# Numerical Analysis

An Undergraduate Course in the Numerical Solution of
Linear Systems, Differential Equations, and Nonlinear Equations

Weinan Wang

Department of Mathematics, University of Oklahoma

ww@ou.edu

April 29, 2024

# Contents

# Preface

These notes accompany a one-semester undergraduate course in numerical analysis: the design and analysis of algorithms that compute approximate solutions to mathematical problems, together with the quantitative study of the errors those algorithms incur. The four subjects treated are the solution of linear systems, the interpolation of data by polynomials, the numerical solution of ordinary differential equations, and the solution of nonlinear equations in one variable.

The prerequisites are the calculus sequence, including Taylor's theorem, and a first acquaintance with linear algebra and with ordinary differential equations. Algorithms are stated in pseudocode and implemented in MATLAB; the programs are written for clarity rather than for speed or robustness. Sections and results marked with a star ($\star$) are more advanced or technical and may be treated as optional. Each chapter ends with exercises, and solution sketches for selected exercises are collected at the end.

*Notation.* Scalars are written in lightface, vectors and matrices in boldface. The entries of a matrix $\mathbf{A}$ are $a_{ij}$ and the components of a vector $\mathbf{x}$ are $x_i$. The symbol $\mathbf{A}^{\mathsf{T}}$ denotes the transpose, $\mathbf{I}$ the identity matrix, and $\|\cdot\|$ a norm, with the particular norm indicated by a subscript. The notation $g(h) = \mathcal{O}(h^p)$ means $|g(h)| \le Ch^p$ for some constant $C$ and all sufficiently small $h > 0$.

# Chapter 1

# Matrices and Linear Algebra Review

The central computational problem of numerical linear algebra is the solution of a system of $n$ linear equations in $n$ unknowns, written in matrix form as $\mathbf{A}\mathbf{x} = \mathbf{b}$. Its efficient and accurate solution rests on the basic facts about matrices, determinants, and inverses, and on a consistent notation for them.

## 1.1 Matrices and their operations

A *matrix* of size $m \times n$ is a rectangular array $\mathbf{A} = (a_{ij})$ with $m$ rows and $n$ columns. Matrices of the same size are added, subtracted, and scaled entrywise. The *product* of an $m \times p$ matrix $\mathbf{A}$ and a $p \times n$ matrix $\mathbf{B}$ is the $m \times n$ matrix with entries

$$(\mathbf{AB})_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

Matrix multiplication is associative and distributes over addition, but it is not commutative: $\mathbf{AB} \neq \mathbf{BA}$ in general. The *transpose* $\mathbf{A}^{\mathsf{T}}$ has entries $(\mathbf{A}^{\mathsf{T}})_{ij} = a_{ji}$, and satisfies $(\mathbf{AB})^{\mathsf{T}} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}$. A column vector $\mathbf{x} \in \mathbb{R}^n$ is an $n \times 1$ matrix, and the matrix-vector product $\mathbf{A}\mathbf{x}$ has components $(\mathbf{A}\mathbf{x})_i = \sum_j a_{ij} x_j$.

## 1.2 Special matrices

Several structured matrices recur throughout. The $n \times n$ *identity* $\mathbf{I}$ has ones on the main diagonal and zeros elsewhere. A *diagonal* matrix $\operatorname{diag}(d_1, \ldots, d_n)$ is zero off the main diagonal. A matrix is *upper triangular* if $a_{ij} = 0$ whenever $i > j$, and *lower triangular* if $a_{ij} = 0$ whenever $i < j$; it is *unit* triangular if in addition its diagonal entries are all 1. A matrix is *symmetric*

if $\mathbf{A}^{\mathsf{T}} = \mathbf{A}$, and *tridiagonal* if its only nonzero entries lie on the main diagonal and the two adjacent diagonals. Triangular and tridiagonal structure make linear-system algorithms dramatically cheaper, since elimination need not touch the entries known in advance to be zero.

