)$ such that:*

$$r(x) := f(x) - p_n(x) = \ell(x)\frac{f^{(n+1)}(\xi)}{(n+1)!},$$

where $f^{(n+1)}$ is the $(n+1)^{\text{th}}$ derivative of f.

Proof. It holds trivially for the interpolation points $x = x_k$, for $k = 0, \ldots, n$ as $r(x_k) = 0$. Supposing $x \neq x_k$, let:

$$\phi(t) := r(t) - \frac{r(x)}{\ell(x)}\ell(t).$$

Note that ϕ, as a function of t, vanishes at $n + 2$ points: the set x_k, for $k = 0, \ldots, n$ and the point x. Therefore, ϕ' will vanish at $n + 1$ points $\xi_0, \ldots, \xi_n$ between these points, ϕ'' will vanish at n points between these new points, and so on and so forth until we conclude that $\phi^{(n+1)}$ will vanish at some (unknown) point $\xi \in (x_0, x_n)$. But:

$$\phi^{(n+1)}(t) = r^{(n+1)}(t) - \frac{r(x)}{\ell(x)}\ell^{(n+1)}(t) = f^{(n+1)}(t) - \frac{r(x)}{\ell(x)}(n+1)!,$$

since $p_n^{(n+1)}(t) \equiv 0$ and since $\ell(t)$ is monic. The result then follows immediately from this identity since $\phi^{(n+1)}(\xi) = 0$. □

3.1.2 Recursive Construction of Lagrange Interpolating Polynomials

Let $L_{i,j}$ be the Lagrange interpolating polynomial at x_k for $k = i, \ldots, j$.

Theorem 3.1.4.

$$L_{i,j}(x) = \frac{(x - x_i)L_{i+1,j}(x) - (x - x_j)L_{i,j-1}(x)}{x_j - x_i}. \tag{3.5}$$

Proof. Let $s(x)$ denote the right-hand side (3.5). Due to the uniqueness of interpolating polynomials, we simply wish to show that $s(x_k) = f_k$. For $k = i + 1, \ldots, j - 1$, $L_{i+1,j}(x_k) = f_k = L_{i,j-1}(x_k)$, and hence:

$$s(x_k) = \frac{(x_k - x_i)L_{i+1,j}(x_k) - (x_k - x_j)L_{i,j-1}(x_k)}{x_j - x_i} = f_k.$$

To complete the proof, since $L_{i+1,j}(x_j) = f_j$ and $L_{i,j-1}(x_i) = f_i$, we have that:

$$s(x_i) = L_{i,j-1}(x_i) = f_i \text{ and } s(x_j) = L_{i+1,j}(x_j) = f_j.$$

□

3.1.3 Barycentric Formulæ

Note that the function $\ell(x)$ from (3.4) can help us rewrite the Lagrange Basis polynomials of (3.1) more compactly:

$$\ell_{n,k}(x) = \frac{\ell(x)}{x - x_k} \lim_{z \to x_k} \frac{z - x_k}{\ell(z)} = \frac{\ell(x)}{x - x_k} \frac{1}{\ell'(x_k)}. \tag{3.6}$$

Let $\lambda_k := 1/\ell'(x_k)$. Then, the Lagrange interpolating polynomial (3.3) can be recast as:

$$p_n(x) = \ell(x) \sum_{k=0}^{n} \frac{\lambda_k f_k}{x - x_k}, \tag{3.7}$$

which is known as the **first barycentric formula**. If we divide through by a "clever" form of 1:

$$1 = \ell(x) \sum_{k=0}^{n} \frac{\lambda_k \cdot 1}{x - x_k}, \tag{3.8}$$

then we arrive at the **second barycentric formula**:

$$p_n(x) = \frac{\displaystyle\sum_{k=0}^{n} \frac{\lambda_k f_k}{x - x_k}}{\displaystyle\sum_{k=0}^{n} \frac{\lambda_k}{x - x_k}}. \tag{3.9}$$

In the barycentric formulæ, what might you expect to go wrong when implementing these formulæ on a computer? Despite this, it turns out they are completely well-behaved (numerically stable): see [9] for the relatively recent proof.

The benefit of the barycentric formulæ is that once the weights λ_k for $k = 0, \ldots, n$ are computed (in $\mathcal{O}(n^2)$ operations), numerical evaluation of the interpolant p_n can be performed in $\mathcal{O}(n)$ operations. This is a significant improvement over the formula (3.3).

3.1.4 Hermite Interpolating Polynomial

The interpolation process can be generalized to also capture derivative information at the interpolating points. Given data $(x_0, f_0, f_0'), \ldots, (x_n, f_n, f_n')$ with x_i distinct, can we find a polynomial p such that $p(x_i) = f_i$ and $p'(x_i) = f_i'$?

Theorem 3.1.5. *There is a unique polynomial $p_{2n+1} \in \mathbb{P}_{2n+1}$ such that $p_{2n+1}(x_i) = f_i$ and $p'_{2n+1}(x_i) = f_i'$ for $i = 0, \ldots, n$.*

Proof. Given the Lagrange basis polynomials $\ell_{n,k}(x)$ from (3.1), let:

$$\begin{aligned} s_{n,k}(x) &= \ell_{n,k}(x)^2 (1 - 2(x - x_k)\ell'_{n,k}(x_k)) \in \mathbb{P}_{2n+1}, \\ \text{and } t_{n,k}(x) &= \ell_{n,k}(x)^2 (x - x_k) \in \mathbb{P}_{2n+1}. \end{aligned}$$

Then:

$$p_{2n+1}(x) = \sum_{k=0}^{n} \left[f_k s_{n,k}(x) + f_k' t_{n,k}(x) \right], \tag{3.10}$$

interpolates the data. □

Theorem 3.1.6. *Let $f \in C^{2n+2}(x_0, x_n)$ and let p_{2n+1} be the Hermite interpolating polynomial of f. For every $x \in [x_0, x_n]$, there exists $\xi = \xi(x) \in (x_0, x_n)$ such that:*

$$r(x) := f(x) - p_{2n+1}(x) = [\ell(x)]^2 \frac{f^{(2n+2)}(\xi)}{(2n+2)!},$$

where $f^{(2n+2)}$ is the $(2n+2)^{\text{th}}$ derivative of f.

3.1.5 Newton's Divided Differences

An alternative to the reformulation of Lagrange interpolation in barycentric coordinates is the use of Newton's divided differences. Consider the sequence of divided differences:

$$\begin{aligned}
f[x_i] &:= f_i, \quad \text{for} \quad i = 0, \ldots, n; \\
f[x_i, x_{i+1}] &:= \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}, \quad \text{for} \quad i = 0, \ldots, n-1; \\
f[x_i, x_{i+1}, x_{i+2}] &:= \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \quad \text{for} \quad i = 0, \ldots, n-2; \\
&\vdots \\
f[x_i, \ldots, x_j] &:= \frac{f[x_{i+1}, \ldots, x_j] - f[x_i, \ldots, x_{j-1}]}{x_j - x_i}, \quad \text{for} \quad i = 0, \ldots, n-j.
\end{aligned}$$

Using these divided differences, we can iteratively construct an polynomial interpolant through $x_0, \ldots, x_n$.

Theorem 3.1.7. *The* **Newton interpolating polynomial***:*

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + \cdots + f[x_0, \ldots, x_n](x - x_0) \cdots (x - x_{n-1}), \tag{3.11}$$

interpolates the data (x_k, f_k) for $k = 0, \ldots, n$.

Proof. Clearly $p_0(x) = f[x_0] = f_0$. Assume the formula (3.11) with $n-1$ interpolates the data (x_k, f_k) at n points. Let $q_{n-1}(x)$ be the unique interpolating polynomial through $x_0, \ldots, x_{n-1}$ and let $r_{n-1}(x)$ be the unique interpolating polynomial through $x_1, \ldots, x_n$. Then, we claim:

$$p_n(x) = q_{n-1}(x) + \frac{x - x_0}{x_n - x_0}(r_{n-1}(x) - q_{n-1}(x)). \tag{3.12}$$

Clearly, $p_n(x_0) = q_{n-1}(x_0) = f_0$ and $p_n(x_n) = q_{n-1}(x_n) + r_{n-1}(x_n) - q_{n-1}(x_n) = r_{n-1}(x_n) = f_n$. For $i = 1, \ldots, n-1$:

$$p_n(x_i) = q(x_i) + \frac{x_i - x_0}{x_n - x_0}(r_{n-1}(x_i) - q_{n-1}(x_i)) = q_{n-1}(x_i) = f_i$$

Therefore, $p_n(x_i)$ is the unique degree-n interpolant through $x_0, \ldots, x_n$. Now, the coefficient of x^n in $p_n(x)$ must be the same as that of the expression on the right of (3.12). That is:

$$f[x_0, \ldots, x_n] = \frac{f[x_1, \ldots, x_n] - f[x_0, \ldots, x_{n-1}]}{x_n - x_0},$$

which is true by definition of Newton's divided differences. □

Newton's divided differences can be computed recursively by organizing the calculation in a tableau:

$$\begin{array}{lll} x_0 & f[x_0] & \\ & & f[x_0,x_1] \\ x_1 & f[x_1] & & f[x_0,x_1,x_2] \\ & & f[x_1,x_2] & & \ddots \\ x_2 & f[x_2] & & \ddots \\ \vdots & \vdots & \end{array}$$

Once the divided differences are computed, numerical evaluation of polynomials in Newton form can be performed in $\mathcal{O}(n)$ flops. In addition, addition of a new data point can be performed in $\mathcal{O}(n)$ operations, provided the original tableau is saved.

Example 3.1.8. *Given the data* $(0,1)$, $(2,2)$, $(3,4)$*, construct the Newton's divided difference table and write down the interpolating polynomial of degree-*2*.*

$$\begin{array}{lll} 0 & 1 & \\ & & \frac{1}{2} \\ 2 & 2 & & \frac{1}{2} \\ & & 2 \\ 3 & 4 & \end{array}$$

The Newton interpolating polynomial is then:

$$p_2(x) = 1 + \tfrac{1}{2}x + \tfrac{1}{2}x(x-2).$$

Remark 3.1.9.

1. Sometimes, it is useful to use the divided difference operator notation $\delta^{(n)}[x_0,\ldots,x_n]$ maps continuous functions to their n^{th} Newton divided differences through the points $x_0,\ldots,x_n$.
2. If we take the limit as $(x_0,\ldots,x_n) \to (a,\ldots,a)$, then the Newton interpolating polynomial becomes the Taylor expansion of f at a, since divided differences become derivatives in the limit.

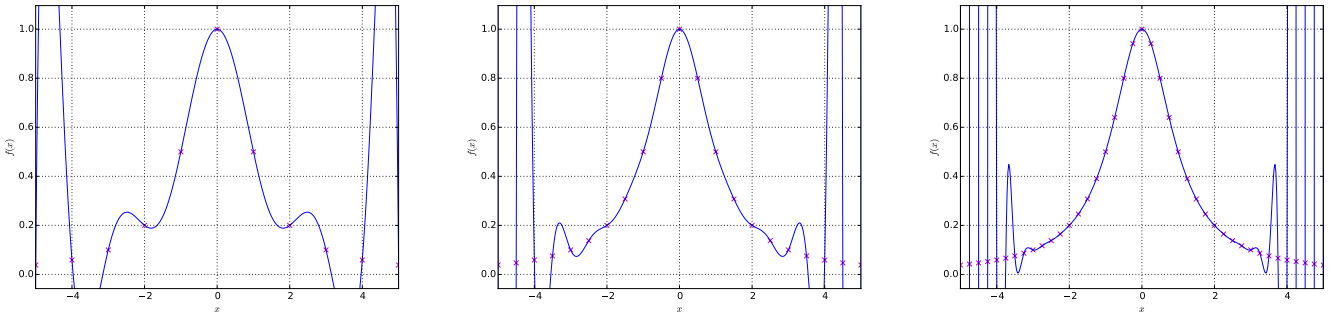
3.1.6 Runge phenomenon

The Weierstrass approximation theorem states that for every $f \in C([a,b])$ there exists a degree-graded set of polynomials $p_n(x) \in \mathbb{P}_n$ for $n \in \mathbb{N}_0$ that approximates f uniformly over $[a,b]$ in the limit as $n \to \infty$, that is:

$$\lim_{n\to\infty} \sup_{x\in[a,b]} |f(x) - p_n(x)| = \lim_{n\to\infty} \|f - p_n\|_\infty = 0. \tag{3.13}$$

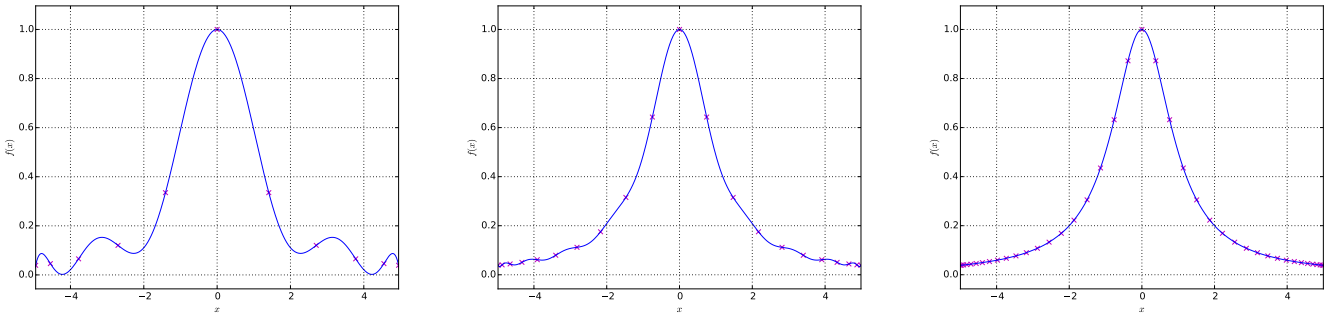
One might expect that many sets of polynomials, and therefore many sets of interpolation points, will converge uniformly to a continuous function. However, it is not so difficult to find a counterexample.

In 1901, Carl Runge showed [10] that interpolation of an analytic function in equispaced points may not converge uniformly to the function f on the interval of interpolation. The function he used to prove such a result is the function $f(x) = (1+x^2)^{-1}$ on $[-5,5]$ using equispaced points. Figure 3.1 shows the resulting interpolating polynomials, and Figure 3.2 shows a preview of the benefits of using points clustered near the ends of the interval.



(a) (b) (c)

Figure 3.1: The Runge phenomenon on interpolating $f(x) = (1 + x^2)^{-1}$ on $[-5, 5]$: (a) using 11 equispaced points (b) using 21 equispaced points (c) using 41 equispaced points.



(a) (b) (c)

Figure 3.2: Uniform convergence on interpolating $f(x) = (1 + x^2)^{-1}$ on $[-5, 5]$: (a) using 11 Chebyshev points (b) using 21 Chebyshev points (c) using 41 Chebyshev points.

3.2 Best Polynomial Approximation

Given a function $f : D \to \mathbb{C}$, we wish to find its best polynomial approximation. To measure our progress, we can use $L^p(D,\, \mathrm{d}\mu(x))$ spaces and their associated norms: find $p_n \in \mathbb{P}_n$ that is the best polynomial approximation to $f \in L^p(D,\, \mathrm{d}\mu(x))$ such that:

$$\|f - p_n\|_p \leq \|f - q\|_p, \qquad \forall q \in \mathbb{P}_n.$$

Posed in this generality, the problem is still the subject of ongoing research. Therefore, to make any reasonable progress, we will consider the solved problems of best approximation in L^2 followed by best approximation in L^∞.

3.2.1 Best Polynomial Approximation in L^2 and Orthogonal Polynomials

We now consider the best polynomial approximations in $L^2(D,\, \mathrm{d}\mu(x))$, i.e. find $p_n \in \mathbb{P}_n$ such that:

$$\|f - p_n\|_2 \leq \|f - q\|_2, \qquad \forall q \in \mathbb{P}_n.$$

Supposing p_n has the form $p_n(x) = \sum_{i=0}^n \alpha_i x^i$ then results in the minimization problem:

$$\min_{(\alpha_0,\ldots,\alpha_n)\in\mathbb{C}^n} \int_D \left| f(x) - \sum_{i=0}^n \alpha_i x^i \right|^2 \mathrm{d}\mu(x).$$

The unique minimizer can be found from the (linear) system:

$$\frac{\partial}{\partial \alpha_k} \int_D \left| f(x) - \sum_{i=0}^n \alpha_i x^i \right|^2 \mathrm{d}\mu(x), \quad \text{for each} \quad k = 0, \ldots, n.$$

But there is some additional structure that we will exploit rather than solving this system of equations.

Theorem 3.2.1. *If $f \in L^2(D,\, \mathrm{d}\mu(x))$ and $p_n \in \mathbb{P}_n$ is such that:*

$$\langle r, f - p_n \rangle = 0, \quad \forall r \in \mathbb{P}_n, \tag{3.14}$$

then:

$$\|f - p_n\|_2 \leq \|f - r\|_2, \quad \forall r \in \mathbb{P}_n, \tag{3.15}$$

i.e. p_n is a best (weighted) least-squares approximation to f on D.

Proof.

$$\begin{aligned} \|f - p_n\|_2^2 &= |\langle f - p_n, f - p_n \rangle| \\ &= |\langle f - r, f - p_n \rangle + \langle r - p_n, f - p_n \rangle|, \quad \forall r \in \mathbb{P}_n. \end{aligned}$$

Since $r - p_n \in \mathbb{P}_n$, the assumption (3.14) implies that:

$$\begin{aligned} \|f - p_n\|_2^2 &= |\langle f - r, f - p_n \rangle| \\ &\leq \|f - r\|_2 \|f - p_n\|_2, \end{aligned}$$

by the Cauchy–Schwarz inequality. □

Remark 3.2.2. The converse is also true.

This gives a direct, albeit poor, way to calculate best polynomial approximations in L^2: we wish to find $p_n(x) = \sum_{i=0}^n \alpha_i x^i$ such that:

$$\int_D \overline{x^k} \left[f(x) - \sum_{i=0}^n \alpha_i x^i \right] \mathrm{d}\mu(x) = 0, \quad \text{for} \quad k = 0, \ldots, n. \tag{3.16}$$

Rearranging these equations, we have:

$$\sum_{i=0}^n \left(\int_D \overline{x^k} x^i \, \mathrm{d}\mu(x) \right) \alpha_i = \int_D \overline{x^k} f(x) \, \mathrm{d}\mu(x), \quad \text{for} \quad k = 0, \ldots, n, \tag{3.17}$$

which is the component-wise statement of the matrix equation:

$$A\alpha = \varphi, \tag{3.18}$$

to determine the coefficients $\alpha = (\alpha_0, \ldots, \alpha_n)^\top$ from the data $\varphi = (\varphi_0, \ldots, \varphi_n)^\top$, where:

$$A_{k,i} = \int_D \overline{x^k} x^i \, \mathrm{d}\mu(x), \quad \text{and} \quad \varphi_k = \int_D \overline{x^k} f(x) \, \mathrm{d}\mu(x). \tag{3.19}$$

The system (3.18) are called the **normal equations**.

Example 3.2.3. *We wish to find the best least-squares approximation to e^x on $[0, 1]$ from $\mathbb{P}_1$. We must solve:*

$$\int_0^1 1[e^x - (\alpha_0 + \alpha_1 x)] \, \mathrm{d}x = 0, \quad \text{and} \quad \int_0^1 x[e^x - (\alpha_0 + \alpha_1 x)] \, \mathrm{d}x = 0.$$

This is equivalent to:

$$\begin{aligned} \alpha_0 \int_0^1 \mathrm{d}x + \alpha_1 \int_0^1 x \, \mathrm{d}x &= \int_0^1 e^x \, \mathrm{d}x, \\ \alpha_0 \int_0^1 x \, \mathrm{d}x + \alpha_1 \int_0^1 x^2 \, \mathrm{d}x &= \int_0^1 e^x x \, \mathrm{d}x, \end{aligned}$$

or:

$$\begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} \begin{bmatrix} e - 1 \\ 1 \end{bmatrix},$$

which has the solution $\alpha_0 = 4e - 10$ and $\alpha_1 = 18 - 6e$. Therefore, $p_1(x) = (18 - 6e)x + (4e - 10)$ is the best approximation to e^x in $L^2([0, 1], \, \mathrm{d}x)$.

Theorem 3.2.4. *The matrix A in* (3.18) *is nonsingular.*

Proof. Suppose A *is* singular, i.e. $\exists \alpha \neq 0$ such that $A\alpha = 0$ implying $\alpha^* A \alpha = 0$. This is:

$$\sum_{k=0}^n \overline{\alpha_k} (A\alpha)_k \iff \sum_{k=0}^n \overline{\alpha_k} \sum_{i=0}^n A_{k,i} \alpha_i = 0. \tag{3.20}$$

Since, by definition $A_{k,i} = \int_D \overline{x^k} x^i \, \mathrm{d}\mu(x)$, we find:

$$\iff \sum_{k=0}^{n} \overline{\alpha_k} \sum_{i=0}^{n} \int_D \overline{x^k} x^i \, \mathrm{d}\mu(x) \alpha_i = 0. \tag{3.21}$$

Rearranging gives:

$$\int_D \overline{\left(\sum_{k=0}^{n} \alpha_k x^k \right)} \left(\sum_{i=0}^{n} \alpha_i x^i \right) \, \mathrm{d}\mu(x) = 0, \quad \text{or} \quad \left\| \sum_{i=0}^{n} \alpha_i x^i \right\|_2^2 = 0. \tag{3.22}$$

But this last equality implies that $\sum_{i=0}^{n} \alpha_i x^i \equiv 0$ and thus $\alpha_i \equiv 0$ for $i = 0, \ldots, n$. We have arrived at a contradiction. □

Invertibility of the matrix A establishes existence and uniqueness of the best polynomial approximation in $L^2(D, \mathrm{d}\mu(x))$. It does not, however, imply that the normal equations are well-conditioned. To find a well-conditioned solution to this problem, we use our knowledge of the QR factorization for matrices to construct degree-graded polynomial bases that behave significantly better than the monomials.

The solution of the normal equations $A\alpha = \varphi$ for best least-squares polynomial approximation would be trivial if A were diagonal. Instead of using $\{1, x, \ldots, x^n\}$ as a basis for $\mathbb{P}_n$, we shall construct a basis $\{\pi_0, \pi_1, \ldots, \pi_n\}$ in order to diagonalize A.

In this basis, our best polynomial approximation $p_n(x) = \sum_{i=0}^{n} \beta_i \pi_i(x)$, and the normal equations become:

$$\int_D \overline{\pi_k(x)} \left[f(x) - \sum_{i=0}^{n} \beta_i \pi_i(x) \right] \, \mathrm{d}\mu(x) = 0, \quad \text{for} \quad k = 0, \ldots, n, \tag{3.23}$$

or equivalently:

$$\sum_{i=0}^{n} \left(\int_D \overline{\pi_k(x)} \pi_i(x) \, \mathrm{d}\mu(x) \right) \beta_i = \int_D \overline{\pi_k(x)} f(x) \, \mathrm{d}\mu(x), \quad \text{for} \quad k = 0, \ldots, n,$$

or:

$$A\beta = \varphi, \tag{3.24}$$

where $\beta = (\beta_0, \ldots, \beta_n)^\top$, $\varphi = (\varphi_0, \ldots, \varphi_n)^\top$ but now:

$$A_{k,i} = \langle \pi_k, \pi_i \rangle = \int_D \overline{\pi_k(x)} \pi_i(x) \, \mathrm{d}\mu(x), \quad \text{and} \quad \varphi_k = \int_D \overline{\pi_k(x)} f(x) \, \mathrm{d}\mu(x). \tag{3.25}$$

The matrix A is diagonalized iff:

$$\langle \pi_k, \pi_i \rangle \begin{cases} = 0 & \text{for} \quad i \neq k, \\ \neq 0 & \text{for} \quad i = k. \end{cases}$$

We can use this (countably infinite) set of conditions to construct the **orthogonal polynomials** with respect to the measure $\mathrm{d}\mu(x)$. This is known as the Gram–Schmidt procedure.

Lemma 3.2.5. *Suppose that $\pi_0, \ldots, \pi_k$, with $\pi_i \in \mathbb{P}_i$ for each i, are orthogonal with respect to the inner product $\langle f, g \rangle = \int_D \overline{f(x)} g(x) \, \mathrm{d}\mu(x)$. Then:*

$$\pi_{k+1}(x) = x^{k+1} - \sum_{i=0}^{k} \delta_i \pi_i(x),$$

satisfies:

$$\langle \pi_j, \pi_{k+1} \rangle = 0, \quad \text{for} \quad j = 0, \ldots, k, \qquad \delta_j = \frac{\langle \pi_j, x^{k+1} \rangle}{\langle \pi_j, \pi_j \rangle}.$$

Proof. For any $j = 0, \ldots, k$,

$$\begin{aligned} \langle \pi_j, \pi_{k+1} \rangle &= \langle \pi_j, x^{k+1} \rangle - \sum_{i=0}^{k} \delta_i \langle \pi_j, \pi_i \rangle \\ &= \langle \pi_j, x^{k+1} \rangle - \delta_j \langle \pi_j, \pi_j \rangle \quad \text{by orthogonality of } \pi_i \text{ and } \pi_j \text{ for } i \neq j, \\ &= 0, \quad \text{by definition of } \delta_j. \end{aligned}$$

□

Remark 3.2.6. The Gram–Schmidt procedure constructs orthogonal polynomials inductively starting from $\pi_0(x) = 1$. $\pi_k(x)$ is always of exact degree k, so $\{\pi_0, \ldots, \pi_k\}$ is a basis for $\mathbb{P}_k$, $\forall k \in \mathbb{N}_0$. The orthogonal polynomials can be normalized to be **orthonormal**, i.e. $\langle \pi_k, \pi_k \rangle = 1$, to be **monic**, i.e. $\hat{\pi}_k(x) = x^k + \cdots +$, or even have other normalizations such as $\pi_k(1) = 1$.

Properties of Orthogonal Polynomials

In order to provide a basis for computational techniques associated with the preceding approximations, as well as for later developments, we next exhibit some useful properties of orthogonal polynomials.

Theorem 3.2.7. *For every $n \in \mathbb{N}$ and $a, b \in \mathbb{R}$, if the measure μ is monotonic, the orthogonal polynomials π_n with respect to $L^2([a, b], \, \mathrm{d}\mu(x))$ possess n distinct real zeros contained in (a, b).*

Proof. If π_n does not change in sign in $[a, b]$, then:

$$\int_a^b \pi_n \, \mathrm{d}\mu(x) = \langle \pi_0, \pi_n \rangle,$$

is either positive or negative. In either case, this is a contradiction since the right-hand side must be zero by orthogonality. Hence, π_n must have a zero in (a, b). Now, let those real zeros of $\pi_n(x)$ which are of *odd* multiplicity, and which lie in (a, b), be denoted by $x_1, x_2, \ldots, x_m$, and *assume* that $m < n$. Then the product:

$$(x - x_1)(x - x_2) \cdots (x - x_m)\pi_n(x),$$

does not change sign in $[a, b]$. Since $m < n$, the product $(x - x_1) \cdots (x - x_m)$ is a polynomial of degree less than n and it may be expressed in terms of the basis $\{\pi_k(x)\}_{k=0}^{m}$. We must have:

$$\langle (x - x_1)(x - x_2) \cdots (x - x_m), \pi_n(x) \rangle = 0. \tag{3.26}$$

However, since the measure is monotonic in $[a, b]$, the integrand therefore has the same property, and we have reached a contradiction, as this guarantees that (3.26) must be different from 0. Hence, it follows that $m = n$, and since the total multiplicity of *all* zeros is equal to n, all roots must be real and distinct and must lie in (a, b). □

Theorem 3.2.8. *For every $n \in \mathbb{N}$, the monic orthogonal polynomials $\hat{\pi}_n$ with respect to $L^2([a,b],\, \mathrm{d}\mu(x))$ satisfy a three-term recurrence relation of the form:*

$$\hat{\pi}_{n+1}(x) = (x - \alpha_n)\hat{\pi}_n(x) - \beta_n \hat{\pi}_{n-1}(x). \tag{3.27}$$

Proof. The case $n = 1$ holds trivially. Assume the result holds for $n-1$. Since $\hat{\pi}_{n+1} - x\hat{\pi}_n \in \mathbb{P}_n$ and since $\{\hat{\pi}_0, \ldots, \hat{\pi}_n\}$ is a basis for $\mathbb{P}_n$, it follows that:

$$\hat{\pi}_{n+1}(x) - x\hat{\pi}_n(x) = -\alpha_n \hat{\pi}_n(x) - \beta_n \hat{\pi}_{n-1} - \sum_{j=0}^{n-2} \gamma_j \hat{\pi}_j(x),$$

for some constants α_n, β_n, and γ_j. If we take the inner product with $\hat{\pi}_n$, we obtain:

$$0 - \langle x\hat{\pi}_n, \hat{\pi}_n \rangle = -\alpha_n \langle \hat{\pi}_n, \hat{\pi}_n \rangle \quad \text{or} \quad \alpha_n = \frac{\langle x\hat{\pi}_n, \hat{\pi}_n \rangle}{\langle \hat{\pi}_n, \hat{\pi}_n \rangle}.$$

Next, if we take the inner product with $\hat{\pi}_{n-1}$, we obtain:

$$0 - \langle x\hat{\pi}_n, \hat{\pi}_{n-1} \rangle = -\beta_n \langle \hat{\pi}_{n-1}, \hat{\pi}_{n-1} \rangle \quad \text{or} \quad \beta_n = \frac{\langle x\hat{\pi}_n, \hat{\pi}_{n-1} \rangle}{\langle \hat{\pi}_{n-1}, \hat{\pi}_{n-1} \rangle} = \frac{\langle \hat{\pi}_n, x\hat{\pi}_{n-1} \rangle}{\langle \hat{\pi}_{n-1}, \hat{\pi}_{n-1} \rangle} = \frac{\langle \hat{\pi}_n, \hat{\pi}_n \rangle}{\langle \hat{\pi}_{n-1}, \hat{\pi}_{n-1} \rangle}.$$

Finally, any of the inner products with $\hat{\pi}_j$ for $j = 0, \ldots, n-2$ give:

$$0 - \langle x\hat{\pi}_n, \hat{\pi}_j \rangle = -\gamma_j \langle \hat{\pi}_j, \hat{\pi}_j \rangle.$$

Since $\deg(x\hat{\pi}_j) = j + 1 < n$, each of the $\gamma_j = 0$. □

Without loss of generality, orthogonal polynomials π_n with any normalization (not necessarily monic) satisfy a three-term recurrence relation. Consider then the notation:

$$\pi_{n+1}(x) = (A_n x + B_n)\pi_n(x) - C_n \pi_{n-1}(x), \qquad \pi_{-1}(x) = 0, \quad \pi_0(x) = 1. \tag{3.28}$$

The following algorithm is the extension of Horner's rule for numerical evaluation of polynomials expressed in general orthogonal polynomial bases. The Clenshaw–Smith algorithm writes the sum:

$$p_n(x) = \sum_{k=0}^{n} c_k \pi_k(x), \tag{3.29}$$

via an inhomogeneous recurrence relation involving the adjoint of (3.28) as follows:

Algorithm 3.2.9 (Clenshaw–Smith [11, 12])**.**

1. *Set:*
$$u_{n+1}(x) = u_{n+2}(x) = 0. \tag{3.30}$$
2. *For $k = n, n-1, \ldots, 0$:*
$$u_k(x) = (A_k x + B_k)u_{k+1}(x) - C_{k+1}u_{k+2}(x) + c_k. \tag{3.31}$$
3. *Then:*
$$p_n(x) = u_0(x). \tag{3.32}$$

The Clenshaw–Smith allows for numerical evaluation of polynomials expressed in orthogonal bases in $\mathcal{O}(n)$ flops with $\mathcal{O}(1)$ storage.

Theorem 3.2.10. *Let $\tilde{\pi}_n$ represent the orthonormal polynomials with respect to $L^2([a,b],\,\mathrm{d}\mu(x))$. Then, the Christoffel–Darboux formula is:*

$$\sum_{k=0}^{n-1}\tilde{\pi}_k(x)\tilde{\pi}_k(y)=\sqrt{\beta_n}\frac{\tilde{\pi}_n(x)\tilde{\pi}_{n-1}(y)-\tilde{\pi}_{n-1}(x)\tilde{\pi}_n(y)}{x-y}. \tag{3.33}$$

Proof. Multiplying the three-term recurrence relation (3.27) by $\hat{\pi}_k(y)$ and subtracting the resulting relation from the one with x and y interchanged yields:

$$(x-y)\hat{\pi}_k(x)\hat{\pi}_k(y)=\beta_k\hat{\pi}_{k-1}(x)\hat{\pi}_k(y)-\beta_k\hat{\pi}_k(x)\hat{\pi}_{k-1}(y)+\hat{\pi}_{k+1}(x)\hat{\pi}_k(y)-\hat{\pi}_k(x)\hat{\pi}_{k+1}(y).$$

Recall that $\beta_k=\frac{\langle\hat{\pi}_k,\hat{\pi}_k\rangle}{\langle\hat{\pi}_{k-1},\hat{\pi}_{k-1}\rangle}$. So this relation is equivalent to:

$$\begin{aligned}(x-y)\frac{\hat{\pi}_k(x)\hat{\pi}_k(y)}{\langle\hat{\pi}_k,\hat{\pi}_k\rangle}&=\frac{\hat{\pi}_{k-1}(x)\hat{\pi}_k(y)}{\langle\hat{\pi}_{k-1},\hat{\pi}_{k-1}\rangle}-\frac{\hat{\pi}_k(x)\hat{\pi}_{k-1}(y)}{\langle\hat{\pi}_{k-1},\hat{\pi}_{k-1}\rangle}\\&\quad-\frac{\hat{\pi}_k(x)\hat{\pi}_{k+1}(y)}{\langle\hat{\pi}_k,\hat{\pi}_k\rangle}+\frac{\hat{\pi}_{k+1}(x)\hat{\pi}_k(y)}{\langle\hat{\pi}_k,\hat{\pi}_k\rangle}.\end{aligned}$$

Summing both sides from $k=0$ to $k=n-1$ and observing $\hat{\pi}_{-1}\equiv 0$ and the telescoping nature of the summation on the right gives:

$$\sum_{k=0}^{n-1}\frac{\hat{\pi}_k(x)\hat{\pi}_k(y)}{\langle\hat{\pi}_k,\hat{\pi}_k\rangle}=\frac{1}{\langle\hat{\pi}_{n-1},\hat{\pi}_{n-1}\rangle}\frac{\hat{\pi}_n(x)\hat{\pi}_{n-1}(y)-\hat{\pi}_{n-1}(x)\hat{\pi}_n(y)}{x-y}.$$

Equation (3.33) only requires orthonormalization. □

The Classical Orthogonal Polynomials

The classical orthogonal polynomials are all eigenfunctions of linear second-order homogeneous differential equations of the form:

$$Q(x)\pi_n''(x)+L(x)\pi_n'(x)=\lambda_n\pi_n(x), \tag{3.34}$$

where $Q\in\mathbb{P}_2$ and $L\in\mathbb{P}_1$, and where the eigenvalues are given by $\lambda_n=nL'(x)+\frac{n(n-1)}{2}Q''(x)$. With this extra structure, an explicit representation in terms of higher order derivatives can be derived.

Theorem 3.2.11 (p. 24 of Nikiforov and Uvarov [13])**.** *The classical orthogonal polynomials $\pi_n\in\mathbb{P}_n$ with respect to the inner product space $L^2(D,w(x)\,\mathrm{d}x)$ can be expressed by* **Rodrigues' formula***:*

$$\pi_n(x)=\frac{1}{\kappa_n w(x)}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left(w(x)(Q(x))^n\right). \tag{3.35}$$

Example 3.2.12. *The Legendre polynomials, denoted by $P_n(x)$, are orthogonal with respect to $L^2(\mathbb{I},\,\mathrm{d}x)$ and are normalized by $P_n(1)=1$. They satisfy the differential equation:*

$$(1-x^2)\pi_n''(x)-2x\pi_n'(x)+n(n+1)\pi_n(x)=0, \tag{3.36}$$

and therefore, Rodrigues' formula states that:

$$P_n(x)=\frac{1}{(-2)^n n!}\frac{\mathrm{d}^n}{\mathrm{d}x^n}\left(1-x^2\right)^n. \tag{3.37}$$

Example 3.2.13. *The Chebyshev polynomials of the first kind, denoted by $T_n(x)$, are orthogonal with respect to $L^2(\mathbb{I}, (1-x^2)^{-\frac{1}{2}}\,\mathrm{d}x)$ and are normalized by $T_n(1) = 1$. By making the variable transformation $x = \cos\theta$, the inner product becomes:*

$$\langle \pi_k, \pi_j \rangle = \int_0^\pi \pi_k(\cos\theta)\pi_j(\cos\theta)\,\mathrm{d}\theta.$$

Explicitly:

$$T_n(\cos\theta) = \cos(n\theta), \quad \text{or} \quad T_n(x) = \cos(n\cos^{-1}(x)). \tag{3.38}$$

Example 3.2.14. *The Chebyshev polynomials of the second kind, denoted by $U_n(x)$, are orthogonal with respect to $L^2(\mathbb{I}, \sqrt{1-x^2}\,\mathrm{d}x)$ and normalized by $U_n(1) = n+1$. By making the variable transformation $x = \cos\theta$, the inner product becomes:*

$$\langle \pi_k, \pi_j \rangle = \int_0^\pi \pi_k(\cos\theta)\pi_j(\cos\theta)\sin^2\theta\,\mathrm{d}\theta.$$

Explicitly:

$$U_n(\cos\theta) = \frac{\sin((n+1)\theta)}{\sin\theta}, \quad \text{or} \quad U_n(x) = \frac{\sin((n+1)\cos^{-1}(x))}{\sqrt{1-x^2}}. \tag{3.39}$$

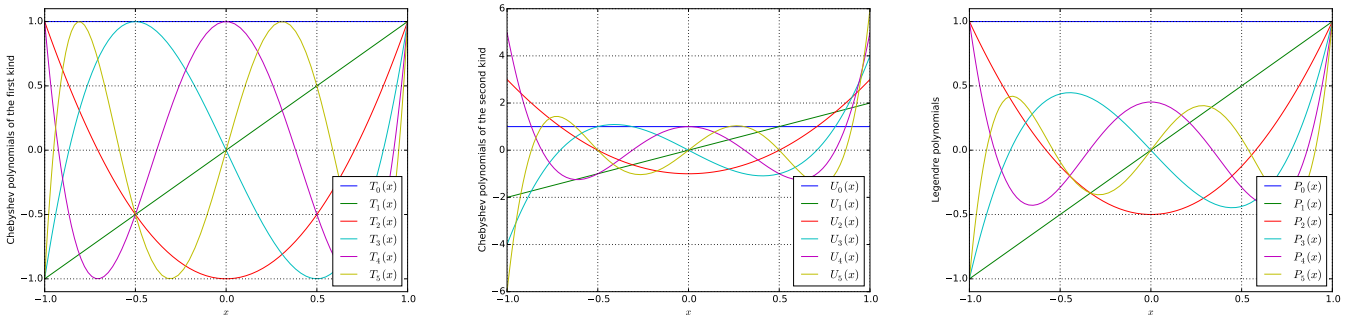
Table 3.1 describes the classical orthogonal polynomials. Note that Legendre and Chebyshev polynomials are special cases of Jacobi polynomials. Figure 3.3 shows the first six polynomials $T_n(x)$, $U_n(x)$, and $P_n(x)$.

Table 3.1: The classical orthogonal polynomials

Name	Jacobi	Laguerre	Hermite
Notation	$P_n^{(\alpha,\beta)}(x)$	$L_n(x)$	$H_n(x)$
Domain	$\mathbb{I}$	$[0,\infty)$	$\mathbb{R}$
$w(x)$	$(1-x)^\alpha(1+x)^\beta$	e^{-x}	e^{-x^2}
A_n	$\dfrac{(2n+\alpha+\beta+1)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)}$	$-\dfrac{1}{n+1}$	2
B_n	$\dfrac{(\alpha^2-\beta^2)(2n+\alpha+\beta+1)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}$	$\dfrac{2n+1}{n+1}$	0
C_n	$\dfrac{(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)}$	$\dfrac{n}{n+1}$	$2n$
$Q(x)$	$1-x^2$	x	1
κ_n	$(-2)^n n!$	$n!$	$(-1)^n$

3.2.2 Best Polynomial Approximation in L^∞ and The Remez Exchange Algorithm

We have seen that best polynomial approximation in L^2 is a rich and well-developed subject. Best polynomial approximation in L^∞ is just as fascinating, though as the results will show, it is hardly as practical.



(a) (b) (c)

Figure 3.3: The first six Chebyshev polynomials of the (a) first kind, (b) second kind, and (c) Legendre polynomials.

It turns out that best polynomial approximation in L^∞ is an old idea dating from the time of Chebyshev and Poncelet.

In this section, we consider the best polynomial approximations in $L^\infty([a,b])$, i.e. find $p_n \in \mathbb{P}_n$ such that:

$$\|f - p_n\|_\infty \leq \|f - q\|_\infty, \qquad \forall q \in \mathbb{P}_n.$$

The main result of this section is to characterize the polynomials of best approximation as **alternants**.

Theorem 3.2.15 (Chap. 10 in Trefethen [14])**.** *Every $f \in C([a,b])$ has a unique best approximation $p_n \in \mathbb{P}_n$. If f is real, then p_n is real, and in this case a polynomial $q \in \mathbb{P}_n$ is equal to p_n if and only if $f - q$ equioscillates in at least $n+2$ extreme points.*

Proof. We partition the proof.

Existence We note that $\|f - q\|_\infty$ is a continuous function of $q \in \mathbb{P}_n$. Since one candidate approximation is the zero function, we know that if p_n exists, it lies in $\{q \in \mathbb{P}_n : \|f - q\|_\infty \leq \|f\|_\infty\}$. This is a closed and bounded subset of a finite-dimensional space, hence compact (the Bolzano-Weierstrass property), and thus the minimum is attained.

Equioscillation $\Longrightarrow$ optimality Suppose f and p are real and $(f-p)(x)$ takes equal extreme values with alternating signs at $n+2$ points $x_0 < \cdots < x_{n+1}$, and suppose $\|f - q\|_\infty < \|f - p\|_\infty$ for some real polynomial $q \in \mathbb{P}_n$. Then $p - q$ must take nonzero values with alternating signs at the equisocillation points, implying that it takes the value zero in at least $n+1$ points in between. This implies that $p - q \equiv 0$, which is a contradiction.

Optimality $\Longrightarrow$ equioscillation Suppose $f - p$ equioscillates at fewer than $n+2$ points, and set $e = \|f - p\|_\infty$. Without loss of generality, suppose the leftmost extremum is one where $f - p = -e$. Then there are numbers $a < x_1 < \cdots < x_k < b$ with $k \leq n$ such that $(f-p)(x) < e$ for $x \in [a, x_1] \cup [x_2, x_3] \cup [x_4, x_5] \cup \cdots \cup$ and $(f-p)(x) > -e$ for $x \in [x_1, x_2] \cup [x_3, x_4] \cup \cdots \cup$. If we define $\delta p(x) := (x_1 - x)(x_2 - x) \cdots (x_k - x)$, then $(p - \varepsilon \delta p)(x)$ will be a better approximation than p to f for all sufficiently small $\varepsilon > 0$.

Uniqueness Suppose p is a best approximation with equioscillation extrema $x_0 < x_1 < \cdots < x_{n+1}$, and suppose $\|f - q\|_\infty \leq \|f - p\|_\infty$ for some real polynomial $q \in \mathbb{P}_n$. Then, without loss of generality, $(p - q)(x)$ must be nonpositive at $x_0, x_2, x_4, \ldots$, and nonnegative at $x_1, x_3, x_5, \ldots$. This implies that $p - q$ has roots in each of the $n + 1$ closed intervals $[x_0, x_1], [x_1, x_2], \ldots, [x_n, x_{n+1}]$. We wish to conclude that $p - q$ has at least $n + 1$ roots in total, counted with multiplicity, implying that $p = q$. To make the argument we prove by induction that $p - q$ has at least k roots in $[x_0, x_k]$ for each k. The case $k = 1$ is immediate. For the general case, suppose that $p - q$ has at least j roots in $[x_0, x_j]$ for each $j \leq k - 1$ but only $k - 1$ roots in $[x_0, x_k]$. Then there must be a simple root at x_{k-1}. By the induction hypothesis, $p - q$ must have exactly $k - 2$ roots in $[x_0, x_{k-2}]$ with a simple root at x_{k-2}, $k - 3$ roots in $[x_0, x_{k-3}]$ with a simple root at x_{k-3}, and so on down to one root in $[x_0, x_1]$, with a simple root at x_1. It follows that $p - q$ must be nonzero at x_0 and at x_k, and since the sign of $p - q$ changes at each of the simple roots $x_1, \ldots, x_{k-1}$, the signs at x_0 and x_k must be the same if k is odd and opposite if k is even. On the other hand, from the original alternation condition, we know that $p - q$ must take the same signs at x_0 and at x_k if k is even and opposite signs if k is odd.

□

Theorem 3.2.15 provides the mathematical foundation for the Remez exchange algorithm, the algorithm that iteratively computes best polynomial approximations in $L^\infty([a, b])$.

Algorithm 3.2.16. *The Remez exchange algorithm on* $[-1, 1]$. *Let* $p_n(x) = \sum_{k=0}^{n} c_k T_k(x)$.

1. *Given the points* $-1 \leq x_0 < \cdots < x_{n+1} \leq 1$, *solve the linear system of* $n + 2$ *equations:*

$$c_0 + c_1 T_1(x_i) + \cdots + c_n T_n(x_i) + (-1)^i e = f(x_i) \quad \text{for} \quad i = 0, \ldots, n + 1,$$

   for the unknowns $c_0, \ldots, c_n$ *and* e.