## 1.3 The inverse

A square matrix $\mathbf{A}$ is *nonsingular* (or invertible) if there is a matrix $\mathbf{A}^{-1}$ with $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$; otherwise it is *singular*. The inverse, when it exists, is unique, and

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}, \qquad (\mathbf{A}^{\mathsf{T}})^{-1} = (\mathbf{A}^{-1})^{\mathsf{T}}.$$

If $\mathbf{A}$ is nonsingular, the system $\mathbf{Ax} = \mathbf{b}$ has the unique solution $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. This formula is the right way to *think* about the solution but the wrong way to *compute* it: forming $\mathbf{A}^{-1}$ explicitly costs several times more arithmetic than solving the system directly, and is more sensitive to rounding error. The methods of Chapter 2 compute $\mathbf{x}$ without ever forming the inverse.

## 1.4 The determinant

To each square matrix is attached a scalar, the *determinant* $\det \mathbf{A}$, with the following properties. It is multiplicative, $\det(\mathbf{AB}) = \det \mathbf{A} \det \mathbf{B}$; unchanged by transposition, $\det \mathbf{A}^{\mathsf{T}} = \det \mathbf{A}$; equal to the product of the diagonal entries when $\mathbf{A}$ is triangular; and it responds to the elementary row operations in a definite way—interchanging two rows reverses its sign, scaling a row by $c$ multiplies it by $c$, and adding a multiple of one row to another leaves it unchanged. The determinant detects singularity: $\mathbf{A}$ is nonsingular if and only if $\det \mathbf{A} \neq 0$. Because reducing $\mathbf{A}$ to triangular form by row operations changes the determinant only in controlled ways, the elimination process of Chapter 2 also furnishes the value of $\det \mathbf{A}$ as a by-product.

## 1.5 Linear systems

The questions of when a linear system has a solution, and whether it is unique, are settled by nonsingularity.

**Theorem 1.1.** *For an $n \times n$ matrix $\mathbf{A}$ the following are equivalent:*

(i) $\mathbf{A}$ *is nonsingular;*

(ii) $\det \mathbf{A} \neq 0$*;*

(iii) *the homogeneous system* $\mathbf{Ax} = \mathbf{0}$ *has only the solution* $\mathbf{x} = \mathbf{0}$*;*

(iv) $\mathbf{Ax} = \mathbf{b}$ *has a unique solution for every* $\mathbf{b} \in \mathbb{R}^n$*.*

When $\mathbf{A}$ is singular the system $\mathbf{Ax} = \mathbf{b}$ has either no solution or infinitely many, depending on $\mathbf{b}$. The numerical methods that follow assume $\mathbf{A}$ is nonsingular and concentrate on computing the unique solution accurately and efficiently.

## 1.6 Cofactor expansion and Cramer's rule

The determinant can be computed recursively by expansion along a row or column. The *minor* $M_{ij}$ is the determinant of the $(n-1) \times (n-1)$ matrix obtained by deleting row $i$ and column $j$, and the *cofactor* is $C_{ij} = (-1)^{i+j} M_{ij}$. Expansion along row $i$ gives

$$\det \mathbf{A} = \sum_{j=1}^{n} a_{ij} C_{ij},$$

and likewise along any column. The cofactors also produce an explicit inverse, $\mathbf{A}^{-1} = \frac{1}{\det \mathbf{A}} \operatorname{adj} \mathbf{A}$, where the *adjugate* $\operatorname{adj} \mathbf{A} = (C_{ji})$ is the transpose of the matrix of cofactors, and they solve linear systems through *Cramer's rule*: if $\mathbf{A}$ is nonsingular, the solution of $\mathbf{Ax} = \mathbf{b}$ has components

$$x_i = \frac{\det \mathbf{A}_i}{\det \mathbf{A}},$$

where $\mathbf{A}_i$ is $\mathbf{A}$ with its $i$-th column replaced by $\mathbf{b}$.

**Example 1.2.** For $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 3 \end{smallmatrix}\right)\mathbf{x} = \left(\begin{smallmatrix} 3 \\ 5 \end{smallmatrix}\right)$ one has $\det \mathbf{A} = 5$, and Cramer's rule gives $x_1 = \det\left(\begin{smallmatrix} 3 & 1 \\ 5 & 3 \end{smallmatrix}\right)/5 = \frac{4}{5}$ and $x_2 = \det\left(\begin{smallmatrix} 2 & 3 \\ 1 & 5 \end{smallmatrix}\right)/5 = \frac{7}{5}$.