2. *Improve upon the points* $x_0 < \cdots < x_{n+1}$ *and their errors* $\pm e$ *by finding the local maxima:*

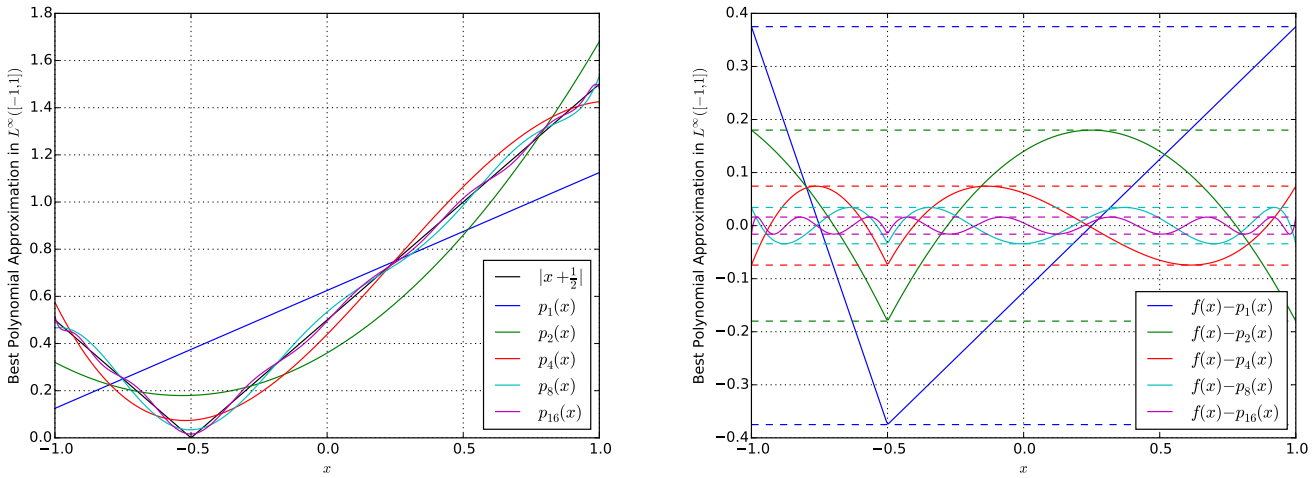
$$\bar{x}_k = \underset{x \in (x_{k-1}, x_{k+1})}{\arg\max} |f(x) - p_n(x)|, \quad \text{for} \quad k = 1, \ldots, n,$$

   and expect $x_0 = a$ *and* $x_{n+1} = b$.

3. *Define* $z_i := |f(\bar{x}_i) - p_n(\bar{x}_i)|$ *for* $i = 0, \ldots, n+1$. *Then with* $z_{\min} := \min\{z_i\}$ *and* $z_{\max} := \max\{z_i\}$ *as lower and upper bounds for the best approximation errors, iterate until* $z_{\min} \approx z_{\max}$ *in the working precision.*

The best polynomial approximation is represented in the basis of the Chebyshev polynomials of the first kind so as to render the system as well-conditioned as possible. As simple variable transformation allows for the algorithm to be applied to the general interval $[a, b]$. The Chebyshev nodes are a common choice for the initial set of points because Chebyshev polynomial interpolants give near-best polynomial approximants in $L^\infty([-1, 1])$.

Figure 3.4 shows the approximants generated by the Remez algorithm to the continuous function $f(x) = |x + \frac{1}{2}|$.



(a) (b)

Figure 3.4: The Remez algorithm for best polynomial approximation to the continuous function $f(x) = |x+\frac{1}{2}|$ on $[-1,1]$: (a) the function and the polynomial approximants (b) the error, showing equioscillation.

3.2.3 The Best Choice of Interpolation Nodes on $\mathbb{I}$

We have now studied Lagrange interpolation; we have seen how best polynomial approximation in L^2 leads to the beautiful theory of orthogonal polynomials; and, we have examined the more complicated task of best polynomial approximation in L^∞. We now have all the necessary tools to eliminate the Runge phenomenon that was depicted in Figure 3.1. Given the remainder Theorem 3.1.3, we know that for sufficiently smooth functions, the remainder is given by:

$$r(x) = f(x) - p_n(x) = \ell(x)\frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

In an effort to obtain uniformly convergent interpolating polynomials, one may ask how best to choose the interpolation points $x_0, \ldots, x_n$. Since the function $\ell(x) = \prod_{i=0}^{n}(x - x_i)$, we may ask which set of points will minimize the uniform norm:

$$\inf_{(x_0,\ldots,x_n)\in\mathbb{I}} \|\ell\|_\infty ?$$

This question is answered by the following theorem.

Theorem 3.2.17. *The Chebyshev points of the first kind (i.e. the roots of $T_n(x)$) infimize the norm $\|\ell\|_\infty$ on $\mathbb{I}$.*

Proof. From (3.38), it is clear that the Chebyshev polynomials of the first kind have unit uniform norm, $\|T_n\|_\infty = 1$. Furthermore, they *equioscillate* between their $n+1$ extrema. Invoking Theorem 3.2.15, we now see that T_n are the errors associated with *best polynomial approximations* (to wit $f(x) = x^n$ and $p_n(x) = x^n - T_n(x)$), and thus infimize the norm. Since $\ell(x)$ is monic, and since the coefficient of x^n in T_n is 2^{n-1} (using induction on the recurrence relations), then:

$$\inf_{(x_0,\ldots,x_n)\in\mathbb{I}} \|\ell\|_\infty = \|2^{-n}T_{n+1}\|_\infty = 2^{-n},$$

and the points $x_0, \ldots, x_n$ are the roots of $T_{n+1}(x)$:

$$x_k = \cos\left(\frac{k+\frac{1}{2}}{n+1}\pi\right), \quad \text{for} \quad k = 0, \ldots, n.$$

□

Using the Chebyshev points of the first kind as interpolation points leads to the drastic improvement of Figure 3.2, and furthermore the uniform approximation bound:

$$\|f - p_n\|_\infty \leq \frac{\|f^{(n+1)}\|_\infty}{2^n(n+1)!}.$$

3.3 Spline Approximation

Sometimes a global approximation such as a Lagrange interpolant is not appropriate. For example, if we must sample data at equispaced points, there is no escaping the Runge phenomenon via the selection of a different set of points. In other scenarios, data will be just too "rough" to be interpolated efficiently with a global interpolant, shown in Figure 3.5.

Figure 3.5: (Piecewise) polynomial approximation of a rough function. Left: the Lagrange interpolant p_6 oscillates through the data. Right: the **s**mooth **p**iecewise **line**ar (spline) interpolant, formed via joining consecutive data points, appears to interpolate the data more faithfully.

Given the theory we have developed for global polynomial approximation via Lagrange interpolating polynomials, it should be clear that there exists a unique solution to the interpolation problem of linear interpolating splines. Furthermore, if we start making regularity assumptions on the data, as if they were samples of a function, then we can demonstrate convergence.

Theorem 3.3.1. *Let $a = x_0 < x_1 < \cdots < x_n = b$, and let $s(x)$ be the linear interpolating spline. Then, $s \in C([a,b])$ and it is linear on each subinterval $[x_{i-1}, x_i]$, for $i = 1, \ldots, n$. Let $f(x) \in C^2([a,b])$. Then:*

$$\|f - s\|_\infty \leq \frac{h^2}{8}\|f''\|_\infty,$$

where $h = \max_{1 \leq i \leq n}(x_i - x_{i-1})$.

The nodes x_i for $i = 0, \ldots, n$ are known as the *knots* of the linear spline.

The linear interpolating spline $s(x)$ can be constructed by calculating slopes of data between knots and using a piecewise definition. Let:

$$s_i(x) := f(x_{i-1}) + \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}(x - x_{i-1}), \quad \text{for} \quad i = 1, \ldots, n.$$

Then:

$$s(x) := \begin{cases} s_1(x) & \text{for} \quad x \in [x_0, x_1], \\ \vdots & \vdots \quad\quad \vdots \\ s_i(x) & \text{for} \quad x \in [x_{i-1}, x_i], \\ \vdots & \vdots \quad\quad \vdots \\ s_n(x) & \text{for} \quad x \in [x_{n-1}, x_n]. \end{cases}$$

Note that we can take either the value of s_i or s_{i+1} at the knot x_i since they are equal.

It may occur that we would like to construct piecewise polynomial interpolants with higher polynomial degree on each subinterval. It is clear that we will want our splines to continue to be interpolatory in nature, thus we will have $2n$ interpolation conditions. With piecewise linear splines, we had $2n$ unknown coefficients and these coefficients are uniquely determined. But if we use piecewise *quadratic* splines, then we would have $3n$ unknown coefficients and still only $2n$ interpolation conditions, and generally $(m+1)n$ unknown coefficients with piecewise splines of degree-m and only $2n$ interpolation conditions.

Instead, the added unknowns in piecewise splines of higher degree allow us to enforce continuity of the interpolant, and as many of its higher order derivatives in proportion to the degree. Thus, piecewise quadratics allow for continuity of the first derivative, piecewise cubics allow for continuity of the second derivative, and so on and so forth. These continuity conditions are only applicable at the interior knots $x_1, \ldots, x_{n-1}$, and thus these conditions only give $(m-1)(n-2)$ additional conditions, where $(m-1)$ is the number of derivatives of the spline that are continuous.

Numerous methods have been proposed to add the extra conditions required to ensure that:

$$\underbrace{(2n)}_{\text{interpolation}} + \underbrace{(m-1)(n-2)}_{\text{continuity}} + \underbrace{2(m-1)}_{\text{extra}} = \underbrace{(m+1)n.}_{\text{\# degrees of freedom}}$$

Example 3.3.2. *Piecewise cubic interpolating splines enforce continuity in the first and second derivatives and require two extra conditions. Three common ways of fulfilling these conditions are:*

1. *specify $s'(x_0) = f'(x_0)$ and $s'(x_n) = f'(x_n)$ if derivative information is available;*
2. *specify $s''(x_0) = 0 = s''(x_n)$, which gives* natural *cubic splines; or,*
3. *enforce continuity of s''' at x_1 and x_n, which is usually described at the "not-a-knot" condition*[1].

3.4 Problems

1. Naïvely, one might expect to obtain a formula for the error of the derivative of the Lagrange interpolating polynomial $f' - p_n'$. However, it is not clear that $\xi = \xi(x)$ in Theorem 3.1.3 is even

[1]This can be reasoned because the continuity of s''' implies that the first two pieces are the same cubic spline on $[x_0, x_2]$, thus x_1 is not a knot, and a similar argument for x_{n-1}.

differentiable. An alternative approach leads to the following formula. Prove (and state conditions) that there exist distinct points $z_i \in (x_i, x_{i+1})$ for $i = 0, \ldots, n-1$ such that:

$$\rho_n(x) := f'(x) - p_n'(x) = \frac{f^{(n+1)}(\eta)}{n!} \prod_{i=0}^{n-1} (x - z_i),$$

for some $\eta \in (a, b)$ and for each $x \in [a, b]$. *Hint: take the error formula $r(x)$ in Theorem 3.1.3, and apply the Intermediate Value Theorem on each sub-interval (x_i, x_{i+1}) for $i = 0, \ldots, n-1$. This gives you the n points z_i where $f'(z_i) - p_n'(z_i) = 0$.*

2. It is sometimes possible to obtain explicit formulæ for the barycentric weights λ_k. Show that for the equispaced nodes in $[-1, 1]$, $x_k = -1 + 2k/n$ for $k = 0, \ldots, n$, the barycentric weights are:

$$\lambda_k = \frac{(-\frac{n}{2})^n}{n!} \binom{n}{k} (-1)^k.$$

Why may we discard the first part, depending only on n, and use the modified weights:

$$\tilde{\lambda}_k = \binom{n}{k} (-1)^k,$$

in the second barycentric formula?

3. Generalize Horner's rule for numerical evaluation of polynomials in the monomial basis to numerical evaluation of polynomial interpolants in Newton form.

4. The Chebyshev expansion of a function $f \in L^2(\mathbb{I}, (1-x^2)^{-\frac{1}{2}}\,\mathrm{d}x)$ is determined by:

$$f(x) \sim \sum_{n=0}^{\infty} c_n T_n(x), \quad \text{where} \quad c_n = \frac{\langle T_n, f \rangle}{\langle T_n, T_n \rangle}.$$

In some instances, Chebyshev expansions are known in closed form, such as:

$$f(x) = \frac{1}{x^2 + a^2} \sim \frac{1}{a\sqrt{a^2+1}} \left(1 + 2 \sum_{n=1}^{\infty} \frac{(-1)^n}{(a + \sqrt{a^2+1})^{2n}} T_{2n}(x) \right), \quad \text{for} \quad a > 0.$$

(a) Show that:

$$\langle T_0, T_0 \rangle = \pi, \quad \text{and} \quad \langle T_n, T_n \rangle = \frac{\pi}{2}, \quad \text{for} \quad n \in \mathbb{N}.$$

(b) Calculate $\|f\|_\infty$ and $\|f\|_{2,w}$, the weighted 2-norm associated with $L^2(\mathbb{I}, (1-x^2)^{-\frac{1}{2}}\,\mathrm{d}x)$. *Hint: for the weighted 2-norm, use the Chebyshev expansion and orthogonality of the Chebyshev polynomials.*

(c) Let $p_N(x)$ denote the degree-N Chebyshev expansion of f. Which $N(= N(a, \varepsilon))$ is required to satisfy:

$$\|f - p_N\|_{2,w} \leq \varepsilon \|f\|_{2,w}?$$

(d) Which $N(= N(a, \varepsilon))$ is required to satisfy:

$$\|f - p_N\|_\infty \leq \varepsilon \|f\|_\infty?$$

5. Calculate the first five orthogonal polynomials with respect to $L^2([-1, -a] \cup [a, 1],\, \mathrm{d}x)$ for some $a \in [0, 1)$. While inapplicable, Theorem 3.2.7 makes a statement on the locii of the zeros of orthogonal polynomials with respect to $L^2([a, b],\, \mathrm{d}\mu(x))$ for some monotonic measure μ. Can anything be said about the locii of the zeros in the current case? What do you expect when $\lim_{a \to 0} \pi_n(x; a)$?

6. Prove Theorem 3.3.1.

Chapter 4

Numerical Differentiation and Integration

Given a differentiable function, we can always find its derivative, although this may not be inviting if the function is complicated. Numerical differentiation approximates the derivative using only function values. Numerical differentiation formulæ can be used for solving initial-value problems and other differential equations. Integration is another problem that can be handled numerically only using function values. Numerical integration is clearly useful since there are many functions that cannot be integrated by hand nor analytically by powerful symbolic software, and often the analytical expressions that *do* exist are numerically impractical.

4.1 Finite Differences

Given $f \in C^1((a,b))$, recall that:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}.$$

For a sufficiently small h, we can make the approximation:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}.$$

Let us study the error in this approximation, which is known as the **first-order forward difference**. From Taylor's remainder theorem, if we assume $f \in C^2(a,b)$:

$$\left| f'(x) - \frac{f(x+h) - f(x)}{h} \right| \leq \frac{h}{2} \max_{\xi \in (a,b)} |f''(\xi)|.$$

Since this error term holds for all $x \in (a,b)$, we say that this approximation of first-order. If we take a small step backward instead, then we arrive at the backward difference formula:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h},$$

which has the same upper bound on the error as the forward difference.

In many applications, this is not accurate enough. Higher order schemes can be created by ensuring more terms in the Taylor series are removed as $h \to 0$ in the formula. Consider the second-order formula created by averaging the forward and backward differences:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}.$$

This is called a centered difference scheme. To see this, subtract the Taylor expansions:

$$f(x \pm h) = f(x) \pm f'(x)h + \frac{f''(x)}{2}h^2 \pm \frac{f'''(c_\pm)}{6}h^3,$$

for some constants $c_\pm \in (x, x \pm h)$ to obtain:

$$\left| f'(x) - \frac{f(x+h) - f(x-h)}{2h} \right| = \left| \frac{f'''(c_+) + f'''(c_-)}{12}h^2 \right| \leq \frac{h^2}{6} \max_{\xi \in (a,b)} |f'''(\xi)|,$$

for all $x \in (a, b)$.

It is not difficult to derive a second-order finite difference approximation of the second derivative. Assume that $f \in C^4(a, b)$. Adding the Taylor expansions:

$$f(x \pm h) = f(x) \pm f'(x)h + \frac{f''(x)}{2}h^2 \pm \frac{f'''(x)}{6}h^3 + \frac{f^{(\mathrm{iv})}(c_\pm)}{24}h^4,$$

we obtain, after some algebra:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2},$$

and indeed the approximation is second-order since:

$$\left| f''(x) - \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \right| = \left| \frac{f^{(\mathrm{iv})}(c_+) + f^{(\mathrm{iv})}(c_-)}{24}h^2 \right| \leq \frac{h^2}{12} \max_{\xi \in (a,b)} |f^{(\mathrm{iv})}(\xi)|.$$

4.1.1 Complex Differences

Consider using the Taylor expansion of f with a small perturbation into the complex plane:

$$f(x + \mathrm{i}h) = f(x) + \mathrm{i}f'(x)h - \frac{f''(x)}{2}h^2 - \mathrm{i}\frac{f'''(c)}{6}h^3,$$

where $c - x \in \mathrm{i}(0, h)$. This expansion motivates the complex finite difference formula:

$$f'(x) \approx \Im\left\{ \frac{f(x + \mathrm{i}h)}{h} \right\},$$

which is second-order since:

$$\left| f'(x) - \Im\left\{ \frac{f(x + \mathrm{i}h)}{h} \right\} \right| = \left| \frac{f'''(c)}{6}h^2 \right| \leq \frac{h^2}{6} \max_{\xi \in (x, x+\mathrm{i}h)} |f'''(\xi)|.$$

Using just a single function evaluation in the complex difference formula, we obtain second-order accuracy equivalent to the centered difference formula! Furthermore, as there is no subtraction involved in the

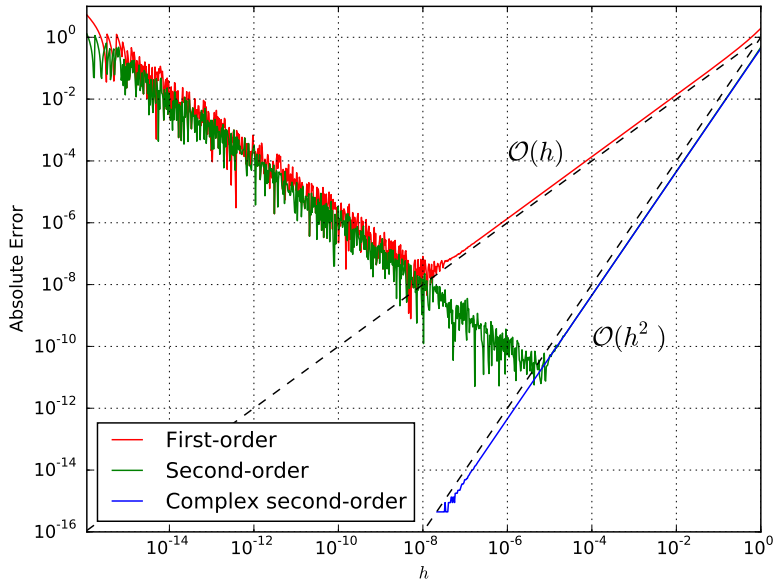


Figure 4.1: Comparison of first- and second-order finite difference formulæ and the second-order complex finite difference formula for $f(x) = e^x$ at $x = 1$. The dashed black lines are representative first- and second-order scalings.

complex finite difference formula, we expect the formula to have better numerical stability properties. Indeed, Figure 4.1 shows the resulting error when numerically differentiating $f(x) = e^x$ at $x = 1$ using the first- and second-order finite difference formulæ and the second-order complex finite difference formula.

While the complex finite difference formulæ are impressive, and relieve the numerical instability of finite difference formulæ due to subtractive cancellation, the ability to numerically evaluate a function in the complex plane assumes *analyticity*, which is much stronger than the weak assumptions of second- and third-order differentiability. So, any comparison with complex finite differencing is unfair.

4.1.2 Dual Numbers and Forward-Mode Automatic Differentiation

Let $\mathbb{F}$ be the field of real or complex numbers $\mathbb{R}$ or $\mathbb{C}$. Dual numbers extend the field $\mathbb{F}$ with a unit dual ε to the quotient ring $\mathbb{D} := \mathbb{F}[\varepsilon]/(\varepsilon^2)$, i.e. the unit dual number satisfies $\varepsilon^2 = 0$. Thus, it is clear that dual numbers obey the following rules for addition and multiplication:

$$(a + b\varepsilon) + (c + d\varepsilon) = (a + c) + (b + d)\varepsilon, \quad \text{and} \quad (a + b\varepsilon)(c + d\varepsilon) = ac + (ad + bc)\varepsilon.$$

Remarkably, the product of two dual numbers is the product of the two primal components and the dual component can be interpreted as the product rule differentiation formula.

Example 4.1.1. *Let $f(x) = x^2$. Then evaluation at the dual number $x = a + b\varepsilon$ is:*

$$f(a + b\varepsilon) = (a + b\varepsilon)^2 = a^2 + 2ab\varepsilon.$$

Dual numbers can be interpreted as evaluation at the matrix:

$$\begin{bmatrix} a & b \\ 0 & a \end{bmatrix},$$

since this matrix satisfies the addition and multiplication rules of $\mathbb{D}$.

Example 4.1.2. *Let $f(x) = e^x$. The evaluation at the dual number $x = a + b\varepsilon$ can be achieved via the matrix exponential. Since $A := \operatorname{diag}(a, a)$ and $B := \begin{bmatrix} 0 & b \\ 0 & 0 \end{bmatrix}$ commute, i.e. $AB = BA$, using the property $e^{A+B} = e^A e^B$, we find:*

$$e^{a+b\varepsilon} = e^{A+B} = e^A e^B = \begin{bmatrix} e^a & 0 \\ 0 & e^a \end{bmatrix} \begin{bmatrix} 1 & b \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} e^a & be^a \\ 0 & e^a \end{bmatrix} = e^a(1 + b\varepsilon).$$

These simple examples motivate the following definition for evaluation of a function at a dual number.

Definition 4.1.3. *Let $D \subset \mathbb{F}$. Evaluation of f at a dual number $x \in D$ is defined by:*

$$f(a + b\varepsilon) := f(a) + \lim_{h \to 0} \frac{f(a + b\varepsilon h) - f(a)}{h}. \tag{4.1}$$

When f is differentiable at a, we can identify the limit as the derivative:

$$f(a + b\varepsilon) = f(a) + f'(a)b\varepsilon.$$

Endowed with only the above properties, dual numbers are a powerful tool for numerical differentiation. In fact, the dual component of the differentiable function $f(a+b\varepsilon)$ *is* the function's derivative scaled by b, effectively implementing the chain rule differentiation formula since b is allowed to store other differentiation information. Since multiplication of two dual numbers implements the product rule differentiation formula, we have everything we need in order to evaluate complicated derivatives on a computer. In any computer programming language that allows *operator overloading*, we can *overload* the functions

```
exp(x::Dual) = exp(x.value) * (1 + x.dual * epsilon)
sin(x::Dual) = sin(x.value) + x.dual * cos(x.value) * epsilon
cos(x::Dual) = cos(x.value) - x.dual * sin(x.value) * epsilon
```

In this pseudocode, the double colon indicates a type assertion, i.e. `x` must be a number of the type `Dual`, the dot following the variable `x` accesses the field storing the primal entry `value` or the dual entry `dual`, and `epsilon` is used to indicate the unit dual number ε.

Any function that we create from this point onward that uses `exp`, `sin`, or `cos`, can return the derivative in the dual component.

Example 4.1.4. *Evaluate the derivative of $f(x) = e^{\sin x} + \cos(\sin(e^x))$ at the point $x = 1$. Using the rules outlined above, we evaluate f at the dual number $x = 1 + \varepsilon$, and obtain:*

$$f(1 + \varepsilon) = 3.236585937969746 + 2.243049618601819\varepsilon,$$

which turns out to return an exact result in 64-bit floating-point arithmetic.

This use of dual numbers is known as forward-mode automatic differentiation, and is extremely important in optimization problems, where large Jacobian vectors and Hessian matrices are required.

4.2 Newton–Cotes Quadrature

In this section and beyond, the word quadrature can be taken to mean numerical integration.

Definition 4.2.1. *Any formula of the form:*

$$\int_a^b f(x)\,\mathrm{d}x \approx \sum_{k=0}^{n} w_k f(x_k),$$

where the nodes $x_k \in [a,b]$ and the weights w_k are independent of f is called a **quadrature formula**.

We have seen that under certain conditions the Lagrange interpolating polynomials can do quite well in approximating a function f. Given the $n+1$ equispaced points $x_k = x_0 + kh$, for $k = 0,\ldots,n$, and where $h = (x_n - x_0)/n$, if $p_n(x_k) = f(x_k)$, how good of an approximation is:

$$\int_{x_0}^{x_n} f(x)\,\mathrm{d}x \approx \int_{x_0}^{x_n} p_n(x)\,\mathrm{d}x? \tag{4.2}$$

Note that the right-hand side of (4.2) can be expressed using the Lagrange basis polynomials:

$$\begin{aligned}\int_{x_0}^{x_n} p_n(x)\,\mathrm{d}x &= \int_{x_0}^{x_n} \sum_{k=0}^{n} f(x_k)\ell_{n,k}(x)\,\mathrm{d}x,\\ &= \sum_{k=0}^{n} f(x_k)\int_{x_0}^{x_n} \ell_{n,k}(x)\,\mathrm{d}x,\\ &= \sum_{k=0}^{n} w_k f(x_k),\end{aligned} \tag{4.3}$$

where the weights:

$$w_k := \int_{x_0}^{x_n} \ell_{n,k}(x)\,\mathrm{d}x, \quad \text{for} \quad k = 0,\ldots,n, \tag{4.4}$$

are independent of f. The specific form (4.2)–(4.4) of quadrature formulæ based on equispaced points are called the **Newton–Cotes** family of quadrature formulæ.

Classical examples of the Newton–Cotes family are the:

Trapezoidal Rule where the formula:

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x \approx \frac{h}{2}\left[f(x_0) + f(x_1)\right],$$

follows from:

$$\begin{aligned}\int_{x_0}^{x_1} p_1(x)\,\mathrm{d}x &= f(x_0)\int_{x_0}^{x_1} \overbrace{\frac{x - x_1}{x_0 - x_1}}^{\ell_{1,0}(x)}\,\mathrm{d}x + f(x_1)\int_{x_0}^{x_1} \overbrace{\frac{x - x_0}{x_1 - x_0}}^{\ell_{1,1}(x)}\,\mathrm{d}x,\\ &= f(x_0)\frac{(x_1 - x_0)}{2} + f(x_1)\frac{(x_1 - x_0)}{2};\ \text{and},\end{aligned}$$

Simpson's Rule with the formula:

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x \approx \frac{h}{3}\left[f(x_0)+4f(x_1)+f(x_2)\right].$$

Using Theorem 3.1.3 characterizing the remainder in the Lagrange interpolating polynomial, we can immediately derive the remainder for the Newton–Cotes family.

Theorem 4.2.2. *Suppose that $f \in C^{n+1}(x_0, x_n)$. The remainder in the Newton–Cotes family of quadrature formulæ is given by:*

$$\int_{x_0}^{x_n} [f(x)-p_n(x)]\,\mathrm{d}x = \int_{x_0}^{x_n} \frac{\ell(x)}{(n+1)!} f^{(n+1)}(\xi(x))\,\mathrm{d}x. \tag{4.5}$$

Taking the absolute value, we can bound the error by:

$$\left|\int_{x_0}^{x_n} [f(x)-p_n(x)]\,\mathrm{d}x\right| \le \frac{1}{(n+1)!}\max_{\xi\in(x_0,x_n)}|f^{(n+1)}(\xi)|\int_{x_0}^{x_n}|\ell(x)|\,\mathrm{d}x,$$

which, for the trapezoidal rule gives:

$$\left|\int_{x_0}^{x_n} f(x)\,\mathrm{d}x - \frac{h}{2}\left[f(x_0)+f(x_1)\right]\right| \le \frac{(x_1-x_0)^3}{12}\max_{\xi\in(x_0,x_n)}|f''(\xi)|.$$

In fact, we can prove tighter results using the Integral Mean Value Theorem.

Theorem 4.2.3. *Suppose $f \in C^2(x_0, x_1)$. Then:*

$$\int_{x_0}^{x_1} f(x)\,\mathrm{d}x - \frac{h}{2}\left[f(x_0)+f(x_1)\right] = -\frac{(x_1-x_0)^3}{12}f''(\xi), \quad \textit{for some} \quad \xi\in(x_0,x_1).$$

For $n > 1$, the bounds generated by Theorem 4.2.2 are even more pessimistic. In fact, on can prove better results such as:

Theorem 4.2.4. *Suppose $f \in C^4(x_0, x_2)$. Then:*

$$\left|\int_{x_0}^{x_2} f(x)\,\mathrm{d}x - \frac{h}{3}\left[f(x_0)+4f(x_1)+f(x_2)\right]\right| \le \frac{(x_2-x_0)^5}{720}\max_{\xi\in(x_0,x_2)}|f^{(\mathrm{iv})}(\xi)|.$$

Proof. Since $h = (x_2-x_0)/2 = x_1-x_0 = x_2-x_0$, consider the difference $f(x_0)-2f(x_1)+f(x_2) = f(x_1-h)-2f(x_1)+f(x_1+h)$. Using the Taylor expansions:

$$\begin{aligned} f(x_1-h) \quad & f(x_1)-hf'(x_1)+\tfrac{1}{2}h^2f''(x_1)-\tfrac{1}{6}h^3f'''(x_1)+\tfrac{1}{24}h^4f^{(\mathrm{iv})}(\xi_1)\\ -2f(x_1) &= -2f(x_1)+\\ +f(x_1+h) \quad & f(x_1)+hf'(x_1)+\tfrac{1}{2}h^2f''(x_1)+\tfrac{1}{6}h^3f'''(x_1)+\tfrac{1}{24}h^4f^{(\mathrm{iv})}(\xi_2), \end{aligned}$$

for some $\xi_1 \in (x_0, x_1)$ and $\xi_2 \in (x_1, x_2)$, and hence:

$$f(x_0)-2f(x_1)+f(x_2) = h^2f''(x_1)+\tfrac{1}{24}h^4\left[f^{(\mathrm{iv})}(\xi_1)+f^{(\mathrm{iv})}(\xi_2)\right] = h^2f''(x_1)+\tfrac{1}{12}h^4f^{(\mathrm{iv})}(\xi_3), \tag{4.6}$$

for some $\xi_3 \in (\xi_1, \xi_2) \subset (x_0, x_2)$, due to the Intermediate Value Theorem. Now, for any $x \in [x_0, x_2]$, we again use Taylor expansions to deduce:

$$\begin{aligned}
\int_{x_0}^{x_2} f(x)\,\mathrm{d}x &= f(x_1)\int_{x_1-h}^{x_1+h} \mathrm{d}x + f'(x_1)\int_{x_1-h}^{x_1+h}(x-x_1)\,\mathrm{d}x \\
&\quad + \tfrac{1}{2}f''(x_1)\int_{x_1-h}^{x_1+h}(x-x_1)^2\,\mathrm{d}x + \tfrac{1}{6}f'''(x_1)\int_{x_1-h}^{x_1+h}(x-x_1)^3\,\mathrm{d}x \\
&\quad + \tfrac{1}{24}\int_{x_1-h}^{x_1+h} f^{(\mathrm{iv})}(\eta_1(x))(x-x_1)^4\,\mathrm{d}x \\
&= 2hf(x_1) + \tfrac{1}{3}h^3 f''(x_1) + \tfrac{1}{60}h^5 f^{(\mathrm{iv})}(\eta_2) \\
&= \tfrac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] + \tfrac{1}{60}h^5 f^{(\mathrm{iv})}(\eta_2) - \tfrac{1}{36}h^5 f^{(\mathrm{iv})}(\xi_3), \\
&= \int_{x_0}^{x_2} p_2(x)\,\mathrm{d}x + \frac{1}{180}\left(\frac{x_2-x_0}{2}\right)^5 (3f^{(\mathrm{iv})}(\eta_2) - 5f^{(\mathrm{iv})}(\xi_3)),
\end{aligned}$$

where $\eta_1(x), \eta_2 \in (x_0, x_2)$, and where we have used the Integral Mean Value Theorem and (4.6). Thus, taking moduli, we have:

$$\left|\int_{x_0}^{x_2} \left[f(x) - p_2(x)\right]\,\mathrm{d}x\right| \leq \frac{8}{2^5 \cdot 180}(x_2-x_0)^5 \max_{\xi\in(x_0,x_2)} |f^{(\mathrm{iv})}(\xi)|.$$

□

In fact, it is possible to compute a slightly stronger bound.

Theorem 4.2.5 (Theorem 7.2 in Süli and Mayers [1])**.** *Suppose $f \in C^4(x_0, x_2)$. Then:*

$$\int_{x_0}^{x_2} f(x)\,\mathrm{d}x - \frac{h}{3}\left[f(x_0) + 4f(x_1) + f(x_2)\right] = -\frac{(x_2-x_0)^5}{2880} f^{(\mathrm{iv})}(\xi),$$

for some $\xi \in (x_0, x_2)$.

Remark 4.2.6. The n-point Newton–Cotes formulæ are exact if $f \in \mathbb{P}_n$ since $f \in \mathbb{P}_n \implies p_n \equiv f$. However, in certain cases, such as Simpson's rule, the degree of exactness is even higher: indeed, if $f \in \mathbb{P}_3$, then $f^{(\mathrm{iv})} \equiv 0$ and the formula is exact.

4.2.1 Composite Formulæ

In polynomial interpolation, we have seen oscillations and instability set in when approximating high-degree polynomials. One simple fix was to use splines. In direct analogy, for numerical integration we can split an integration interval $[a, b] = [x_0, x_n]$ into n equal subintervals $[x_{i-1}, x_i]$ for $i = 1, \ldots, n$. Then, we can use a **composite rule**:

$$\int_a^b f(x)\,\mathrm{d}x = \int_{x_0}^{x_n} f(x)\,\mathrm{d}x = \sum_{i=1}^{n}\int_{x_{i-1}}^{x_i} f(x)\,\mathrm{d}x,$$

in which integration through each subinterval is approximated by quadrature. Thus, rather than increase the degree of the polynomials to attain high accuracy, we can increase the number of subintervals.

The first two composite rules are the:

Composite Trapezoidal Rule where the formula:

$$\begin{aligned}\int_{x_0}^{x_n} f(x)\,\mathrm{d}x &= \sum_{i=1}^{n}\left[\frac{h}{2}\left[f(x_{i-1})+f(x_i)\right]-\frac{h^3}{12}f''(\xi_i)\right],\\ &= \frac{h}{2}\left[f(x_0)+2f(x_1)+\cdots+2f(x_{n-1})+f(x_n)\right]+e_h^{\mathrm{T}},\end{aligned}$$

where $\xi_i \in (x_{i-1}, x_i)$, $h = x_i - x_{i-1} = (x_n - x_0)/n = (b-a)/n$, and the error e_h^{T} is given by:

$$e_h^{\mathrm{T}} = -\frac{h^3}{12}\sum_{i=1}^{n} f''(\xi_i) = -\frac{nh^3}{12}f''(\xi) = -(b-a)\frac{h^2}{12}f''(\xi),$$

for some $\xi \in (a, b)$, using the Intermediate Value Theorem n times.

Composite Simpson's Rule where the formula:

$$\begin{aligned}\int_{x_0}^{x_n} f(x)\,\mathrm{d}x &= \sum_{i=1}^{n/2}\left[\frac{h}{3}\left[f(x_{2i-2})+4f(x_{2i-1})+f(x_{2i})\right]-\frac{(2h)^5}{2880}f^{(\mathrm{iv})}(\xi_i)\right],\\ &= \frac{h}{3}\left[f(x_0)+4f(x_1)+2f(x_2)+\cdots+2f(x_{n-2})+4f(x_{n-1})+f(x_n)\right]+e_h^{\mathrm{S}},\end{aligned}$$

where $\xi_i \in (x_{2i-2}, x_{2i})$, n is an even integer, $h = x_{2i} - x_{2i-1} = (x_n - x_0)/n = (b-a)/n$, and the error e_h^{S} is given by:

$$e_h^{\mathrm{S}} = -\frac{(2h)^5}{2880}\sum_{i=1}^{n/2} f^{(\mathrm{iv})}(\xi_i) = -\frac{n/2(2h)^5}{2880}f^{(\mathrm{iv})}(\xi) = -(b-a)\frac{h^4}{180}f^{(\mathrm{iv})}(\xi),$$

for some $\xi \in (a, b)$, using the Intermediate Value Theorem $n/2$ times.

Figure 4.2 shows the resulting error when numerically integrating $f(x) = e^x$ on $\mathbb{I}$ using the composite trapezoidal and Simpson's rules.

4.3 Richardson Extrapolation

Richardson extrapolation [15] is an algorithm designed to transform, in certain cases, the asymptotic expansion of the error in a formula into a systematic method that exploits structure of the asymptotic expansion in order to provide a more accurate result. The result is known as an *extrapolant* as it is usually beyond reach. Suppose we are interested in the quantity A and we can approximate it by A_h, where $h > 0$ is some tuneable parameter. Furthermore, suppose we are aware of the asymptotic expansion:

$$A_h = A + c_1 h + c_2 h^2 + \cdots + c_n h^n + \mathcal{O}(h^{n+1}), \quad \text{as} \quad h \to 0.$$

Such an expansion is motivated by Taylor's theorem, and as such Richardson extrapolation can improve the approximation of derivatives by finite differences and the approximation of integrals by Newton–Cotes quadrature. The next step is to assume we are in the régime where h is truly small, such that we only incur a small error in setting:

$$\hat{A} + \hat{c}_1 h + \hat{c}_2 h^2 + \cdots + \hat{c}_n h^n = A_h. \tag{4.7}$$

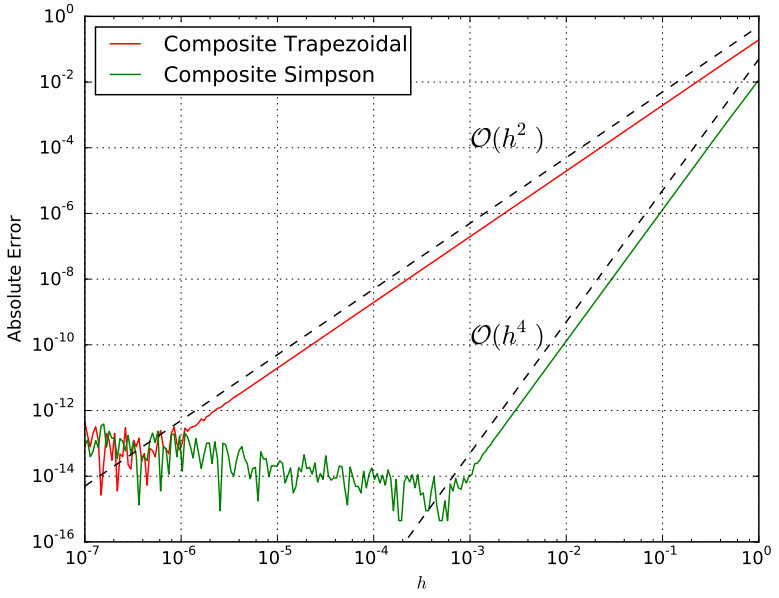


Figure 4.2: Comparison of the composite trapezoidal and Simpson's rules for $\int_{-1}^{1} e^x \, \mathrm{d}x$. The dashed black lines are representative second- and fourth-order scalings.

To solve for the n unknowns $\hat{A}$ and $\hat{c}_k$ for $k = 1, \ldots, n$, we use $n + 1$ distinct values of h to setup and solve the linear system:

$$\begin{bmatrix} 1 & h_0 & \cdots & h_0^n \\ 1 & h_1 & \cdots & h_1^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & h_n & \cdots & h_n^n \end{bmatrix} \begin{bmatrix} \hat{A} \\ \hat{c}_1 \\ \vdots \\ \hat{c}_n \end{bmatrix} = \begin{bmatrix} A_{h_0} \\ A_{h_1} \\ \vdots \\ A_{h_n} \end{bmatrix}. \tag{4.8}$$

The matrix in (4.8) is known as the Vandermonde matrix. While a direct method would require $\mathcal{O}(n^3)$ flops for the solution of the unknowns $\hat{A}$ and $\hat{c}_k$, recall that we are only interested in the extrapolant $\hat{A}$: the unknowns $\hat{c}_k$ are auxiliary variables introduced to satisfy the asymptotic expansion. Thus, we may intuitively expect to only require $\mathcal{O}(n^2)$ operations to only solve for the unknown $\hat{A}$.

We turn to an already familiar tool for the solution. First, we rewrite (4.7) as follows:

$$\hat{c}_1 + \hat{c}_2 h + \cdots + \hat{c}_n h^{n-1} = \frac{A_h - \hat{A}}{h}.$$

Since the left-hand side is a degree-$(n-1)$ polynomial in h, the n^{th} Newton divided difference will annihilate it:

$$0 \equiv \delta^{(n)}[h_0, \ldots, h_n] \left(\hat{c}_1 + \hat{c}_2 h + \cdots + \hat{c}_n h^{n-1} \right) = \delta^{(n)}[h_0, \ldots, h_n] \left(\frac{A_h - \hat{A}}{h} \right).$$

Since the unknown $\hat{A}$ is a constant (in terms of the parameter h), we may isolate for it as follows:

$$\hat{A} = \frac{\delta^{(n)}[h_0, \ldots, h_n]\left(\frac{A_h}{h}\right)}{\delta^{(n)}[h_0, \ldots, h_n]\left(\frac{1}{h}\right)}.$$

Since Newton divided differences can be computed in $\mathcal{O}(n^2)$ by organizing the calculation in a tableau, we now have a fast method for computing Richardson extrapolants.

Example 4.3.1. *Let us approximate π by inscribed polygons in the unit circle. For a regular n-gon, the circumference is $2n\sin(\pi/n) \leq 2\pi$, so let $C_n = n\sin(\pi/n) \leq \pi$ be half the circumference. Indeed, $C_2 = 2\sin(\pi/2) = 2$. Furthermore, we can compute a sequence of circumferences recursively:*

$$\begin{aligned} C_{2n} = 2n\sin(\pi/2n) &= 2n\sqrt{\frac{1}{2}(1-\cos(\pi/n))}, \\ &= 2n\sqrt{\frac{1}{2}\left(1-\sqrt{1-\sin^2(\pi/n)}\right)}, \\ &= 2n\sqrt{\frac{1}{2}\left(1-\sqrt{1-(C_n/n)^2}\right)}. \end{aligned}$$

Since this expression is sensitive to rounding errors, we rewrite it as:

$$C_{2n} = \frac{C_n\sqrt{2}}{\sqrt{1+\sqrt{1-(C_n/n)^2}}}.$$

If we let $h = 1/n^2$, then:

$$C_{1/\sqrt{h}} = A_h = \frac{\sin(\pi\sqrt{h})}{\sqrt{h}} = \pi - \frac{\pi^3 h}{6} + \frac{\pi^5 h^2}{120} + \cdots + .$$

Table 4.1 shows the result of applying Richardson extrapolation when $h = 1/n^2$ for $n = 2, 4, 8, \ldots$. Bold digits represent correct values when approximating π.

4.4 Gaussian Quadrature

While the family of (composite) Newton–Cotes quadrature formulæ take equispaced nodes, if we relax this restriction, then the resulting quadrature formula can be more accurate. Consider the problem of choosing the nodes and weights in the quadrature formula:

$$\int_a^b f(x)w(x)\,\mathrm{d}x \approx \sum_{k=1}^{n} w_k f(x_k)$$

so that the rule is exact for polynomials of degree as high as possible. Being optimistic, with n nodes and weights at our disposal, we hope it will satisfy $2n$ exactness conditions of the type:

$$\int_a^b x^m w(x)\,\mathrm{d}x = \sum_{k=1}^{n} w_k x_k^m, \quad \text{for} \quad m = 0, \ldots, 2n-1.$$

Table 4.1: Richardson extrapolation of the circumference of an n-gon.

n	C_n	Richardson Extrapolants
2	2.000000000000000	2.000000000000000
4	2.8284271247461903	**3.1**04569499661587
8	**3.**0614674589207187	**3.141**4527750222714
16	**3.1**21445152258053	**3.14159**25775443454
32	**3.1**365484905459398	**3.1415926535**83072
64	**3.14**0331156954754	**3.1415926535897**44
128	**3.141**277250932774	**3.14159265358979**5
256	**3.1415**138011443027	**3.14159265358979**5
$+\infty$	$\mathbf{3.141592653589793}\cdots$	$\mathbf{3.141592653589793}\cdots = \pi$

The word "hope" was used in the previous sentence because these last equations, while linear in the weights, are nonlinear in the nodes. Hence, these might *a priori* be complex numbers or even real numbers outside of (a, b). In either case, the quadrature formula would be useless. Fortunately, the following theorem relates the nodes of the Gaussian quadrature formula to the zeros of the orthogonal polynomials, and by construction, it follows that these nodes are located within the interval (a, b).