Elegant as they are, these formulas are computationally hopeless beyond the smallest sizes. Cofactor expansion of an $n \times n$ determinant invokes $n$ determinants of size $n-1$, so it costs on the order of $n!$ operations, and Cramer's rule needs $n+1$ such determinants. For $n = 20$ this is about $20! \approx 2.4 \times 10^{18}$ operations, beyond any computer, whereas Gaussian elimination solves the same system in roughly $\frac{2}{3}(20)^3 \approx 5000$ operations. The determinant and Cramer's rule are indispensable as theory and useless as computation; practical algorithms never use them.

## 1.7 Rank, positive definiteness, and orthogonality

The *rank* of a matrix is the number of linearly independent columns, which equals the number of linearly independent rows. An $n \times n$ matrix has rank $n$ exactly when it is nonsingular, so full rank, nonzero determinant, and invertibility coincide. Three structured classes of matrices recur throughout numerical analysis.

A symmetric matrix $\mathbf{A}$ is *positive definite* if $\mathbf{x}^{\mathsf{T}}\mathbf{A}\mathbf{x} > 0$ for every nonzero $\mathbf{x}$. Such matrices have strictly positive eigenvalues and positive leading principal minors, and they admit a Cholesky factorization $\mathbf{A} = \mathbf{L}\mathbf{L}^{\mathsf{T}}$ with $\mathbf{L}$ lower triangular. They arise as the coefficient matrices of least-squares problems and of discretized differential operators.

A square matrix $\mathbf{Q}$ is *orthogonal* if $\mathbf{Q}^{\mathsf{T}}\mathbf{Q} = \mathbf{I}$, so that $\mathbf{Q}^{-1} = \mathbf{Q}^{\mathsf{T}}$. Its columns are orthonormal, it preserves Euclidean length and inner products, $\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2$, and its determinant is $\pm 1$. The rotation $\left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix}\right)$ is the prototype. Because they neither stretch nor shrink vectors, orthogonal matrices are the building blocks of the most numerically stable algorithms.

A *permutation matrix* is obtained from the identity by permuting its rows. It is orthogonal, its inverse equals its transpose, and multiplying by it merely reorders the entries of a vector. Permutation matrices record the row interchanges performed during pivoting.

**Example 1.3.** The symmetric tridiagonal matrix $\mathbf{A} = \left(\begin{smallmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{smallmatrix}\right)$ is positive definite, as the leading principal minors are all positive: the $1\times 1$ minor is $2$, the $2\times 2$ minor is $\det\left(\begin{smallmatrix} 2 & -1 \\ -1 & 2 \end{smallmatrix}\right) = 3$, and $\det \mathbf{A} = 2 \cdot 3 - (-1)\cdot(-2) = 4$. This matrix, which arises from discretizing a second derivative, recurs throughout the subject as a model sparse system.

## Exercises

**Exercise 1.1.** Give $2 \times 2$ matrices $\mathbf{A}, \mathbf{B}$ with $\mathbf{A}\mathbf{B} \neq \mathbf{B}\mathbf{A}$, and verify that $(\mathbf{A}\mathbf{B})^{\mathsf{T}} = \mathbf{B}^{\mathsf{T}}\mathbf{A}^{\mathsf{T}}$ for your example.

**Exercise 1.2.** Show that the product of two upper triangular matrices is upper triangular, and that the diagonal entries of the product are the products of the corresponding diagonal entries.

**Exercise 1.3.** Prove that if $\mathbf{A}$ is nonsingular then so is $\mathbf{A}^{\mathsf{T}}$, and that $(\mathbf{A}^{\mathsf{T}})^{-1} = (\mathbf{A}^{-1})^{\mathsf{T}}$.

**Exercise 1.4.** Using the row-operation properties of the determinant, explain why a matrix with two equal rows has determinant zero.

**Exercise 1.5.** Let $\mathbf{A} = \operatorname{diag}(d_1, \dots, d_n)$. State when $\mathbf{A}$ is nonsingular and write down $\mathbf{A}^{-1}$ and $\det \mathbf{A}$.

**Exercise 1.6.** *Show that the inverse of a nonsingular lower triangular matrix is lower triangular.

# Chapter 2

# Gaussian Elimination and LU Factorization

## 2.1 Elimination and back substitution

Gaussian elimination solves $\mathbf{Ax} = \mathbf{b}$ by transforming it through elementary row operations into an equivalent system with an upper triangular coefficient matrix, which is then solved from the bottom up. At the first step the first equation is used to eliminate the unknown $x_1$ from the remaining equations: for each row $i > 1$ the *multiplier*

$$m_{i1} = \frac{a_{i1}}{a_{11}}$$

is formed and $m_{i1}$ times the first row is subtracted from row $i$, which sets the new $(i, 1)$ entry to zero. The entry $a_{11}$ used in the division is the first *pivot*. Repeating with the second equation to eliminate $x_2$ from rows $3, \ldots, n$, and so on, produces after $n - 1$ steps an upper triangular system $\mathbf{Ux} = \mathbf{c}$.

The triangular system is solved by *back substitution*: the last equation gives $x_n = c_n/u_{nn}$, and working upward,

$$x_i = \frac{1}{u_{ii}} \left( c_i - \sum_{j=i+1}^{n} u_{ij} x_j \right), \qquad i = n-1, \ldots, 1.$$

**Example 2.1.** Consider the system

$$\begin{aligned} 2x_1 + x_2 - x_3 &= 8, \\ -3x_1 - x_2 + 2x_3 &= -11, \\ -2x_1 + x_2 + 2x_3 &= -3. \end{aligned}$$

The first-stage multipliers are $m_{21} = -\frac{3}{2}$ and $m_{31} = -1$. Subtracting $m_{21}$ and $m_{31}$ times the first equation from the second and third gives

$$\tfrac{1}{2}x_2 + \tfrac{1}{2}x_3 = 1, \qquad 2x_2 + x_3 = 5.$$

The second-stage multiplier is $m_{32} = 2/\frac{1}{2} = 4$; eliminating $x_2$ leaves $-x_3 = 1$. Back substitution then yields $x_3 = -1$, $x_2 = 3$, $x_1 = 2$.

## 2.2 Operation count

The cost of a numerical method is measured by counting floating-point operations (additions, subtractions, multiplications, divisions), or *flops*. At stage $k$ of elimination there are $n-k$ rows to modify, each requiring about $2(n-k)$ flops, so the forward elimination costs

$$\sum_{k=1}^{n-1} 2(n-k)^2 = 2\sum_{j=1}^{n-1} j^2 \approx \frac{2}{3}n^3$$

flops to leading order. Back substitution costs about $n^2$ flops. Gaussian elimination is therefore an $\mathcal{O}(n^3)$ process, dominated by the elimination stage. Solving the same system by first computing $\mathbf{A}^{-1}$ and then forming $\mathbf{A}^{-1}\mathbf{b}$ costs several times as much, which is the quantitative reason inverses are avoided.

## 2.3 LU factorization

The multipliers and the final upper triangular matrix are not merely intermediate quantities: they constitute a factorization of $\mathbf{A}$.

**Theorem 2.2.** *If Gaussian elimination on $\mathbf{A}$ encounters no zero pivot, then $\mathbf{A} = \mathbf{L}\mathbf{U}$, where $\mathbf{U}$ is the upper triangular matrix produced by elimination and $\mathbf{L}$ is the unit lower triangular matrix whose below-diagonal entries are the multipliers, $\ell_{ik} = m_{ik}$. This factorization is unique.*

The factorization separates the work that depends only on $\mathbf{A}$ from the work that depends on $\mathbf{b}$. Once $\mathbf{A} = \mathbf{LU}$ is known, the system $\mathbf{Ax} = \mathbf{b}$, that is $\mathbf{LUx} = \mathbf{b}$, is solved in two triangular sweeps:

$$\text{solve } \mathbf{Ly} = \mathbf{b} \text{ (forward substitution)}, \qquad \text{solve } \mathbf{Ux} = \mathbf{y} \text{ (back substitution)}.$$

The factorization costs $\mathcal{O}(n^3)$ but each triangular solve costs only $\mathcal{O}(n^2)$, so a system that must be solved for many right-hand sides $\mathbf{b}$ is factored once and then solved repeatedly at the cheaper rate.

**Example 2.3.** For the matrix of Example 2.1,

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ -\frac{3}{2} & 1 & 0 \\ -1 & 4 & 1 \end{pmatrix}, \qquad \mathbf{U} = \begin{pmatrix} 2 & 1 & -1 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & -1 \end{pmatrix},$$

and one checks directly that $\mathbf{LU}$ reproduces $\mathbf{A}$.