Theorem 4.4.1. *Let $w \geq 0$ be a non-negative weight function on $[a, b]$ and let $a, b, \in \mathbb{R}$. Then:*

1. *the nodes of the Gaussian quadrature formula are the zeros of the orthogonal polynomial π_n with respect to $L^2([a, b], w(x)\,\mathrm{d}x)$;*
2. *The formula:*
$$\int_a^b f(x)w(x)\,\mathrm{d}x = \sum_{k=1}^{n} w_k f(x_k),$$
is exact for every $f \in \mathbb{P}_{2n-1}$; and,
3. *The weights $w_k > 0$ for $k = 1, \ldots, n$.*

Proof. Let π_n be the degree-n orthogonal polynomial with respect to $L^2([a, b], w(x)\,\mathrm{d}x)$, and let $x_1, \ldots, x_n$ be its zeros. If $f \in \mathbb{P}_{2n-1}$, there exists $q, r \in \mathbb{P}_{n-1}$ such that $f = q\pi_n + r$. Since x_k are the roots of π_n, it follows that $f(x_k) = r(x_k)$ for $k = 1, \ldots, n$. Hence:

$$r(x) = \sum_{k=1}^{n} \ell_{n,k}(x) f(x_k),$$

where $\ell_{n,k}(x) \in \mathbb{P}_{n-1}$ for $k = 1, \ldots, n$ are the Lagrange basis polynomials through the roots of π_n. We integrate to obtain:

$$\begin{aligned}
\int_a^b f(x)w(x)\,\mathrm{d}x &= \int_a^b \left(q(x)\pi_n(x) + r(x)\right) w(x)\,\mathrm{d}x, \\
&= \int_a^b q(x)\pi_n(x)w(x)\,\mathrm{d}x + \int_a^b r(x)w(x)\,\mathrm{d}x, \\
&= 0 + \sum_{k=1}^{n} w_k f(x_k),
\end{aligned}$$

where the $q\pi_n$ integrates to 0 by orthogonality and where:

$$w_k = \int_a^b \ell_{n,k}(x)w(x)\,\mathrm{d}x, \quad \text{for} \quad k = 1, \ldots, n.$$

Clearly, 1. holds by applying Theorem 3.2.7, 2. holds without loss of generality, and 3. holds by repeating the argument with the squared Lagrange Basis polynomials $\ell_{n,k}^2$. □

Remark 4.4.2. Note that due to the limiting form of the Lagrange basis polynomials:

$$w_k = \int_a^b \frac{\tilde{\pi}_n(x)w(x)\,\mathrm{d}x}{\tilde{\pi}_n'(x_k)(x - x_k)}, \quad \text{for} \quad k = 1, \ldots, n. \tag{4.9}$$

To find a more convenient expression for the weights w_k, recall the Christoffel–Darboux formula (3.33):

$$\sum_{j=0}^{n-1} \tilde{\pi}_j(x)\tilde{\pi}_j(y) = \sqrt{\beta_n}\frac{\tilde{\pi}_n(x)\tilde{\pi}_{n-1}(y) - \tilde{\pi}_{n-1}(x)\tilde{\pi}_n(y)}{x - y}.$$

If we let $y = x_k$ in (3.33), where x_k is a zero of π_n, then:

$$\sum_{j=0}^{n-1} \tilde{\pi}_j(x)\tilde{\pi}_j(x_k) = \sqrt{\beta_n}\frac{\tilde{\pi}_n(x)\tilde{\pi}_{n-1}(x_k)}{x - x_k}. \tag{4.10}$$

Multiplying by $w(x)$, integrating over $[a, b]$, and comparing the result with (4.9), we find:

$$1 = \sqrt{\beta_n}\tilde{\pi}_{n-1}(x_k)\int_a^b \frac{\tilde{\pi}_n(x)w(x)\,\mathrm{d}x}{x - x_k} = \sqrt{\beta_n}\tilde{\pi}_{n-1}(x_k)\tilde{\pi}_n'(x_k)w_k.$$

On the other hand, if we take the limit as $x \to x_k$ in (4.10), then:

$$\sum_{j=0}^{n-1}[\tilde{\pi}_j(x_k)]^2 = \sqrt{\beta_n}\tilde{\pi}_n'(x_k)\tilde{\pi}_{n-1}(x_k). \tag{4.11}$$

Comparing these two equations, we find:

$$w_k^{-1} = \sum_{j=0}^{n-1}[\tilde{\pi}_j(x_k)]^2, \quad \text{for} \quad k = 1, \ldots, n. \tag{4.12}$$

This theorem gives the nodes of the Gaussian quadrature formula as the roots of the associated orthogonal polynomials. The following theorem, motivated by Theorems 3.2.8, 3.2.10, and 4.4.1, represents the nodes of Gaussian quadrature (zeros of orthogonal polynomials) as the eigenvalues of a special matrix, and the weights are given in terms of the first components of the eigenvectors.

Theorem 4.4.3. *Let α_n and β_n be the recurrence coefficients defined in Theorem 3.2.8 for the monic orthogonal polynomials $\hat{\pi}$:*

$$\alpha_n = \frac{\langle \hat{\pi}_n, x\hat{\pi}_n\rangle}{\langle \hat{\pi}_n, \hat{\pi}_n\rangle} \quad \text{and} \quad \beta_n = \frac{\langle \hat{\pi}_n, \hat{\pi}_n\rangle}{\langle \hat{\pi}_{n-1}, \hat{\pi}_{n-1}\rangle}.$$

The eigenvalues of the symmetric **Jacobi matrix***:*

$$J := \begin{bmatrix} \alpha_0 & \sqrt{\beta_1} & & & \\ \sqrt{\beta_1} & \alpha_1 & \sqrt{\beta_2} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\beta_{n-2}} & \alpha_{n-2} & \sqrt{\beta_{n-1}} \\ & & & \sqrt{\beta_{n-1}} & \alpha_{n-1} \end{bmatrix},$$

are the nodes of the Gaussian quadrature formula. Furthermore, let $\mu_0 = \int_a^b w(x)\,\mathrm{d}x$ *and let* q_k *denote the* k^{th} *normalized eigenvector of the Jacobi matrix* J*. Then the weights are given by:*

$$w_k = [q_k]_1^2 \mu_0. \tag{4.13}$$

Proof. We write the recurrence relation:

$$\hat{\pi}_{n+1}(x) = (x - \alpha_n)\hat{\pi}_n(x) - \beta_n \hat{\pi}_{n-1}(x),$$

in matrix form for $k = 0, \ldots, n-1$, and obtain:

$$\hat{J} \begin{bmatrix} \hat{\pi}_0 \\ \hat{\pi}_1 \\ \vdots \\ \hat{\pi}_{n-1} \end{bmatrix} = x \begin{bmatrix} \hat{\pi}_0 \\ \hat{\pi}_1 \\ \vdots \\ \hat{\pi}_{n-1} \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \vdots \\ \hat{\pi}_n \end{bmatrix},$$

where:

$$\hat{J} = \begin{bmatrix} \alpha_0 & 1 & & & \\ \beta_1 & \alpha_1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{n-2} & \alpha_{n-2} & 1 \\ & & & \beta_{n-1} & \alpha_{n-1} \end{bmatrix}.$$

Thus, $\tilde{\pi}_n(x_k)$ is zero if and only if x_k is an eigenvalue of the Jacobi matrix $J\tilde{\pi}(x_k) = x_k\tilde{\pi}(x_k)$. The matrix J and $\hat{J}$ only differ in the diagonal similarity transformation $J = S\hat{J}S^{-1}$, where:

$$S = \begin{bmatrix} 1 & & & \\ & 1/\sqrt{\beta_1} & & \\ & & \ddots & \\ & & & 1/\sqrt{\beta_1 \cdots \beta_{n-1}} \end{bmatrix}.$$

Since the eigenvectors q_k are normalized, $\langle q_k, q_k \rangle = 1$, they can be written as:

$$q_k = \left(\sum_{j=0}^{n-1} [\tilde{\pi}_j(x_k)]^2 \right)^{-1/2} \begin{bmatrix} \tilde{\pi}_0(x_k) \\ \tilde{\pi}_1(x_k) \\ \vdots \\ \tilde{\pi}_{n-1}(x_k) \end{bmatrix}$$

Since $\tilde{\pi}_0(x_k) \equiv \mu_0^{-1/2}$, the weights (4.12) are given by (4.13). □

Remark 4.4.4. While J and $\hat{J}$ are similar matrices and therefore have the same spectrum, it is computationally better to work with the symmetric Jacobi matrix J.

Example 4.4.5. *The monic Legendre polynomials satisfy the recurrence relation:*

$$\hat{P}_{n+1}(x) = x\hat{P}_n(x) - \frac{n^2}{4n^2-1}\hat{P}_{n-1}(x).$$

Therefore, the zeros of the Legendre polynomials are the eigenvalues of the matrix:

$$J_{\text{Legendre}} = \begin{bmatrix} 0 & \sqrt{\frac{1}{3}} & & & \\ \sqrt{\frac{1}{3}} & 0 & \sqrt{\frac{4}{15}} & & \\ & \ddots & \ddots & \ddots & \\ & & \sqrt{\frac{(n-2)^2}{4(n-2)^2-1}} & 0 & \sqrt{\frac{(n-1)^2}{4(n-1)^2-1}} \\ & & & \sqrt{\frac{(n-1)^2}{4(n-1)^2-1}} & 0 \end{bmatrix},$$

Since $\mu_0 = \int_{\mathbb{I}} \mathrm{d}x = 2$, for $n = 10$, for example, we can reproduce the Table 4.2. Notice the symmetry in

Table 4.2: Ten nodes and weights of Gauss quadrature with respect to $L^2(\mathbb{I}, \mathrm{d}x)$.

k	Nodes	Weights
1	-0.9739065285171719	0.06667134430868686
2	-0.8650633666889844	0.14945134915058303
3	-0.6794095682990244	0.21908636251598385
4	-0.4333953941292472	0.2692667193099954
5	-0.14887433898163138	0.29552422471475015
6	0.1488743389816315	0.2955242247147527
7	0.4333953941292473	0.26926671930999535
8	0.6794095682990242	0.21908636251598348
9	0.8650633666889844	0.14945134915058114
10	0.9739065285171715	0.06667134430868785

the nodes and weights.

Now that we can compute general Gaussian quadrature rules, it is interesting to see how quickly they converge. For this, we have the following theorem.

Theorem 4.4.6. *Let $f \in C^{2n}([a,b])$. Then $\exists \xi \in (a,b)$ such that:*

$$\int_a^b f(x)w(x)\,\mathrm{d}x - \sum_{k=1}^{n} w_k f(x_k) = \langle \hat{\pi}_n, \hat{\pi}_n \rangle \frac{f^{(2n)}(\xi)}{(2n)!}.$$

4.5 Monte Carlo Integration

This method approximates the value of an integral by the average of the integrand evaluated at many random points. It is only practical for high-dimensional integrals.

Given $f \in C([a,b])$, we wish to approximate:

$$I = \int_a^b f(x)\,\mathrm{d}x.$$

Suppose x_i are independent samples from the uniform distribution on $[a,b]$. Then, with N such random points, an approximation to I is:

$$I_N = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

We now compute two statistics, the expectation $E(I_N)$ and the standard deviation $\sigma(I_N)$:

$$\begin{aligned} E(I_N) &= \frac{b-a}{N}\sum_{i=1}^{N} E(f), \\ &= (b-a)E(f), \\ &= (b-a)\frac{\int_a^b f(x)\,\mathrm{d}x}{b-a}, \\ &= I. \end{aligned}$$

This is promising. Next, we compute the variance of I_N:

$$\begin{aligned} \operatorname{var}\left(\frac{b-a}{N}\sum_{i=1}^{N} f(x_i)\right) &= \frac{(b-a)^2}{N^2}\operatorname{var}\left(\sum_{i=1}^{N} f(x_i)\right) \\ &= \frac{(b-a)^2}{N^2} N \operatorname{var}(f). \end{aligned}$$

Recall that $\operatorname{var}(f) = E(f^2) - E(f)^2$. Then:

$$\sigma(I_N) = (\operatorname{var}(I_N))^{1/2} = \frac{(b-a)(E(f^2)-I^2)^{1/2}}{\sqrt{N}}.$$

Thus, the standard deviation, which measures the deviation from the expected value and thus is in some sense the "error" of the approximation, behaves like $N^{-1/2}$. Since the error of the trapezoidal rule is $\mathcal{O}(N^{-2})$ for $f \in C^2$, the Monte Carlo method is not competitive for one-dimensional integrals. However, in d dimensions, the error of the trapezoidal rule is $\mathcal{O}(N^{-2/d})$ while that of the Monte Carlo method is still $\mathcal{O}(N^{-1/2})$, independent of d! Hence, when $d > 4$, the Monte Carlo method is more efficient than the trapezoidal rule. As an added bonus, the Monte Carlo method is insensitive to endpoint singularities or cusps of the integrand.

4.6 Problems

1. Finite differences have many applications. Consider the first-order ordinary differential equation:

$$y' = f(t,y), \quad y(0) = y_0.$$

This is an initial-value problem (IVP).

(a) Use the forward and backward difference formulæ to derive Euler's forward and backward approximations:

$$y(t+h) \approx y(t) + hf(t, y(t)), \quad \text{and} \quad y(t+h) \approx y(t) + hf(t+h, y(t+h)).$$

(b) Use the forward/backward Euler methods to write a time-stepping code that solves the IVP $y' = y$ with $y(0) = 1$ for $t \in [0, 1]$. Keep track of your results in the form of a table, recording the step size h and the final values $y_{\text{Approx}}(1) \approx e$.

(c) Refine your time-stepping estimates using Richardson extrapolation on the table values. What is your best approximation to e?

2. Use the rules of evaluation at dual numbers to determine $f'(1)$ for $f(x) = \sqrt{\sin(e^x) + \cos(x^2)}$, i.e. evaluate $f(1+\varepsilon) = f(1) + f'(1)\varepsilon$.

3. Prove Theorem 4.2.3.

4. Go to `https://www.wolframalpha.com`, and ask "What is the rubber ducky curve?" Use a composite Newton–Cotes quadrature formula to estimate the area enclosed by the parametric curve:

$$\text{Area} = \frac{1}{2} \int_0^{2\pi} r^2(\theta) \, \mathrm{d}\theta,$$

where $r(\theta) = \sqrt{x^2(\theta) + y^2(\theta)}$. Tabulate the number of points you use and the resulting approximation. *And no, the answer is not "one duck." If you don't want to do the duck, go to* `https://gist.github.com/simcop2387/5493206` *for a full list of Wolfram's curves. Be sure to choose one that is periodic and doesn't involve the Heaviside step function, i.e. they must be plotted on* $[0, 2\pi)$*, otherwise low accuracy will result. Moose, pig, lion, elephant, mammoth, bunny, first cat, Ferrari, Statue of Liberty, etc…, all work. If you are not interested in typing in the formula, check UM Learn for the* JULIA *script* `duck.jl` *for* $x(\theta)$ *and* $y(\theta)$.

5. Hermite polynomials are orthogonal polynomials with respect to $L^2(\mathbb{R}, e^{-x^2} \, \mathrm{d}x)$. They satisfy the following recurrence:

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x), \quad H_0(x) = 1, \quad H_1(x) = 2x.$$

(a) Derive the recurrence for the *monic* Hermite polynomials $\hat{H}_n(x)$.

(b) Calculate $\mu_0 = \int_{\mathbb{R}} e^{-x^2} \, \mathrm{d}x$ by writing $\mu_0^2 = \int_{\mathbb{R}} e^{-x^2} \, \mathrm{d}x \times \int_{\mathbb{R}} e^{-y^2} \, \mathrm{d}y$ and using a variable transformation to polar coordinates ($x = r\cos\theta$, $y = r\sin\theta$, and $\mathrm{d}x \, \mathrm{d}y = r \, \mathrm{d}r \, \mathrm{d}\theta$).

(c) Use the above to construct the Jacobi matrix J_{Hermite} and use Theorem 4.4.3 to calculate the nodes and weights of Gauss–Hermite quadrature of order 10.

6. The function $B_d(s) = \displaystyle\int_{[0,1]^d} (x_1^2 + \cdots + x_d^2)^{s/2} \, \mathrm{d}x_1 \cdots \mathrm{d}x_d = \int_{[0,1]^d} \|x\|_2^s \, \mathrm{d}^d x$ is called a box integral. Box integrals appear in Jellium physics. Write a Monte Carlo integrator to approximate box integrals. Compare your results with:

(a) $B_4(-2) = \pi \log\left(2 + \sqrt{3}\right) - 2G - \dfrac{\pi^2}{8}$, where $G = 0.91596\,55941\,77219\ldots$ is Catalan's constant; and,

(b) $B_d(1) \sim \sqrt{\dfrac{d}{3}} \left(1 - \dfrac{1}{10d}\right)$ for $d = 10, 1\,000$, and $1\,000\,000$.

Chapter 5

Fourier Analysis

We will discuss interpolation and approximation theory when the function or data have an underlying periodicity. Therefore, instead of living on the unit interval $\mathbb{I}$, we start our investigations with functions that live on the unit circle $\mathbb{U}$. The main result of this section is that Fourier and Laurent coefficients of periodic functions may be obtained in only $\mathcal{O}(N \log N)$ operations by the fast Fourier transform (FFT). When Cooley and Tukey derived the first modern FFT [16] in 1965, the Americans we able to use it to determine nuclear test sites of the USSR, from seismic data, to within 15 km *anywhere on the globe*. Since then, it has become an indispensable algorithm and is essential in digital signal processing. Due to the special form of the Chebyshev polynomials of the first and second kinds, they also admit fast construction of interpolants. The availability of a fast transform from function values to approximate Chebyshev coefficients adds to the long list of fascinating properties that make them ubiquitous in numerical analysis.

Suppose we wish to interpolate the function $f : \mathbb{U} \to \mathbb{C}$ with the Laurent polynomials:

$$f(z) = \sum_{k \in \mathbb{Z}} f_k z^k. \tag{5.1}$$

Since the Laurent polynomials are the orthogonal polynomials on the unit circle $L^2(\mathbb{U},\, \mathrm{d}z)$, the Laurent coefficients f_k may be obtained by:

$$f_k = \frac{1}{2\pi \mathrm{i}} \int_{\mathbb{U}} \frac{f(z)}{z^{k+1}}\, \mathrm{d}z. \tag{5.2}$$

5.1 The Discrete Fourier Transform

Suppose we wish to interpolate the function $f : \mathbb{U} \to \mathbb{C}$ with only the Laurent modes $k = 0, \ldots, N-1$:

$$f(z) \approx f_{N-1}(z) = \sum_{k=0}^{N-1} f_k z^k. \tag{5.3}$$

Using the trapezoidal rule with equispaced points on $\mathbb{U}$:

$$f_k \approx \hat{f}_k := \frac{1}{N} \sum_{j=0}^{N-1} f(e^{2\pi \mathrm{i} j/N}) e^{-2\pi \mathrm{i} jk/N}, \quad \text{for} \quad k = 0, \ldots, N-1. \tag{5.4}$$

Let $z_j^N := e^{2\pi \mathrm{i} j/N}$ and let $\omega := e^{-2\pi \mathrm{i}/N}$. Then, the transformation from function samples $f(z_j^N)$ to approximate projections $\hat{f}_k$ can be represented as the matrix-vector product:

$$\begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{N-1} \end{bmatrix} = \frac{1}{N} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)^2} \end{bmatrix} \begin{bmatrix} f(z_0^N) \\ f(z_1^N) \\ \vdots \\ f(z_{N-1}^N) \end{bmatrix}. \tag{5.5}$$

The matrix in (5.5) is the so-called discrete Fourier transform (DFT) matrix:

$$\mathcal{F}_N := \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega & \cdots & \omega^{N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \cdots & \omega^{(N-1)^2} \end{bmatrix}. \tag{5.6}$$

5.1.1 The Inverse Discrete Fourier Transform

Now that we have the coefficients $\hat{f}_k$, it will be interesting to see what happens when we evaluate our approximation $f_{N-1}(x)$ at the equispaced points we used for their construction. In this case, we have:

$$f(e^{2\pi \mathrm{i} j/N}) = \sum_{k=0}^{N-1} e^{2\pi \mathrm{i} jk/N} \hat{f}_k, \quad \text{for} \quad j = 0, \ldots, N-1. \tag{5.7}$$

Using the same definitions as earlier, let $z_j^N := e^{2\pi \mathrm{i} j/N}$ and let $\omega := e^{-2\pi \mathrm{i}/N}$. In matrix form:

$$\begin{bmatrix} f(z_0^N) \\ f(z_1^N) \\ \vdots \\ f(z_{N-1}^N) \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^* & \cdots & \omega^{N-1*} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1*} & \cdots & \omega^{(N-1)^2*} \end{bmatrix} \begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{N-1} \end{bmatrix}. \tag{5.8}$$

Clearly, the inverse discrete Fourier transform (iDFT) matrix (5.8) is the conjugate transpose of the DFT matrix (5.6).

Theorem 5.1.1. *The DFT matrix* (5.6) *satisfies:*

$$\mathcal{F}_N \mathcal{F}_N^* = N I_N.$$

Furthermore, the matrix $\frac{1}{\sqrt{N}} \mathcal{F}_N$ *is unitary.*

5.1.2 The Fast Fourier Transform

Recall that if $\omega := e^{-2\pi \mathrm{i}/N}$, then $\omega^{kN} \equiv 1$ and $\omega^{kN/2} \equiv (-1)^k$ for every $k \in \mathbb{Z}$.

Theorem 5.1.2. *Let* $N = 2^n$*, for some* $n \in \mathbb{N}_0$*. Let* p *be the permutation* $p = (1, 3 \ldots, N-1, 2, 4, \ldots, N)^\top$ *and let* $P_N = I_N[:, p]$ *be the odd-even permutation matrix. Let* $\Omega_N := \mathrm{diag}(1, \ldots, \omega^{N-1})$*. Then:*

$$\mathcal{F}_N P_N = \begin{bmatrix} I_{N/2} & \Omega_{N/2} \\ I_{N/2} & -\Omega_{N/2} \end{bmatrix} \begin{bmatrix} \mathcal{F}_{N/2} & 0 \\ 0 & \mathcal{F}_{N/2} \end{bmatrix}. \tag{5.9}$$

Proof. Multiplying the blocks, we have:

$$\mathcal{F}_N P_N = \begin{bmatrix} \mathcal{F}_{N/2} & \Omega_{N/2}\mathcal{F}_{N/2} \\ \mathcal{F}_{N/2} & -\Omega_{N/2}\mathcal{F}_{N/2} \end{bmatrix}. \tag{5.10}$$

We insert the identity $I_{N/2} \equiv \mathcal{I}_{N/2} := \mathrm{diag}(1, \omega^N, \ldots, \omega^{(N/2-1)N})$ in the lower left block for column-scaling:

$$\mathcal{F}_N P_N = \begin{bmatrix} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^2 & \cdots & \omega^{2(N/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2(N/2-1)} & \cdots & \omega^{2(N/2-1)^2} \end{bmatrix} & \Omega_{N/2} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^2 & \cdots & \omega^{2(N/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2(N/2-1)} & \cdots & \omega^{2(N/2-1)^2} \end{bmatrix} \\ \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^2 & \cdots & \omega^{2(N/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2(N/2-1)} & \cdots & \omega^{2(N/2-1)^2} \end{bmatrix} \mathcal{I}_{N/2} & -\Omega_{N/2} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & \omega^2 & \cdots & \omega^{2(N/2-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{2(N/2-1)} & \cdots & \omega^{2(N/2-1)^2} \end{bmatrix} \end{bmatrix}. \tag{5.11}$$

The column scaling in the lower left block ensures that we recover the odd columns of $\mathcal{F}_N$. For the even columns, if we use the fact that $-\Omega_{N/2} \equiv \mathrm{diag}(\omega^{N/2}, \omega^{N/2+1}, \ldots, \omega^{N-1})$, then all the row-scaling is correct. □

Now that we have a factorization of the matrix $\mathcal{F}_N$ in terms of a sparse matrix requiring only $2N$ entries and two copies of the DFT matrix of half the original size. If we continue this process recursively, we can obtain a favourable complexity for applying the DFT matrix to a vector.

Theorem 5.1.3. *Let $N = 2^n$, for some $n \in \mathbb{N}_0$. The cost of the matrix-vector product with $\mathcal{F}_N$ is $N(2n+1)$ or $N(2\log_2(N)+1)$.*

Proof. Assume permutation matrices P_N can be applied for free (after all, they only amount to a different indexing). Let C_N denote the cost of applying the matrix $\mathcal{F}_N$. Then, from the factorization (5.9) of the matrix $\mathcal{F}_N$:

$$C_N = 2N + 2C_{N/2}.$$

Since there are n powers of 2 in N, $C_N = 2nN + NC_1$. Since $C_1 = 1$, we have proved the result. □

Example 5.1.4. *The first $N = 16$ Taylor coefficients of e^x can be computed as simply as:*

```
N = 2^4
z = exp(2*pi*im*(0:N-1)/N)
[real(fft(exp(z))/N)  1./gamma(1:N)]
```

The resulting output is shown in Table 5.1. Note, as this example shows, that the FFT can only give high absolute *accuracy in the Taylor coefficients.*

Since the development of Cooley and Tukey's *radix-2* FFT, many others have contributed to derive new algorithms that are optimal for DFTs of other lengths. Suffice it to say, these are beyond the scope of this course, but most computer implementations of the FFT are able to handle any input length. The

Table 5.1: The Taylor coefficients of e^x computed using the FFT and the analytical result.

k	FFT	$1/k!$
0	1.0000000000000477	1.000000000000000
1	1.0000000000000027	1.000000000000000
2	0.5000000000000002	0.500000000000000
3	0.16666666666666674	0.16666666666666666
4	0.041666666666666706	0.041666666666666664
5	0.008333333333333463	0.008333333333333333
6	0.001388888888888995	0.001388888888888889
7	0.00019841269841280873	0.0001984126984126984
8	$2.480158730161497 \times 10^{-5}$	$2.48015873015873 \times 10^{-5}$
9	$2.7557319223214805 \times 10^{-6}$	$2.7557319223985893 \times 10^{-6}$
10	$2.7557319215443243 \times 10^{-7}$	$2.755731922398589 \times 10^{-7}$
11	$2.5052108353074143 \times 10^{-8}$	$2.505210838544172 \times 10^{-8}$
12	$2.087675678164036 \times 10^{-9}$	$2.08767569878681 \times 10^{-9}$
13	$1.6059027718728913 \times 10^{-10}$	$1.6059043836821613 \times 10^{-10}$
14	$1.1470602245822192 \times 10^{-11}$	$1.1470745597729725 \times 10^{-11}$
15	$7.646105970593453 \times 10^{-13}$	$7.647163731819816 \times 10^{-13}$

Cooley–Tukey radix-2 FFT can be described as a *divide-and-conquer* algorithm due to the structure of the factorization (5.9).

The inverse fast Fourier Transform (iFFT) can be implemented analogously to the FFT with the added complex conjugation and without the scaling by $1/N$.

Example 5.1.5. *Suppose $N > M$. The iFFT can be used to rapidly evaluate Taylor interpolants $p_{M-1}(z)$ on an N-point equispaced grid on $\mathbb{U}$ in only $\mathcal{O}(N \log N)$ operations, by padding the M Taylor coefficients with zeros to degree $N-1$. This can offer significant savings over the generic Horner's rule requiring $\mathcal{O}(MN)$ operations.*

Remark 5.1.6. The FFT can be extended to construct interpolants using *any* Laurent modes, not just Taylor modes. The FFT can also be used to construct trigonometric interpolants using either complex exponentials or sines and cosines.

5.1.3 Aliasing

Now that we know that the polynomial with coefficients $\hat{f}_k$ is an *interpolant*, it is of interest to know how close the coefficients are to the true Laurent coefficients $f_k \approx \hat{f}_k$.

Theorem 5.1.7. *For $k = 0, \ldots, N-1$, the coefficients:*

$$\hat{f}_k = f_k + f_{k-N} + f_{k+N} + f_{k-2N} + f_{k+2N} + \cdots + .$$

Proof. We insert $f(z) = \sum_{l \in \mathbb{Z}} f_l z^l$ into (5.4) to obtain:

$$\hat{f}_k = \frac{1}{N} \sum_{j=0}^{N-1} \left(\sum_{l \in \mathbb{Z}} f_l e^{2\pi \mathrm{i} j l/N} \right) e^{-2\pi \mathrm{i} j k/N} = \frac{1}{N} \sum_{l \in \mathbb{Z}} f_l \sum_{j=0}^{N-1} e^{2\pi \mathrm{i} j (l-k)/N}.$$

Each of the sums over complex exponentials is 0 except when $(l-k)/N \in \mathbb{Z}$. In these cases, the sums over complex exponentials are all equal to N. □

Theorem 5.1.7 shows us that the approximate coefficients $\hat{f}_k$ are equal to the true Laurent coefficients f_k, with the addition of higher and lower Laurent coefficients *aliased* in. When the function we are approximating has decaying Laurent coefficients, it can be seen that the approximate coefficients $\hat{f}_k \to f_k$ as $N \to \infty$.

5.2 The Discrete Cosine Transform

Suppose we wish to interpolate the function $f : \mathbb{I} \to \mathbb{C}$ with the Chebyshev polynomials of the first kind:

$$f(x) = \sum_{k \in \mathbb{N}_0} f_k T_k(x). \tag{5.12}$$

Since the Chebyshev polynomials are the orthogonal polynomials on the unit interval $L^2(\mathbb{I}, \frac{\mathrm{d}x}{\sqrt{1-x^2}})$, and since:

$$\langle T_k, T_j \rangle = \int_{\mathbb{I}} \frac{T_k(x)T_j(x)}{\sqrt{1-x^2}}\,\mathrm{d}x = \int_0^{\pi} \cos(k\theta)\cos(j\theta)\,\mathrm{d}\theta = \begin{cases} \pi & \text{for} \quad k = j = 0, \\ \frac{\pi}{2} & \text{for} \quad k = j > 0, \\ 0 & \text{for} \quad k \neq j, \end{cases} \tag{5.13}$$

the Chebyshev-T coefficients f_k can be obtained by:

$$f_k = \frac{1}{\langle T_k, T_k \rangle} \int_{\mathbb{I}} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\,\mathrm{d}x = \frac{2-\delta_{k,0}}{\pi} \int_{\mathbb{I}} \frac{f(x)T_k(x)}{\sqrt{1-x^2}}\,\mathrm{d}x. \tag{5.14}$$

Suppose we wish to interpolate the function $f : \mathbb{I} \to \mathbb{C}$ with the first kind Chebyshev polynomials $k = 0, \ldots, N-1$:

$$f(x) \approx f_{N-1}(x) = \sum_{k=0}^{N-1} f_k T_k(x). \tag{5.15}$$

Let $x_j^N := \cos(\pi(j+\frac{1}{2})/N)$. Using Gauss–Chebyshev quadrature:

$$f_k \approx \hat{f}_k := \frac{2-\delta_{k,0}}{N} \sum_{j=0}^{N-1} f(x_j^N) T_k(\cos(\pi(j+\tfrac{1}{2})/N)), \quad \text{for} \quad k = 0, \ldots, N-1, \tag{5.16}$$

$$= \frac{2-\delta_{k,0}}{N} \sum_{j=0}^{N-1} f(x_j^N) \cos(\pi k(j+\tfrac{1}{2})/N). \tag{5.17}$$

Let $\mathcal{D}_N$ be the matrix with entries $[\mathcal{D}_N]_{k,j} = 2\cos(\pi k(j+\frac{1}{2})/N)$. This matrix is the so-called discrete cosine transform matrix of type-II (DCT-II)[1]. It can be seen that the DCT can be related to the real-to-real map of a DFT with even symmetry in the input data. Thus, a DCT can be applied in $\mathcal{O}(N \log N)$ operations.

[1]The DCT-II is also known as *the* DCT. We adopt this commonality.

5.2.1 The Inverse Discrete Cosine Transform

Since the operation of applying the DCT is fundamentally the same as the DFT except with assumptions on certain symmetries in the input and output, the DCT has an inverse. Clearly, the inverse discrete cosine transform (iDCT) matrix (5.19) is the transpose of the DCT matrix $\mathcal{D}_N$. In fact, if we evaluate the interpolant $f_{N-1}(x)$ at the Chebyshev points of the first kind, x_j^N, we get:

$$f(x_j^N) = \sum_{k=0}^{N-1} \hat{f}_k T_k(\cos(\pi(j+\tfrac{1}{2})/N)) = \sum_{k=0}^{N-1} \hat{f}_k \cos(\pi k(j+\tfrac{1}{2})/N)). \tag{5.18}$$

In matrix form:

$$\begin{bmatrix} f(x_0^N) \\ f(x_1^N) \\ \vdots \\ f(x_{N-1}^N) \end{bmatrix} = \frac{1}{2}\mathcal{D}_N^\top \begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{N-1} \end{bmatrix}. \tag{5.19}$$

Theorem 5.2.1. *The DCT matrix $\mathcal{D}_N$ satisfies:*

$$\mathcal{D}_N\mathcal{D}_N^\top = \begin{bmatrix} 4N & & & \\ & 2N & & \\ & & \ddots & \\ & & & 2N \end{bmatrix}.$$

Furthermore, the matrix $\sqrt{\frac{1}{2N}}\,\mathrm{diag}(\sqrt{\frac{1}{2}}, 1, \ldots, 1)\mathcal{D}_N$ is orthogonal.

Example 5.2.2. *We use the DCT to calculate the Chebyshev-T coefficients of e^x on $\mathbb{I}$.*

```
N = 2^4
x = cospi((0.5:N-0.5)/N)
fhatT = FFTW.r2r(exp(x),FFTW.REDFT10)/N; fhatT[1] *= 0.5;
ftrueT = besseli(0:N-1,1); ftrueT[2:end] *= 2;
@show [fhatT ftrueT]
```

The resulting output is shown in Table 5.2. Note, as this example shows, that the DCT can only give high absolute *accuracy in the Chebyshev-T coefficients.*

5.3 The Discrete Sine Transform

Suppose we wish to interpolate the function $f : \mathbb{I} \to \mathbb{C}$ with the Chebyshev polynomials of the second kind:

$$f(x) = \sum_{k\in\mathbb{N}_0} f_k U_k(x). \tag{5.20}$$

Since the Chebyshev polynomials are the orthogonal polynomials on the unit interval $L^2(\mathbb{I}, \sqrt{1-x^2}\,\mathrm{d}x)$, and since:

$$\langle U_k, U_j\rangle = \int_{\mathbb{I}} U_k(x)U_j(x)\sqrt{1-x^2}\,\mathrm{d}x = \int_0^\pi \sin((k+1)\theta)\sin((j+1)\theta)\,\mathrm{d}\theta = \begin{cases} \frac{\pi}{2} & \text{for} \quad k=j\geq 0, \\ 0 & \text{for} \quad k\neq j, \end{cases} \tag{5.21}$$

Table 5.2: The Chebyshev-T coefficients of e^x computed using the DCT and the analytical result (where $I_k(x)$ is a Bessel function.

k	DCT	$(2-\delta_{k,0})I_k(1)$
0	1.2660658777520084	1.2660658777520082
1	1.1303182079849703	1.13031820798497
2	0.27149533953407656	0.2714953395340767
3	0.04433684984866378	0.04433684984866381
4	0.005474240442093785	0.005474240442093732
5	0.0005429263119138974	0.0005429263119139442
6	$4.497732295427654 \times 10^{-5}$	$4.497732295429519 \times 10^{-5}$
7	$3.1984364625158 \times 10^{-6}$	$3.1984364624019926 \times 10^{-6}$
8	$1.992124804817033 \times 10^{-7}$	$1.9921248066727963 \times 10^{-7}$
9	$1.1036771896790056 \times 10^{-8}$	$1.103677172551734 \times 10^{-8}$
10	$5.505896578301994 \times 10^{-10}$	$5.50589607967375 \times 10^{-10}$
11	$2.4979670919635447 \times 10^{-11}$	$2.4979566169849818 \times 10^{-11}$
12	$1.0391104209722668 \times 10^{-12}$	$1.039152230678574 \times 10^{-12}$
13	$3.9919456629178285 \times 10^{-14}$	$3.991263356414398 \times 10^{-14}$
14	$1.4363510381087963 \times 10^{-15}$	$1.4237580108256627 \times 10^{-15}$
15	$6.938893903907228 \times 10^{-17}$	$4.7409261025615024 \times 10^{-17}$

the Chebyshev-U coefficients f_k can be obtained by:

$$f_k = \frac{1}{\langle U_k, U_k \rangle} \int_{\mathbb{I}} f(x) U_k(x) \sqrt{1-x^2}\, \mathrm{d}x = \frac{2}{\pi} \int_{\mathbb{I}} f(x) U_k(x) \sqrt{1-x^2}\, \mathrm{d}x. \tag{5.22}$$

Suppose we wish to interpolate the function $f : \mathbb{I} \to \mathbb{C}$ with the second kind Chebyshev polynomials $k = 0, \ldots, N-1$:

$$f(x) \approx f_{N-1}(x) = \sum_{k=0}^{N-1} f_k U_k(x). \tag{5.23}$$

Let $x_j^N := \cos(\pi(j+1)/(N+1))$ and let $\omega_j^N = \sin(\pi(j+1)/(N+1))$. Using Gauss–Chebyshev quadrature:

$$f_k \approx \hat{f}_k := \frac{2}{N+1} \sum_{j=0}^{N-1} w_j^N f(x_j^N) U_k(\cos(\pi(j+1)/(N+1))), \quad \text{for} \quad k = 0, \ldots, N-1, \tag{5.24}$$

$$= \frac{2}{N+1} \sum_{j=0}^{N-1} \left[\omega_j^N f(x_j^N)\right] \sin(\pi(k+1)(j+1)/(N+1)). \tag{5.25}$$

Let $\mathcal{D}_N$ be the matrix with entries $[\mathcal{D}_N]_{k,j} = 2\sin(\pi(k+1)(j+1)/(N+1))$. This matrix is the so-called discrete sine transform matrix of type-I (DST-I)[2]. It can be seen that the DST can be related to the real-to-real map of a DFT with odd symmetry in the input data. Thus, a DST can be applied in $\mathcal{O}(N \log N)$ operations.

[2]We will adopt the same commonality as the DCT-II when we call the DST-I *the* DST.

5.3.1 The Inverse Discrete Sine Transform

Since the operation of applying the DST is fundamentally the same as the DFT except with assumptions on certain symmetries in the input and output, the DST has an inverse. Clearly, the inverse discrete sine transform (iDST) matrix (5.27) is the transpose of the DST matrix $\mathcal{D}_N$. In fact, if we evaluate the interpolant $f_{N-1}(x)$ at the Chebyshev points of the second kind, x_j^N, we get:

$$f(x_j^N) = \sum_{k=0}^{N-1} \hat{f}_k U_k(\cos(\pi(j+1)/(N+1))) = \sum_{k=0}^{N-1} \hat{f}_k \frac{\sin(\pi(k+1)(j+1)/(N+1)))}{\omega_j^N}. \tag{5.26}$$

In matrix form:

$$\begin{bmatrix} \omega_0^N f(x_0^N) \\ \omega_1^N f(x_1^N) \\ \vdots \\ \omega_{N-1}^N f(x_{N-1}^N) \end{bmatrix} = \frac{1}{2}\mathcal{D}_N^\top \begin{bmatrix} \hat{f}_0 \\ \hat{f}_1 \\ \vdots \\ \hat{f}_{N-1} \end{bmatrix}. \tag{5.27}$$

Theorem 5.3.1. *The DST matrix $\mathcal{D}_N$ satisfies:*

$$\mathcal{D}_N \mathcal{D}_N^\top = 2(N+1) I_N.$$

Furthermore, the matrix $\dfrac{1}{\sqrt{2(N+1)}}\mathcal{D}_N$ *is orthogonal.*

Example 5.3.2. *We use the DST to calculate the Chebyshev-U coefficients of e^x on $\mathbb{I}$.*

```
N = 2^4
x = cospi((1:N)/(N+1))
wr = sinpi((1:N)/(N+1))
fhatU = FFTW.r2r(wr.*exp(x),FFTW.RODFT00)/(N+1)
@show [fhatU ftrueU]
```

The resulting output is shown in Table 5.2. Note, as this example shows, that the DST can only give high absolute *accuracy in the Chebyshev-U coefficients.*

5.4 Conversion, Differentiation, and Integration

By exploiting the structure of the discrete Fourier transform, we now have fast methods for computing DFTs, DCTs, and DSTs, and therefore fast methods of obtaining polynomial interpolants in the Taylor, Chebyshev-T, and Chebyshev-U bases. On $\mathbb{I}$, we already know from § 3.2.3 that interpolation in the Chebyshev points of the first kind is optimal[3]. Therefore, the Chebyshev bases are the most practical ways to store an approximation to a function f on a computer. In this section, we describe the techniques of numerical differentiation and integration with Chebyshev series. Suppose we have the Chebyshev-T expansion:

$$f(x) \approx f_{N-1}^{\mathrm{T}}(x) = \sum_{k=0}^{N-1} f_k^{\mathrm{T}} T_k(x), \tag{5.28}$$

[3] And interpolation in the Chebyshev points of the second kind is asymptotically optimal, up to a constant factor.

Table 5.3: The Chebyshev-U coefficients of e^x computed using the DST and the exact result.

k	DST	Exact
0	1.13031820798497	1.1303182079849698
1	0.542990679068153	0.5429906790681531
2	0.13301054954599145	0.13301054954599148
3	0.02189696176837487	0.021896961768374933
4	0.0027146315595697563	0.0027146315595697186
5	0.00026986393772582246	0.00026986393772577 11
6	$2.2389055236805698 \times 10^{-5}$	$2.2389055236813955 \times 10^{-5}$
7	$1.5936998453884536 \times 10^{-6}$	$1.5936998453382377 \times 10^{-6}$
8	$9.933094555562687 \times 10^{-8}$	$9.933094552965612 \times 10^{-8}$
9	$5.5058960808261906 \times 10^{-9}$	$5.505896079673745 \times 10^{-9}$
10	$2.7477525458179843 \times 10^{-10}$	$2.747752278683482 \times 10^{-10}$
11	$1.2469791131048173 \times 10^{-11}$	$1.2469826768142837 \times 10^{-11}$
12	$5.188349749829513 \times 10^{-13}$	$5.188642363338741 \times 10^{-13}$
13	$1.9957891548555754 \times 10^{-14}$	$1.9932612151559182 \times 10^{-14}$
14	$6.791952621236251 \times 10^{-16}$	$7.111389153842272 \times 10^{-16}$
15	$1.0449157878825003 \times 10^{-16}$	$2.3682880915332792 \times 10^{-17}$

or the Chebyshev-U expansion:

$$f(x) \approx f_{N-1}^{\mathrm{U}}(x) = \sum_{k=0}^{N-1} f_k^{\mathrm{U}} U_k(x). \tag{5.29}$$

5.4.1 Conversion

Since we have two bases, the first question that may arise is how to convert between Chebyshev-T and Chebyshev-U expansions? Using the facts that $T_k(\cos\theta) = \cos(k\theta)$ and $\sin\theta U_k(\cos\theta) = \sin((k+1)\theta)$ and the trigonometric identity:

$$\sin\theta\cos(k\theta) = \frac{1}{2}\left[\sin((k+1)\theta) - \sin((k-1)\theta)\right], \tag{5.30}$$

we can describe the change of basis from $T \Rightarrow U$ by:

$$T_0(x) = U_0(x), \tag{5.31}$$

$$T_1(x) = \frac{U_1(x)}{2}, \tag{5.32}$$

$$T_k(x) = \frac{U_k(x) - U_{k-2}(x)}{2}, \quad \text{for} \quad k \geq 2. \tag{5.33}$$

In matrix form, we can represent the change of basis by the banded conversion matrix $\mathcal{C}_N$:

$$\mathcal{C}_N = \begin{bmatrix} 1 & 0 & -\frac{1}{2} & & & \\ & \frac{1}{2} & 0 & -\frac{1}{2} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \frac{1}{2} & 0 & -\frac{1}{2} \\ & & & & \frac{1}{2} & 0 \\ & & & & & \frac{1}{2} \end{bmatrix}. \tag{5.34}$$

Then:

$$\begin{bmatrix} T_0(x) & T_1(x) & \cdots & T_{N-1}(x) \end{bmatrix} = \begin{bmatrix} U_0(x) & U_1(x) & \cdots & U_{N-1}(x) \end{bmatrix} \mathcal{C}_N. \tag{5.35}$$

Example 5.4.1. *Given the Chebyshev-T coefficients of e^x on $\mathbb{I}$, from Table 5.2, we can compute the Chebyshev-U coefficients of Table 5.3 in $\mathcal{O}(N)$ operations by applying the conversion matrix:*

$$\begin{bmatrix} T_0(x) & T_1(x) & \cdots & T_{N-1}(x) \end{bmatrix} \begin{bmatrix} \hat{f}_0^{\mathrm{T}} \\ \hat{f}_1^{\mathrm{T}} \\ \vdots \\ \hat{f}_{N-1}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} U_0(x) & U_1(x) & \cdots & U_{N-1}(x) \end{bmatrix} \mathcal{C}_N \begin{bmatrix} \hat{f}_0^{\mathrm{T}} \\ \hat{f}_1^{\mathrm{T}} \\ \vdots \\ \hat{f}_{N-1}^{\mathrm{T}} \end{bmatrix}, \tag{5.36}$$

$$= \begin{bmatrix} U_0(x) & U_1(x) & \cdots & U_{N-1}(x) \end{bmatrix} \begin{bmatrix} \hat{f}_0^{\mathrm{U}} \\ \hat{f}_1^{\mathrm{U}} \\ \vdots \\ \hat{f}_{N-1}^{\mathrm{U}} \end{bmatrix}. \tag{5.37}$$

Note that there are rounding errors on the order of machine precision.

Remark 5.4.2. The inverse conversion matrix $\mathcal{C}_N^{-1}$ can be applied in $\mathcal{O}(N)$ operations using back substitution.