## 2.4 Matrix inversion

When the inverse is genuinely required, it is computed by solving $\mathbf{AX} = \mathbf{I}$ one column at a time: the $j$-th column of $\mathbf{X} = \mathbf{A}^{-1}$ solves $\mathbf{Ax}_j = \mathbf{e}_j$, where $\mathbf{e}_j$ is the $j$-th column of $\mathbf{I}$. Using a single LU factorization, the $n$ columns are obtained by $n$ pairs of triangular solves, for a total cost of $\mathcal{O}(n^3)$. This confirms that the inverse is more expensive than a single solve and explains why it is formed only when the entries of $\mathbf{A}^{-1}$ are themselves wanted.

## 2.5 Algorithm and implementation

**Algorithm 2.1 (Gaussian elimination with back substitution).**

1. For $k = 1, \dots, n-1$ and each $i = k+1, \dots, n$: set $m_{ik} = a_{ik}/a_{kk}$, replace row $i$ of $\mathbf{A}$ by (row $i$) $- m_{ik}$ (row $k$) in columns $k$ through $n$, and set $b_i \leftarrow b_i - m_{ik} b_k$.
2. Set $x_n = b_n / a_{nn}$.
3. For $i = n-1, \dots, 1$: set $x_i = \big(b_i - \sum_{j=i+1}^{n} a_{ij} x_j\big)/a_{ii}$.

The following MATLAB function carries out this algorithm.

```
function x = gauss_elim(A, b)
% GAUSS_ELIM Solve A x = b by Gaussian elimination (no pivoting).
n = length(b);
for k = 1:n-1
    for i = k+1:n
        m = A(i,k) / A(k,k); % multiplier
        A(i,k:n) = A(i,k:n) - m*A(k,k:n);
        b(i) = b(i) - m*b(k);
    end
end
x = zeros(n,1); % back substitution
x(n) = b(n) / A(n,n);
for i = n-1:-1:1
    x(i) = (b(i) - A(i,i+1:n)*x(i+1:n)) / A(i,i);
end
end
```

The code divides by $a_{kk}$ at every stage, so it fails outright if a pivot is zero and loses accuracy if a pivot is small. Both defects are remedied by pivoting, the subject of Chapter 3.

## 2.6 Elementary matrices and the derivation of LU

The factorization $\mathbf{A} = \mathbf{L}\mathbf{U}$ can be derived directly by writing each elimination stage as a matrix multiplication. Eliminating below the pivot at stage $k$ subtracts multiples of row $k$ from the rows beneath it, which is left multiplication by the *elementary* matrix

$$\mathbf{M}_k = \mathbf{I} - \mathbf{m}_k \mathbf{e}_k^{\mathsf{T}}, \qquad \mathbf{m}_k = (0, \dots, 0, m_{k+1,k}, \dots, m_{n,k})^{\mathsf{T}},$$

where $\mathbf{e}_k$ is the $k$-th standard basis vector. After all stages,

$$\mathbf{M}_{n-1} \cdots \mathbf{M}_2 \mathbf{M}_1 \mathbf{A} = \mathbf{U}.$$

Each $\mathbf{M}_k$ is unit lower triangular, and because $\mathbf{e}_k^{\mathsf{T}} \mathbf{m}_k = 0$ its inverse is obtained simply by reversing the signs of the multipliers, $\mathbf{M}_k^{-1} = \mathbf{I} + \mathbf{m}_k \mathbf{e}_k^{\mathsf{T}}$. Therefore

$$\mathbf{A} = \mathbf{M}_1^{-1} \mathbf{M}_2^{-1} \cdots \mathbf{M}_{n-1}^{-1} \, \mathbf{U} = \mathbf{L}\mathbf{U},$$

and a short computation shows that this product simply places each multiplier in its natural position, so that $\mathbf{L}$ is unit lower triangular with $\ell_{ik} = m_{ik}$ for $i > k$. This proves Theorem 2.2 and explains why the multipliers may be stored, as they are computed, in the zeroed-out lower part of the working array: the array ends holding both $\mathbf{L}$ and $\mathbf{U}$.