5.4.2 Differentiation

We can differentiate the expansion (5.28) term-by-term using the fact that $T_k(x) = \cos(k\cos^{-1}(x))$ and:

$$\begin{aligned} \frac{\mathrm{d}T_k(x)}{\mathrm{d}x} &= -k\sin(k\cos^{-1}(x))\frac{\mathrm{d}\cos^{-1}(x)}{\mathrm{d}x} = k\frac{\sin(k\cos^{-1}(x))}{\sqrt{1-x^2}}, \\ &= k\frac{\sin(k\cos^{-1}(x))}{\sin(\cos^{-1}(x))} = kU_{k-1}(x). \end{aligned} \tag{5.38}$$

This means that the derivative of the Chebyshev expansion $f_{N-1}^{\mathrm{T}}(x)$ can be computed by a *scaling* of the coefficients, and a *change* of the basis:

$$f'(x) \approx f_{N-1}^{\mathrm{T}'}(x) = \sum_{k=0}^{N-1} k f_k^{\mathrm{T}} U_{k-1}(x) = \sum_{k=0}^{N-2} (k+1) f_{k+1}^{\mathrm{T}} U_k(x). \tag{5.39}$$

If we wanted to take the *second* derviative of f, then the inverse conversion matrix $\mathcal{C}_{N-1}^{-1}$ must convert the coefficients to new Chebyshev-T coefficients so that we can continue using the simple formula (5.38).

5.4.3 Integration

Using the relation (5.38), we can perform indefinite integration of Chebyshev-U expansions:

$$\int U_k(x)\,\mathrm{d}x = \int \frac{T'_{k+1}(x)}{k+1}\,\mathrm{d}x = \frac{T_{k+1}(x)}{k+1}. \tag{5.40}$$

This means that the indefinite integral of the Chebyshev expansion $f^{\mathrm{U}}_{N-1}(x)$ can be computed by a *scaling* of the coefficients, and a *change* of basis:

$$\int f(x)\,\mathrm{d}x \approx \int f^{\mathrm{U}}_{N-1}(x)\,\mathrm{d}x = \sum_{k=0}^{N-1} \frac{f^{\mathrm{U}}_k}{k+1} T_{k+1}(x) = f^{\mathrm{U}}_{-1} T_0(x) + \sum_{k=1}^{N} \frac{f^{\mathrm{U}}_{k-1}}{k} T_k(x). \tag{5.41}$$

Here, we added the indefinite integration constant f^{U}_{-1}.

Clenshaw–Curtis Quadrature

Clenshaw–Curtis quadrature [17] is the result of combining the conversion matrix $\mathcal{C}_N$ with the indefinite integration of Chebyshev-U expansions for the purpose of definite integration of Chebyshev-T expansions:

$$\int_{\mathbb{I}} T_0(x)\,\mathrm{d}x = \int_{\mathbb{I}} U_0(x)\,\mathrm{d}x = T_1(x)\big|^1_{-1} = 2, \tag{5.42}$$

$$\int_{\mathbb{I}} T_1(x)\,\mathrm{d}x = \int_{\mathbb{I}} \frac{U_1(x)}{2}\,\mathrm{d}x = \left.\frac{T_2(x)}{4}\right|^1_{-1} = 0, \tag{5.43}$$

$$\int_{\mathbb{I}} T_k(x)\,\mathrm{d}x = \int_{\mathbb{I}} \frac{U_k(x) - U_{k-2}(x)}{2}\,\mathrm{d}x = \frac{1}{2}\left(\frac{T_{k+1}(x)}{k+1} - \frac{T_{k-1}(x)}{k-1}\right)\Bigg|^1_{-1}, \tag{5.44}$$

$$= \frac{1+(-1)^k}{2}\left(\frac{1}{k+1} - \frac{1}{k-1}\right) = \frac{1+(-1)^k}{2}\left(\frac{2}{1-k^2}\right). \tag{5.45}$$

Notice that odd terms integrate to zero (as we expect). With the Chebyshev-T expansion (5.28), we get the definite integration formula:

$$\int_{\mathbb{I}} f(x)\,\mathrm{d}x \approx \int_{\mathbb{I}} f^{\mathrm{T}}_{N-1}(x)\,\mathrm{d}x = 2f^{\mathrm{T}}_0 + \sum_{k=2,2}^{N-1} \frac{2f^{\mathrm{T}}_k}{1-k^2}. \tag{5.46}$$

5.5 Problems

1. Consider $f(z) = \sum_{k=0}^{m} f_k z^k$ and $g(z) = \sum_{k=0}^{n} g_k z^k$. Conventionally, the product $f(z)g(z) =: h(z) = \sum_{k=0}^{m+n} h_k z^k$ can be computed, where the coefficients:

$$h_k = \sum_{j=\max\{0,k-n\}}^{\min\{k,m\}} f_j g_{k-j},$$

are the Cauchy product of the coefficients of f and g.

Estimate the complexity of obtaining all the coefficients h_k using the Cauchy product.

The FFT can be used to speed up multiplication of two functions as follows:

(a) Pad f and g with zero coefficients up to degree $m+n$ such that $f(z) = \sum_{k=0}^{m+n} f_k z^k$ and $g(z) = \sum_{k=0}^{m+n} g_k z^k$;

(b) Convert the coefficients $\{f_k\}_{k=0}^{m+n}$ and $\{g_k\}_{k=0}^{m+n}$ to samples of f and g at the $(m+n+1)^{\text{th}}$ roots of unity using the iFFT;

(c) Use the fact that, pointwise, $h(z_j^{m+n+1}) = f(z_j^{m+n+1})g(z_j^{m+n+1})$ to get samples of h; and,

(d) Convert samples of $h(z_j^{m+n+1})$ to coefficients h_k using the FFT.

Estimate the complexity of obtaining all the coefficients h_k using the FFT.

2. Theorem 5.1.7 may seem benign, but it can actually be used to prove that the maximum absolute approximation error of $\hat{f}_{N-1}(z) = \sum_{k=0}^{N-1} \hat{f}_k z^k$ to $f(z) = \sum_{k=0}^{\infty} f_k z^k$ is at most twice that of $f_{N-1}(z) = \sum_{k=0}^{N-1} f_k z^k$ with the true Laurent coefficients. Give the proof.

3. Let $N = YYYYMMDD$ be the integer corresponding to your birthdate and consider the function $f(x) = |x|$ `= abs(x)`. If N is odd, calculate the first N approximate Chebyshev-T coefficients via the DCT by adapting the JULIA code in Example 5.2.2. If N is even, calculate the first N approximate Chebyshev-U coefficients via the DST by adapting the JULIA code in Example 5.3.2. Use the coefficients and the property $T_n(1) = 1$ or $U_n(1) = n+1$ to estimate the value of the interpolant at $x = 1$. Find a classmate with opposite parity in their birthdate. Explain in your own terms whose approximation of 1 is better and why. *Note: The absolute value function is merely continuous on $\mathbb{I}$ so we don't expect a fast decay in the coefficients, which is why using tens of millions of points is somehow acceptable.*

4. In Chapter 3, we found an explicit expression for the barycentric weights of Lagrange interpolation in equispaced points. It was a hard grind that can be avoided by interpolating at the roots of the classical orthogonal polynomials. Use the fact that $T_{n+1}(x)$ is the orthogonal polynomial whose $n+1$ roots are the Chebyshev points of the first kind to assert:

$$\prod_{\substack{i=0 \\ i\neq k}}^{n} (x - x_i) = \frac{2^{-n} T_{n+1}(x)}{x - x_k}.$$

Combine it with $T'_{n+1}(x) = (n+1)U_n(x)$ to obtain the simple expression:

$$\tilde{\lambda}_k = (-1)^k \sin\left(\frac{k+\frac{1}{2}}{n+1}\pi\right),$$

for the normalized barycentric weights of polynomial interpolants at the Chebyshev points of the first kind. You may now create a Lagrange interpolant through the Chebyshev points in $\mathcal{O}(n)$ flops!

Chapter 6

Mathematical Programming

Mathematical programming is the subject of locating optimal solutions of problems or systems when they are formulated mathematically. Many important problems in science and engineering take the shape of a mathematical program. We will investigate a few algorithms for nonlinear programming and the simplex algorithm for linear programming.

6.1 Nonlinear Programming

Let $f : \mathbb{R}^n \to \mathbb{R}$, and let $g_i : \mathbb{R}^n \to \mathbb{R}$ and $h_j : \mathbb{R}^n \to \mathbb{R}$. A nonlinear program (NP) is expressed mathematically as:

$$\begin{aligned} &\text{minimize } f(x), && && \text{(6.1a)}\\ &\text{subject to } g_i(x) \leq 0, && \text{for} \quad i = 1, \ldots, m, && \text{(6.1b)}\\ &\text{and } h_j(x) = 0, && \text{for} \quad j = 1, \ldots, p. && \text{(6.1c)} \end{aligned}$$

Of course, any maximization problem may converted into a minimization problem with a sign change, and this may also be used to reformulate constraint equations.

While numerous algorithms[1] have been designed to take advantage of special structure, we will only look at two such algorithms that are applicable in the case of an unconstrained NP. In this section, we will focus solely on the NP:

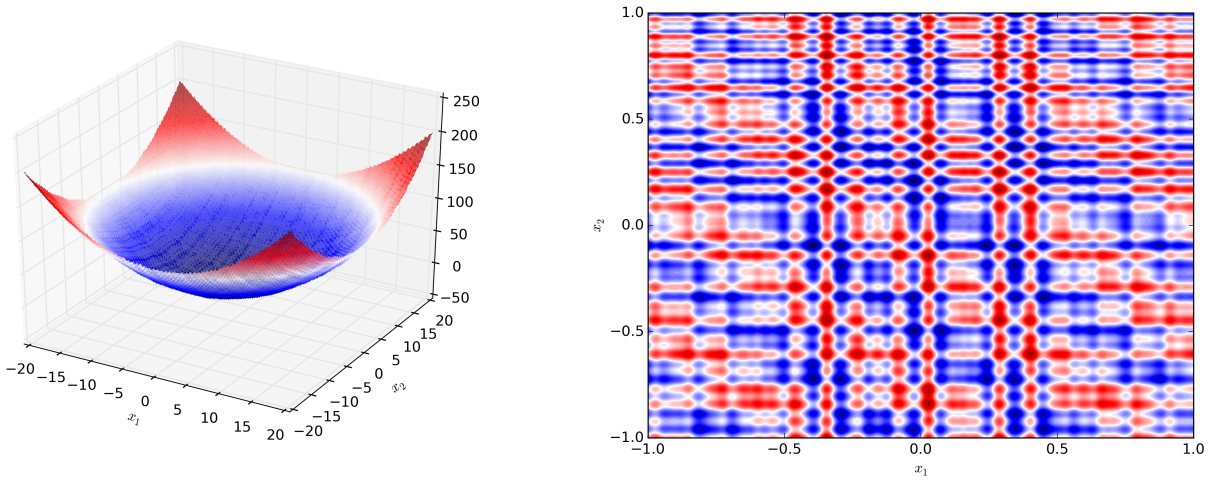
$$\begin{aligned} \min_{x \in \mathbb{R}^2} f(x) := e^{\sin(50x_1)} + \sin(60e^{x_2}) + \sin(70\sin(x_1)) \\ + \sin(\sin(80x_2)) - \sin(10(x_1 + x_2)) + (x_1^2 + x_2^2)/4, \end{aligned} \tag{6.2}$$

which appears as Problem 4 in "The SIAM 100-Digit Challenge" [18].

The first step any mathematician should take when faced with a NP is to obtain realistic bounds on the minimum and its location. For the function (6.2), on a global scale f is dominated by the quadratic term $(x_1^2 + x_2^2)/4$, since the values of the first five terms are all bounded in the intervals $[e^{-1}, e]$, $[-1, 1]$, $[-1, 1]$, $[-\sin 1, \sin 1]$, and $[-1, 1]$, respectively. Thus the overall graph of f resembles a familiar paraboloid,

[1] Some of these algorithms are even derived from the theory of Lagrange multipliers; methods which use Lagrange multipliers as relaxed constraints that iteratively tighten are known as penalty and barrier methods.

depicted in Figure 6.1 (a). A closer look, however, reveals that the trigonometric and exponential terms introduce significant complexity to the fine structure of the function. A contour plot is depicted on a zoomed-in square in Figure 6.1 (b), showing the 2720 critical points of f inside the square $[-1,1]^2$.



(a) (b)

Figure 6.1: Views of the function (6.2): (a) a surface plot on $[-20,20]^2$ showing the overall paraboloidal shape, and (b) a contour plot on $[-1,1]^2$.

If we evaluate the function at the values of x_1 and x_2 from -0.25 to 0.25 in steps of 0.01, we find that the function can be as small as -3.24. On the other hand, outside the square $[-1,1]^2$, the function is at least $e^{-1}-1-1-\sin 1-1+\frac{1}{4} > -3.23$. Therefore, the global minimum must be inside the square $[-1,1]^2$.

6.1.1 Evolutionary Algorithms

While a grid search can be effective for this particular problem (because the critical points do have a rough lattice structure), it is easy to design a more general search procedure that uses randomness to achieve good coverage of the domain. Several effective algorithms have been derived by abstracting biological processes and theories. Microevolution is a biological theory which describes the dynamics of the gene pool in the members of a species over time. Natural selection and genetic mutation during meiosis are some of the processes that contribute to microevolution. In relation to a NP, for every coordinate $x \in \mathbb{R}^n$, we may associate the *fitness value* $f(x)$.

If we begin with a generation of N_{members} random points in a realistic subdomain $D \subset \mathbb{R}^n$, and introduce an additional $N_{\text{members}} - 1$ points randomly distributed around each of the original N_{members} random points, then we may form a new generation by *culling the population* and retaining the N_{members} points-of-best-fit. While we iterate this process, we may use a shrinking scale to define the largest variability of the offspring from their parents.

Algorithm 6.1.1 (Prokaryotic Evolutionary Search)**.**
Inputs:

$f(x)$ *, the objective function;*

D , the search domain;

N_{members} , the number of points in a generation;

$N_{\text{generations}}$, the number of generations; and,

s , the scaling factor for shrinking the search domain.

Output: an upper bound to the minimum of f in D and its location.

Step 1: *Initialization:*

Let z be the centre of D; `parents` *= z; and,* `fvals = f(z).`

Step 2: *The main loop:*

```
for generation = 1:ngenerations
    children = parents
    for p in parents
        append (nmembers-1) random points in D
        centred at p to the children
    end
    fvals = f(children)
    Let parents be the set of nmembers children with lowest fvals
    Shrink D by s
end
```

Step 3: *Return:*

The smallest value in `fvals` *and the corresponding* `parent`.

This algorithm may be associated with prokaryotic evolution as the children are strictly descendants from a single parent. If, on the other hand, children may be descendants from two parents, then we obtain a eukaryotic evolutionary search. In the following algorithm, N_{members} children are allowed to be descendants from every pair of parents.

Algorithm 6.1.2 (Eukaryotic Evolutionary Search)**.**
Inputs:

$f(x)$, the objective function;

D , the search domain;

N_{members} , the number of points in a generation; and,

$N_{\text{generations}}$, the number of generations.

Output: an upper bound to the minimum of f in D and its location.

Step 1: *Initialization:*

Let `parents` *be random points in D and* `fvals = f(parents).`

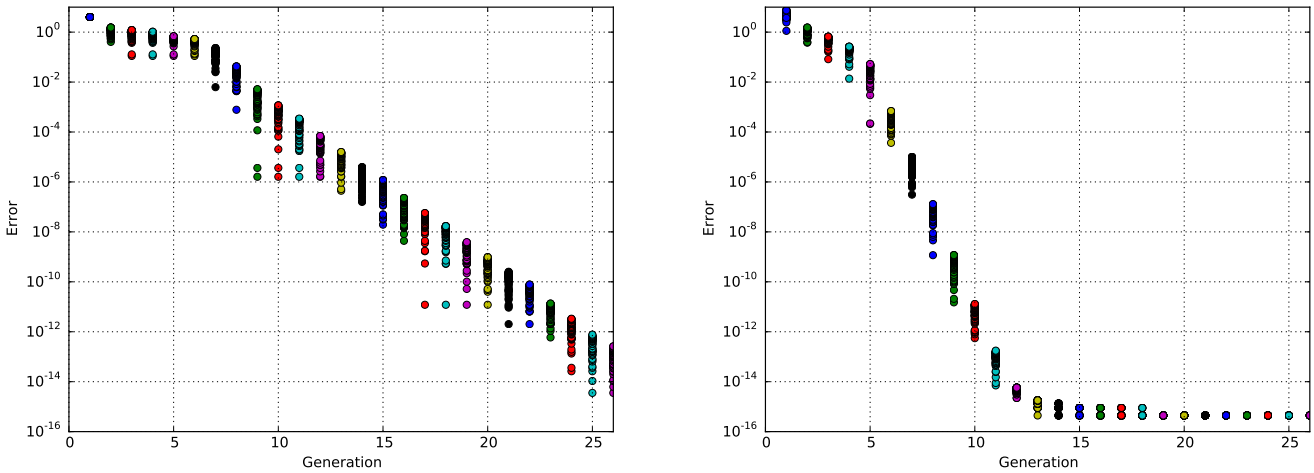
Step 2: *The main loop:*

```
for generation = 1:ngenerations
    children = []
    for p in parents, q in parents
        append a random point on the line segment
        in between p and q to the children
    end
    fvals = f(children)
    Let parents be the set of nmembers children with lowest fvals
end
```

Step 3: *Return:*

The smallest value in `fvals` *and the corresponding* `parent`.

In Figure 6.2, an execution of both evolutionary algorithms is depicted for a population with 50 members, and a scaling factor of $1/2$ for the prokaryotic search. In both (a) and (b), the data represent the discrepancy between the fitness value of every parent in the generation and the global minimum. Clearly, the eukaryotic evolutionary algorithm is able to home in on the global minimum significantly faster than the prokaryotic search. This may be explained as follows: if two parents are equidistant from the global minimum, but on opposite sides (in every coordinate), then we expect a child to appear right over top of the global minimum; on the other hand, no such estimation may be made of prokaryotic evolution.

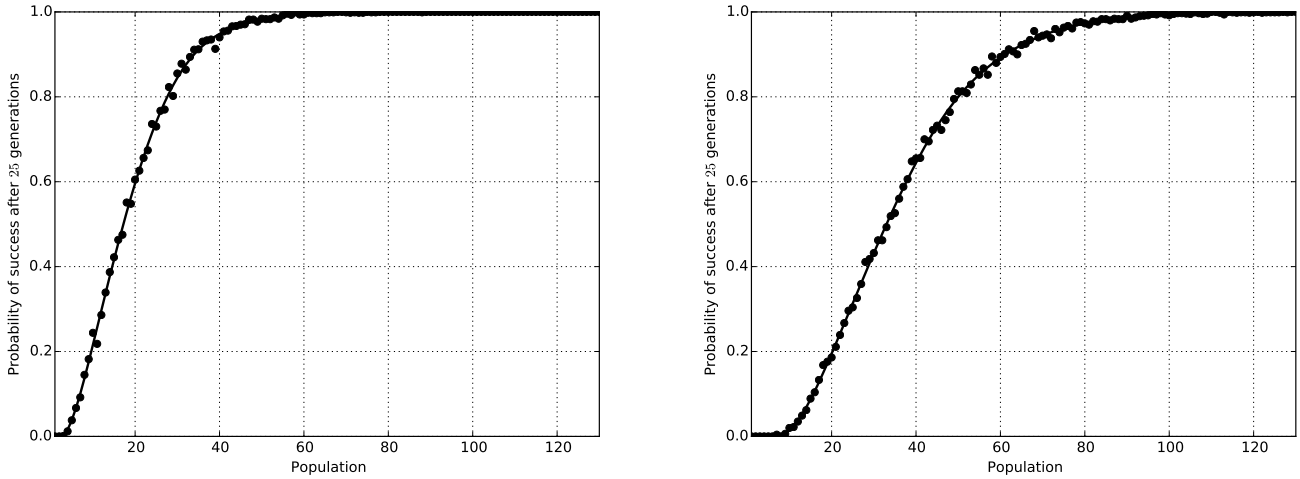


(a) (b)

Figure 6.2: The error of the evolutionary algorithms for the NP (6.2) with 50 members in the population searching through $D = [-1, 1]^2$: (a) with prokaryotic search, and (b) with eukaryotic search.

While it is reassuring that the eukaryotic search converges to the global minimum at a faster rate, it is much easier for the algorithm to end up stuck in a local minimizer. Figure 6.3 shows the probability of success after 25 generations of both evolutionary algorithms over $1,000$ iterations. Here, we recognize that the prokaryotic evolutionary search, which supports more random fluctuations throughout the generations,

spends more energy "searching" through the subdomain D rather than "settling," and is much more likely to succeed with a lower population. This is also reassuring since the earliest living organisms on Earth are thought to be prokaryotes; these organisms had many millions of years to search far and wide for the best (evolutionary) way forward, before eukaryotic search accelerated the process.



(a) (b)

Figure 6.3: The probability of success of the evolutionary algorithms over $1,000$ iterations for the NP (6.2) searching through $D = [-1, 1]^2$: (a) with prokaryotic search, and (b) with eukaryotic search. In both plots, the solid line denotes a degree-15 least-squares Chebyshev approximant.

Note that both of these algorithms may be adapted to constrained nonlinear programming by rejecting children that do not satisfy the constraints.

6.1.2 Quadratic Models

When we are in striking distance from the global minimizer, we might consider switching to a more efficient algorithm that is able to reproduce extremely high precision results. Recall that a minimizer x^* of a continuously differentiable function f is a critical point:

$$x^* = \underset{x \in \mathbb{R}^n}{\arg\min}\, f(x) \quad \Rightarrow \quad \nabla f(x^*) = 0.$$

Near a minimizer, we shall approximate a twice continuously differentiable function f by its multivariate Taylor series:

$$\begin{aligned} f(x + \delta x) = f(x) &+ \left[\nabla f(x)\right]^\top \delta x \\ &+ \frac{1}{2} \delta x^\top \left[\nabla^2 f(x)\right] \delta x + o(\|\delta x\|^2), \quad \text{as} \quad \|\delta x\| \to 0. \end{aligned}$$

To help keep the notation clean, let us introduce $g(x) := \nabla f(x)$ to be the vector of gradients of f and:

$$H(x) := \nabla^2 f(x) = \begin{bmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_1 \partial x_n} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} & \cdots & \dfrac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial^2 f}{\partial x_n \partial x_1} & \dfrac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \dfrac{\partial^2 f}{\partial x_n^2} \end{bmatrix},$$

to be the *Hessian*. Then a quadratic approximation to f is:

$$f(x + \delta x) \approx q(\delta x) = f(x) + g(x)^\top \delta x + \frac{1}{2} \delta x^\top H(x) \delta x,$$

and if we define the vector δx by the condition that $\nabla_{\delta x} q(\delta x) = 0$, then we derive Newton's iteration.