## 2.7 Banded systems and the Thomas algorithm

Discretizations of differential equations, cubic spline interpolation, and many other applications produce *tridiagonal* systems, in which only the main diagonal and its two neighbors are nonzero:

$$b_1 x_1 + c_1 x_2 = d_1, \qquad a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i \;\; (1 < i < n), \qquad a_n x_{n-1} + b_n x_n = d_n.$$

Gaussian elimination specializes to the *Thomas algorithm*, which exploits the structure to run in $\mathcal{O}(n)$ operations and $\mathcal{O}(n)$ storage rather than the $\mathcal{O}(n^3)$ and $\mathcal{O}(n^2)$ of the general method. A forward sweep eliminates the subdiagonal,

$$c'_1 = \frac{c_1}{b_1}, \quad d'_1 = \frac{d_1}{b_1}, \qquad c'_i = \frac{c_i}{b_i - a_i c'_{i-1}}, \quad d'_i = \frac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} \quad (i \ge 2),$$

and back substitution recovers the solution,

$$x_n = d'_n, \qquad x_i = d'_i - c'_i \, x_{i+1} \quad (i = n-1, \dots, 1).$$

The algorithm is stable without pivoting whenever the matrix is diagonally dominant, $|b_i| \ge |a_i| + |c_i|$, as it is in the common applications. The MATLAB implementation operates on the three diagonals as vectors.

```
function x = thomas(a, b, c, d)
% THOMAS Solve a tridiagonal system with sub-, main-, and super-diagonals.
n = length(d);
cp = zeros(n,1); dp = zeros(n,1);
cp(1) = c(1)/b(1); dp(1) = d(1)/b(1);
for i = 2:n
    m = b(i) - a(i)*cp(i-1);
    cp(i) = c(i) / m;
    dp(i) = (d(i) - a(i)*dp(i-1)) / m;
end
x = zeros(n,1);
x(n) = dp(n);
for i = n-1:-1:1
    x(i) = dp(i) - cp(i)*x(i+1);
end
end
```

A general *banded* matrix, with $p$ nonzero diagonals on each side of the main diagonal, is handled the same way at a cost of $\mathcal{O}(p^2 n)$ operations: elimination never touches the zeros outside the band, so the work scales with the bandwidth rather than the dimension.

**Example 2.4.** The factorization of $\mathbf{A} = \left(\begin{smallmatrix} 2 & 1 & 1 \\ 4 & 3 & 3 \\ 8 & 7 & 9 \end{smallmatrix}\right)$ proceeds by elimination. Subtracting 2 times row 1 from row 2 and 4 times row 1 from row 3, then 3 times the new row 2 from row 3, leaves

$$\mathbf{U} = \begin{pmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 2 \end{pmatrix}, \qquad \mathbf{L} = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & 3 & 1 \end{pmatrix},$$

where the entries of $\mathbf{L}$ are the multipliers used. One checks $\mathbf{LU} = \mathbf{A}$. With the factorization in hand, solving $\mathbf{Ax} = \mathbf{b}$ for any $\mathbf{b}$ costs only a forward and a backward substitution, each $\mathcal{O}(n^2)$.

# Exercises

**Exercise 2.1.** Solve the system $x_1 + x_2 + x_3 = 6,\ 2x_1 + 3x_2 + x_3 = 11,\ x_1 - x_2 + 2x_3 = 5$ by Gaussian elimination, recording the multipliers at each step.

**Exercise 2.2.** Find the LU factorization of $\mathbf{A} = \left(\begin{smallmatrix} 4 & 3 \\ 6 & 3 \end{smallmatrix}\right)$ and use it to solve $\mathbf{Ax} = (10, 12)^{\mathsf{T}}$.

**Exercise 2.3.** Count the flops in the forward- and back-substitution solves of a triangular system, and confirm each is $\mathcal{O}(n^2)$.

**Exercise 2.4.** Explain why, given the LU factorization of $\mathbf{A}$, the determinant is $\det \mathbf{A} = u_{11} u_{22} \cdots u_{nn}$.

**Exercise 2.5.** Modify `gauss_elim` to return the matrices $\mathbf{L}$ and $\mathbf{U}$ in addition to the solution $\mathbf{x}$.

**Exercise 2.6.** $^{\star}$A tridiagonal system can be solved in $\mathcal{O}(n)$ flops by specializing the elimination to the three nonzero diagonals (the Thomas algorithm). Write out the recurrence for the multipliers and the back substitution in this case.

# Chapter 3

# Pivoting and Numerical Stability

## 3.1 Failure of naive elimination

Gaussian elimination as stated divides by the pivot $a_{kk}$ at stage $k$. If a pivot is zero the process breaks down even though the system may be perfectly solvable; if a pivot is merely small, the process completes but can return a badly inaccurate answer in floating-point arithmetic.

**Example 3.1.** Consider, for a small parameter $\varepsilon > 0$,

$$\varepsilon x_1 + x_2 = 1, \qquad x_1 + x_2 = 2,$$

whose exact solution is $x_1 = 1/(1-\varepsilon)$, $x_2 = (1-2\varepsilon)/(1-\varepsilon)$, both close to 1. Naive elimination uses the pivot $\varepsilon$ and the multiplier $m_{21} = 1/\varepsilon$, which is enormous when $\varepsilon$ is small. The second equation becomes $(1 - 1/\varepsilon)x_2 = 2 - 1/\varepsilon$. In exact arithmetic this is correct, but in finite precision the large term $1/\varepsilon$ swamps the modest terms 1 and 2: once $\varepsilon$ is below the machine precision the additions $1 - 1/\varepsilon$ and $2 - 1/\varepsilon$ both round to $-1/\varepsilon$, giving the computed value $x_2 \approx 1$ with the correction lost, and then $x_1 = (1 - x_2)/\varepsilon$ evaluates to 0 instead of 1. The computed solution is completely wrong.

## 3.2 Partial pivoting

The remedy is to avoid small pivots by reordering the equations. At stage $k$, *partial pivoting* selects as the pivot row the row $p \geq k$ for which $|a_{pk}|$ is largest, and interchanges rows $p$ and $k$ before eliminating. Because the pivot is then at least as large in magnitude as every entry below it, all multipliers satisfy $|m_{ik}| \leq 1$, which prevents the explosive growth seen in Example 3.1.

**Algorithm 3.1 (Gaussian elimination with partial pivoting).**

1. For $k = 1, \dots, n-1$: choose $p \geq k$ maximizing $|a_{pk}|$; if $a_{pk} = 0$ for all such $p$ the matrix is singular, so stop; otherwise interchange rows $p$ and $k$ (of $\mathbf{A}$ and of $\mathbf{b}$).
2. Eliminate as before: for $i = k+1, \dots, n$ set $m_{ik} = a_{ik}/a_{kk}$ and subtract $m_{ik}$ times row $k$ from row $i$.
3. Solve the resulting triangular system by back substitution.

Applying partial pivoting to Example 3.1 interchanges the two equations, because $|1| > |\varepsilon|$. The pivot is now 1, the multiplier is $\varepsilon$, and the elimination gives $(1-\varepsilon)x_2 = 1 - 2\varepsilon$, from which $x_2 \approx 1$ and $x_1 \approx 1$ are computed accurately.

## 3.3 Permutation matrices and the PA = LU factorization

The row interchanges of partial pivoting are recorded by a *permutation matrix* $\mathbf{P}$, obtained from the identity by the same interchanges. Elimination with partial pivoting factors the row-permuted matrix,

$$\mathbf{PA} = \mathbf{LU},$$

with $\mathbf{L}$ unit lower triangular (its multipliers all bounded by 1 in magnitude) and $\mathbf{U}$ upper triangular. The system $\mathbf{Ax} = \mathbf{b}$ is then solved by applying the same permutation to $\mathbf{b}$ and carrying out two triangular solves: solve $\mathbf{Ly} = \mathbf{Pb}$, then $\mathbf{Ux} = \mathbf{y}$.

## 3.4 Stability of elimination with partial pivoting

The accuracy of Gaussian elimination is governed by the size of the entries that appear during elimination, measured by the *growth factor*, the ratio of the largest entry occurring in any $\mathbf{U}$ at any stage to the largest entry of $\mathbf{A}$. Partial pivoting keeps every multiplier bounded by 1 and in practice keeps the growth factor modest, so that the computed solution is the exact solution of a system with a slightly perturbed matrix $\mathbf{A} + \Delta\mathbf{A}$, where $\Delta\mathbf{A}$ is small relative to $\mathbf{A}$. How small a perturbation of $\mathbf{A}$ and $\mathbf{b}$ translates into how large a change in $\mathbf{x}$ is a separate question, the *conditioning* of the system, which is measured with vector and matrix norms.