Algorithm 6.1.3 (Newton Iteration)**.**
Let $x^{(1)}$ be an initial guess for the global minimizer. For $k \in \mathbb{N}$,

1. *Solve $H(x^{(k)})\delta x^{(k)} = -g(x^{(k)})$, for $\delta x^{(k)}$; and,*
2. *Set $x^{(k+1)} = x^{(k)} + \delta x^{(k)}$.*

The first step in Newton iteration involves the solution of an n-by-n system of linear equations. This may be done with a matrix factorization in $\mathcal{O}(n^3)$ operations[2]. Now that we know how to implement Newton iteration, we may be interested in its convergence properties.

Theorem 6.1.4. *Let $f \in C^2(D)$ and let the Hessian be Lipschitz continuous, i.e. $|H_{i,j}(x) - H_{i,j}(y)| \leq K \|x - y\|$, for some $K > 0$, some norm $\|\cdot\|$, and for $x, y \in D \subset \mathbb{R}^n$. If, for some $k \in \mathbb{N}$, a Newton iterate $x^{(k)}$ is sufficiently close to $x^* \in D$, then Newton's method converges quadratically to x^*.*

Proof. The Taylor series for each term in $g(x^*)$ is:

$$0 = g(x^*) = g(x^{(k)}) - H(x^{(k)})e^{(k)} + \mathcal{O}(\left\|e^{(k)}\right\|^2),$$

where $e^{(k)} := x^{(k)} - x^*$. Let $x^{(k)}$ be in a neighbourhood of x^* such that $H(x^{(k)})^{-1}$ exists and is bounded above. Such a neighbourhood exists by continuity of $H(x)$. Then, the k^{th} iteration exists, and multiplying through by $H(x^{(k)})^{-1}$ yields:

$$0 = -\delta x^{(k)} - e^{(k)} + \mathcal{O}(\left\|e^{(k)}\right\|^2) = -e^{(k+1)} + \mathcal{O}(\left\|e^{(k)}\right\|^2),$$

by step 2 of Newton iteration and by definition of $e^{(k+1)}$. Hence by definition of the order notation, there exists a constant $M > 0$ such that:

$$\left\|e^{(k+1)}\right\| \leq M \left\|e^{(k)}\right\|^2.$$

If $x^{(k)}$ is in a neighbourhood for which $\left\|e^{(k)}\right\| < M^{-1}$, then it follows that $\left\|e^{(k+1)}\right\| < \left\|e^{(k)}\right\|$ and by induction, the error contracts and $\lim_{k\to\infty} \left\|e^{(k)}\right\| = 0$. □

[2]Quasi-Newton methods are derived by factorizing the matrix $H(x^{(1)})$ once and updating the factorization at every iteration.

6.2 Linear Programming

Linear programming (LP) is a mathematical model for optimizing a linear objective (such as maximum profit or minimum cost) constrained by a list of linear relationships. Linear programming was developed during World War II (independently by the Allies and the Soviet Union) to plan expenditures and returns in order to reduce costs to the army and increase losses to the enemy. It was kept secret until 1947.

The founders of this subject are Leonid Kantorovich, a Russian mathematician who developed LP problems in 1939, George Dantzig, who published the simplex method in 1947, and John von Neumann, who developed the theory of the duality in the same year.

An LP problem in the *standard form* is expressed mathematically as:

$$\text{minimize } f(x) := c^\top x, \tag{6.3a}$$

$$\text{subject to } Ax = b, \tag{6.3b}$$

$$\text{and } x \geq 0. \tag{6.3c}$$

where $A \in \mathbb{R}^{m \times n}$ and $m \leq n$ (usually $<$). Thus the allowable constraints on the variables are either linear equations or non-negativity bounds. The coefficients c in the linear objective function are often referred to as *costs*. An example with four variables ($n = 4$) and two equations ($m = 2$) is:

$$\text{minimize } x_1 + 2x_2 + 3x_3 + 4x_4, \tag{6.4a}$$

$$\text{subject to } \begin{array}{rcrcrcrcl} x_1 & + & x_2 & + & x_3 & + & x_4 & = & 1 \\ x_1 & & & + & x_3 & - & 3x_4 & = & \frac{1}{2} \end{array}, \tag{6.4b}$$

$$\text{and } x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0. \tag{6.4c}$$

More general LP problems can be reduced to standard form without undue difficulty. For example, maximization problems are conveniently re-formulated as $-$minimize $-f(x)$. Additionally, a general linear inequality $a^\top x \leq b$ can be transformed using a slack variable z to the equation $a^\top x + z = b$ and the bound $z \geq 0$. More general bounds $x_i \geq \ell_i$ can be dealt with by a shift of origin, and if no bound exists at all on x_i in the original problem, then the standard form can be reached by introducing non-negative variables x_i^+ and x_i^-. However, we will concentrate on the solution of problems which are already in the standard form (6.3).

It is important to realize that an LP problem in standard form may have no solution, either because there is no feasible point (the problem is *infeasible*), or because $f(x) \to -\infty$ for x in the feasible region (the problem is *unbounded*). However, usually there is no difficulty in detecting these situations, and so we will concentrate on the normal case where a solution exists, though it may not be unique.

If (6.3) is considered in more detail, it can be seen that if $m = n$, then the equations $Ax = b$ determine a unique solution x, and the objective function $c^\top x$ and the bounds $x \geq 0$ play no role. In most cases, however, $m < n$ so that the system $Ax = b$ is *underdetermined* and $n - m$ degrees of freedom remain. In particular, the system can determine only m variables, given values for the remaining $n - m$ variables. For example, the equations in (6.4b) can be rearranged as:

$$\begin{array}{rcrcrcl} x_1 & = & \frac{1}{2} & - & x_3 & + & 3x_4 \\ x_2 & = & \frac{1}{2} & & & - & 4x_4 \end{array}, \tag{6.5}$$

which determines x_1 and x_2 given values for x_3 and x_4. Alternatively, the same equations can be rearranged as:

$$\begin{array}{rcrcrcl} x_1 & = & \frac{7}{8} & - & \frac{3}{4}x_2 & - & x_3 \\ x_4 & = & \frac{1}{8} & - & \frac{1}{4}x_2 & & \end{array}, \tag{6.6}$$

determining x_1 and x_4 from x_2 and x_3. It is important to consider what values these remaining $n - m$ variables can take in problems in the standard form. The objective function $c^\top x$ is linear and so contains no curvature which can given rise to a minimizing point. Hence such a point must be created by the conditions $x_i \geq 0$ becoming active on the boundary of the feasible region. For example, if (6.6) is used to eliminate the variables x_1 and x_4 from the problem (6.4), then the objective function can be expressed as:

$$c^\top x = x_1 + 2x_2 + 3x_3 + 4x_4 = \tfrac{11}{8} + \tfrac{1}{4}x_2 + 2x_3. \tag{6.7}$$

Clearly this function has no minimum value unless the conditions $x_2 \geq 0$ and $x_3 \geq 0$ are imposed, in which case the minimum occurs when $x_2 = x_3 = 0$.

To summarize, a solution of an LP problem in standard form always exists at one particular *extreme point* or *vertex* of the feasible region, with at least $n - m$ variables having *zero value*, and the remaining m variables being uniquely determined by the equations $Ax = b$, and taking non-negative values. This result is fundamental to the development of LP methods.

The main difficulty in LP is to find which $n - m$ variables take zero value at the solution and which m variables are not. The first algorithm to solve LP problems is the *simplex method*, which tries different sets of possibilities in a systematic way.

6.2.1 The Simplex Method

The simplex method for solving an LP problem in standard form generates a sequence of feasible points $x^{(1)}, x^{(2)}, \ldots$ which terminates at a solution. Since there exists a vertex at which the solution occurs, each iterate $x^{(k)}$ is also a vertex. Thus m variables have a non-negative value (usually positive) at $x^{(k)}$ and are referred to as *basic variables*, and the remaining $n - m$ variables have zero value and are referred to as *nonbasic variables*. In each iteration, the simplex method swaps a basic and nonbasic variable in order to decrease the objective function. The superscript (k) is often omitted for clarity.

In the simplex method, we start by partitioning $A = [A_B|A_N]$ where A_B is non-singular. Partitioning $x^\top = (x_B^\top, x_N^\top)$ conformably and $c^\top = (c_B^\top, c_N^\top)$, then we may write the LP problem (6.3) in *tableau form*:

$$\left[\begin{array}{c|c} c^\top & 0 \\ \hline A & b \end{array}\right] = \left[\begin{array}{cc|c} c_B^\top & c_N^\top & 0 \\ \hline A_B & A_N & b \end{array}\right].$$

Example 6.2.1. *The LP problem* (6.4) *can be reformulated as:*

$$\left[\begin{array}{cccc|c} 1 & 2 & 3 & 4 & 0 \\ \hline 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & -3 & \frac{1}{2} \end{array}\right].$$

The 2*-by-*2 *sub-block in the lower left is non-singular,* $\begin{vmatrix} 1 & 1 \\ 1 & 0 \end{vmatrix} = -1$*, and thus we start by identifying* x_1 *and* x_2 *as the basic variables.*

Then, by performing *row operations* on the tableau (adding or subtracting multiples of one row from another, or by scaling any row), it is possible to reduce the tableau to the form:

$$\left[\begin{array}{cc|c} c_B^\top & c_N^\top & 0 \\ \hline A_B & A_N & b \end{array}\right] \to \left[\begin{array}{cc|c} 0^\top & \hat{c}_N^\top & -\hat{f} \\ \hline I & \hat{A}_N & \hat{b} \end{array}\right],$$

where $\hat{A}_N = A_B^{-1} A_N$, which represents the *reduced form* of the LP problem.

Example 6.2.2. *Performing row operations on the LP problem* (6.4)*:*

$$\begin{aligned} r_1 \leftarrow r_1 - r_2 \\ r_3 \leftarrow r_2 - r_3 \end{aligned} \Rightarrow \left[\begin{array}{cccc|c} 0 & 1 & 2 & 3 & -1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 4 & \frac{1}{2} \end{array}\right],$$

$$\begin{aligned} r_1 \leftarrow r_1 - r_3 \\ r_2 \leftarrow r_2 - r_3 \end{aligned} \Rightarrow \left[\begin{array}{cccc|c} 0 & 0 & 2 & -1 & -\frac{3}{2} \\ \hline 1 & 0 & 1 & -3 & \frac{1}{2} \\ 0 & 1 & 0 & 4 & \frac{1}{2} \end{array}\right].$$

At this stage, the reduced LP problem is equivalent to:

$$\textit{minimize } \tfrac{3}{2} + 2x_3 - x_4, \tag{6.8a}$$

$$\textit{subject to } \begin{array}{ccccccc} x_1 & & + & x_3 & - & 3x_4 & = & \frac{1}{2} \\ & x_2 & & & + & 4x_4 & = & \frac{1}{2} \end{array}, \tag{6.8b}$$

$$\textit{and } x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0. \tag{6.8c}$$

While originally the variables x_3 and x_4 are the nonbasic variables, and thereby 0, generally they must be non-negative. Therefore, since the coefficient of x_4 is -1, a positive perturbation of x_4 (less than $1/8$ otherwise x_2 would fail its non-negativity constraint!) will decrease the objective.

The next step is to analyze the reduced costs $\hat{c}_N$. If the current vertex is optimal, then the reduced costs satisfy the test:

$$\hat{c}_N \geq 0. \tag{6.9}$$

If not, then we permute the nonbasic variable associated with the most negative cost:

$$j = \underset{1 \leq i \leq n-m}{\arg\min} [\hat{c}_N]_i,$$

for the basic variable which is forced to violate its non-negativity constraint first. Row operations are again performed to recover the reduced form of the LP problem. The simplex algorithm terminates when all the reduced costs in the final iteration satisfy the test (6.9).

Example 6.2.3. *In the LP problem* (6.4)*, -1 is the lowest element in the vector of reduced costs. Therefore, we introduce row operations to make a basis vector to then permute with the basic variable x_2:*

$$r_3 \leftarrow \tfrac{1}{4} r_3 \Rightarrow \left[\begin{array}{cccc|c} 0 & 0 & 2 & -1 & -\frac{3}{2} \\ \hline 1 & 0 & 1 & -3 & \frac{1}{2} \\ 0 & \frac{1}{4} & 0 & 1 & \frac{1}{8} \end{array}\right],$$

$$\begin{aligned} r_1 \leftarrow r_1 + r_3 \\ r_2 \leftarrow r_2 + 3r_3 \end{aligned} \Rightarrow \left[\begin{array}{cccc|c} 0 & \frac{1}{4} & 2 & 0 & -\frac{11}{8} \\ \hline 1 & \frac{3}{4} & 1 & 0 & \frac{7}{8} \\ 0 & \frac{1}{4} & 0 & 1 & \frac{1}{8} \end{array}\right],$$

$$c_2 \leftrightarrow c_4 \Rightarrow \left[\begin{array}{cccc|c} 0 & 0 & 2 & \frac{1}{4} & -\frac{11}{8} \\ \hline 1 & 0 & 1 & \frac{3}{4} & \frac{7}{8} \\ 0 & 1 & 0 & \frac{1}{4} & \frac{1}{8} \end{array}\right].$$

After one iteration, the reduced costs are all positive, satisfying the test (6.9)*, and therefore we know that the nonbasic variables are exactly 0. Therefore, the solution to the LP problem* (6.4) *is $\frac{11}{8}$, satisfied by:*

$$x_1 = \tfrac{7}{8},\ x_2 = 0,\ x_3 = 0,\ x_4 = \tfrac{1}{8}.$$

Remark 6.2.4. If more than one iteration of the simplex method is required, it may be difficult for you to keep track of the variables after multiple column permutations. Therefore, it is sufficient to ensure that columns of the identity matrix are present *somewhere* in the reduced form at each iteration. The simplex algorithm still terminates when all the reduced costs are positive.

Example 6.2.5.

$$\begin{aligned}
&\textit{minimize} && x_1 + 3x_2 + x_3 + 2x_4 + x_5\,, && && && && && && (6.10a)\\
& && x_1 + 2x_3 + 4x_5 &&=&& 3 && && \\
&\textit{subject to} && 2x_2 + 3x_4 &&=&& 1\,, && && (6.10b)\\
& && x_1 + x_2 + x_3 &&=&& 2 && && \\
&\textit{and } && x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0,\ x_5 \geq 0. && && && && && (6.10c)
\end{aligned}$$

In tableau form, the LP problem is:

$$\left[\begin{array}{ccccc|c} 1 & 3 & 1 & 2 & 1 & 0 \\ \hline 1 & 0 & 2 & 0 & 4 & 3 \\ 0 & 2 & 0 & 3 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 2 \end{array}\right].$$

We perform row operations to obtain a reduced form. Note that the lower left 3*-by-*3 *sub-block is non-singular.*

$$\begin{aligned}
r_3 \leftarrow \tfrac{1}{2} r_3 \quad &\Rightarrow \left[\begin{array}{ccccc|c} 1 & 3 & 1 & 2 & 1 & 0 \\ \hline 1 & 0 & 2 & 0 & 4 & 3 \\ 0 & 1 & 0 & \frac{3}{2} & 0 & \frac{1}{2} \\ 1 & 1 & 1 & 0 & 0 & 2 \end{array}\right] \\
r_1 \leftarrow r_1 - r_2 \quad &\Rightarrow \left[\begin{array}{ccccc|c} 0 & 3 & -1 & 2 & -3 & -3 \\ \hline 1 & 0 & 2 & 0 & 4 & 3 \\ 0 & 1 & 0 & \frac{3}{2} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & \frac{3}{2} & 4 & \frac{3}{2} \end{array}\right] \\
r_4 \leftarrow r_2 + r_3 - r_4 \quad & \\
r_1 \leftarrow r_1 - 3r_3 + r_4 \quad & \\
r_2 \leftarrow r_2 - 2r_4 \quad &\Rightarrow \left[\begin{array}{ccccc|c} 0 & 0 & 0 & \textcircled{-1} & 1 & -3 \\ \hline 1 & 0 & 0 & -3 & -4 & 0 \\ 0 & 1 & 0 & \boxed{\frac{3}{2}} & 0 & \frac{1}{2} \\ 0 & 0 & 1 & \frac{3}{2} & 4 & \frac{3}{2} \end{array}\right].
\end{aligned}$$

Now in reduced form, we find that the first reduced cost $[\hat{c}_B]_1 < 0$*, circled, so we will be permuting with the nonbasic variable* x_4*. The second row in the tableau shows that* $x_1 - 3x_4 + 4x_5 = 0$*. Therefore, increasing* x_4 *will not violate the non-negativity of* x_1*. The third row in the tableau shows that* $x_2 + \frac{3}{2}x_4 = \frac{1}{2}$*, implying that* x_2 *violates the non-negativity constraint when* $x_4 \geq \frac{1}{3}$*. On the other hand, the fourth row in the tableau shows that* $x_3 + \frac{3}{2}x_4 + 4x_5 = \frac{3}{2}$*, implying that* x_3 *violates the non-negativity constraint when* $x_4 \geq 1$*. Thus, by increasing the nonbasic variable* x_4*, the basic variable* x_2 *will violate the non-negativity*

constraint before x_3. Therefore, the pivot is boxed.

$$
\begin{array}{r}
\\
r_3 \leftarrow \frac{2}{3} r_3 \\
\\
\\
\\
r_1 \leftarrow r_1 + r_3 \\
r_2 \leftarrow r_2 + 3r_3 \\
\\
r_4 \leftarrow r_4 - \frac{3}{2} r_3
\end{array}
\Rightarrow
\left[\begin{array}{ccccc|c}
0 & 0 & 0 & -1 & 1 & -3 \\
\hline
1 & 0 & 0 & -3 & -4 & 0 \\
0 & \frac{2}{3} & 0 & 1 & 0 & \frac{1}{3} \\
0 & 0 & 1 & \frac{3}{2} & 4 & \frac{3}{2}
\end{array}\right],
\Rightarrow
\left[\begin{array}{ccccc|c}
0 & \frac{2}{3} & 0 & 0 & 1 & -\frac{8}{3} \\
\hline
1 & 2 & 0 & 0 & -4 & 1 \\
0 & \frac{2}{3} & 0 & 1 & 0 & \frac{1}{3} \\
0 & -1 & 1 & 0 & 4 & 1
\end{array}\right].
$$

We have returned the LP problem again to reduced form, but this time the reduced costs are all positive. The minimum of $\frac{8}{3}$ is attained at $x = (1, 0, 1, \frac{1}{3}, 0)^\top$.

It became apparent that the tableau form is inefficient in that it updates the entire tableau $\hat{A}_N$ at each iteration. In fact, all the operations in the simplex method can be carried out with an explicit knowledge of A_B^{-1}. Since A_B^{-1} is often smaller than $\hat{A}_N$, the resulting method, known as the *revised simplex method*, is usually more efficient. The effect of a basis change is to replace the column a_p by a_q in $A_B^{(k)}$, which can be written as the rank-one change:

$$A_B^{(k+1)} = A_B^{(k)} + (a_q - a_p) e_p^\top.$$

By using the Sherman–Morrison formula:

$$\left(A + uv^\top\right)^{-1} = A^{-1} - \frac{A^{-1} u v^\top A^{-1}}{1 + v^\top A^{-1} u},$$

we can update the inverse of $A_B^{(k+1)}$ directly.

For LP problems over hundreds of variables, the Sherman–Morrison formula leads to instabilities due to amplification of rounding errors. Instead, well-conditioned matrix factorizations, such as the QR factorization are employed to ensure stability. The rank-one update can similarly be effected in the Q and R factors. For LP problems over tens of thousands of variables, the sparsity of the matrix A is captured, and a pivoting strategy is used to preserve as much of the original sparsity in the *factors* as possible. This is an area of active research.

6.3 Problems

1. What and where is the global minimum of the function:

$$\begin{aligned} f(x) = e^{\sin(50x_1)} &+ \sin(60e^{x_2})\sin(60x_3) + \sin(70\sin(x_1))\cos(10x_3) \\ &+ \sin(\sin(80x_2)) - \sin(10(x_1 + x_3)) + (x_1^2 + x_2^2 + x_3^2)/4 \ ? \end{aligned}$$

2. Solve the linear program:

$$\text{minimize} \quad 3x_1 + 5x_2 + 4x_3 + 2x_4 + x_5 \ , \tag{6.11a}$$

$$\text{subject to} \quad \begin{array}{rcrcrcrcl} x_1 & & - & 2x_3 & & + & 4x_5 & = & 1 \\ -x_1 & + & 2x_2 & + & 5x_3 & + & 3x_4 & = & 2 \\ x_1 & & + & x_3 & + & 2x_4 & & = & 1 \end{array} , \tag{6.11b}$$

$$\text{and } x_1 \geq 0,\ x_2 \geq 0,\ x_3 \geq 0,\ x_4 \geq 0,\ x_5 \geq 0. \tag{6.11c}$$

Bibliography

[1] E. Süli and D. Mayers. *An Introduction to Numerical Analysis*. Cambridge University Press, 2003.

[2] W. Gautschi. *Numerical Analysis*. Birkhäuser, second edition, 2012.

[3] R. Fletcher. *Practical Methods of Optimization*, volume 1: Unconstrained Optimization. John Wiley & Sons, Ltd., 1980.

[4] R. Fletcher. *Practical Methods of Optimization*, volume 2: Constrained Optimization. John Wiley & Sons, Ltd., 1980.

[5] N. J. Higham. *Accuracy and Stability of Numerical Algorithms*. SIAM, second edition, 2002.

[6] W. G. Horner. A new method of solving numerical equations of all orders, by continuous approximation. *Phil. Trans. Roy. Soc.*, 109:308–335, 1819.

[7] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *J. Res. Nat. Bur. Standards*, 49:409–436, 1952.

[8] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis, Third Edition. Texts in Applied Mathematics 12*. Springer-Verlag, New York, 2002.

[9] N. J. Higham. The numerical stability of barycentric Lagrange interpolation. *IMA J. Numer. Anal.*, 24:547–556, 2004.

[10] C. Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Z. Math. Phys.*, 46:224–243, 1901.

[11] C. W. Clenshaw. A note on the summation of Chebyshev series. *Math. Comp.*, 9:118–120, 1955.

[12] F. J. Smith. An algorithm for summing orthogonal polynomial series and their derivatives with applications to curve-fitting and interpolation. *Math. Comp.*, 19:33–36, 1965.

[13] A. F. Nikiforov and V. B. Uvarov. *Special Functions of Mathematical Physics*. Birkhäuser Verlag, 1988.

[14] L. N. Trefethen. *Approximation Theory and Approximation Practice*. SIAM, 2012.

[15] L. F. Richardson. The approximate arithmetical solution by finite differences of physical problems including differential equations, with an application to the stresses in a masonry dam. *Phil. Trans. of the Royal Society A*, 210:459–470, 1911.

[16] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Math. Comp.*, 19:297–301, 1965.

[17] C. W. Clenshaw and A. R. Curtis. A method for numerical integration on an automatic computer. *Numer. Math.*, 2:197–205, 1960.

[18] F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel. *The SIAM 100-Digit Challenge. A Study in High-Accuracy Numerical Computing*. SIAM, Philadelphia, 2004.