## 3.5 Floating-point arithmetic

Computers represent real numbers in a finite *floating-point* system. The IEEE 754 double-precision format stores a number in 64 bits as a sign, an 11-bit exponent, and a 52-bit fraction, encoding

$$x = \pm(1.f_1 f_2 \cdots f_{52})_2 \times 2^e,$$

a normalized binary fraction times a power of two. The 52 fraction bits provide roughly 16 significant decimal digits, and the exponent range gives representable magnitudes from about $10^{-308}$ to $10^{308}$. The spacing between 1 and the next larger representable number is the *machine epsilon* $\epsilon_{\text{mach}} = 2^{-52} \approx 2.22 \times 10^{-16}$, and rounding a real number to the nearest floating-point number incurs a relative error of at most the *unit roundoff* $u = \frac{1}{2}\epsilon_{\text{mach}}$:

$$\text{fl}(x) = x(1+\delta), \qquad |\delta| \le u.$$

Each elementary operation obeys the same model, $\text{fl}(a \circ b) = (a \circ b)(1+\delta)$ with $|\delta| \le u$. Numbers too large to represent produce *overflow* (the value `Inf`), numbers too small *underflow* toward zero, and undefined results such as $0/0$ produce `NaN`. A consequence worth remembering is that most decimal fractions, 0.1 among them, are not representable exactly, so even simple decimal computations carry rounding error from the very first step.

## 3.6 Scaled pivoting, backward error, and iterative refinement

Partial pivoting compares the magnitudes of candidate pivots directly, which can mislead when the rows of $\mathbf{A}$ are scaled very differently. *Scaled partial pivoting* instead compares the relative sizes $|a_{ik}| / s_i$, where $s_i = \max_j |a_{ij}|$ is the largest magnitude in row $i$, choosing the pivot that is largest relative to its own row. *Complete pivoting* searches the entire remaining submatrix for the largest entry and interchanges both rows and columns; it yields the smallest growth factor but requires $\mathcal{O}(n^3)$ comparisons, and in practice partial or scaled pivoting is adequate.

The accuracy of a computed solution $\hat{\mathbf{x}}$ is described by two distinct quantities. The *backward error* asks what problem $\hat{\mathbf{x}}$ solves exactly: Gaussian elimination with partial pivoting produces an $\hat{\mathbf{x}}$ that solves $(\mathbf{A} + \Delta\mathbf{A})\hat{\mathbf{x}} = \mathbf{b}$ for some small $\Delta\mathbf{A}$, a property called *backward stability*. The size of the backward error is revealed by the *residual* $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$: a small residual means $\hat{\mathbf{x}}$ solves a nearby system. A small residual does not by itself guarantee a small *forward error* $\hat{\mathbf{x}} - \mathbf{x}$; the two are linked by the conditioning of $\mathbf{A}$.

When higher accuracy is needed, *iterative refinement* corrects a computed solution using its residual.

**Algorithm 3.2 (Iterative refinement).**

1. Solve $\mathbf{A}\hat{\mathbf{x}} = \mathbf{b}$ by LU factorization with partial pivoting.
2. Compute the residual $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$, using higher precision for the products where possible.
3. Solve $\mathbf{A}\boldsymbol{\delta} = \mathbf{r}$, reusing the existing LU factors.
4. Update $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}} + \boldsymbol{\delta}$ and repeat from step 2 until the correction is negligible.

Because the factorization is reused, each refinement step costs only $\mathcal{O}(n^2)$, and when the residual is computed accurately the procedure recovers digits lost to rounding during the original solve.

**Example 3.2.** The system $\left(\begin{smallmatrix} 0.0001 & 1 \\ 1 & 1 \end{smallmatrix}\right)\mathbf{x} = \left(\begin{smallmatrix} 1 \\ 2 \end{smallmatrix}\right)$ has the well-conditioned solution $\mathbf{x} \approx (1, 1)$. Solved in three-digit arithmetic *without* pivoting, the multiplier $1/0.0001 = 10000$ produces $y = 1.00$ but then $0.0001\,x + 1.00 = 1$ forces $x = 0$—a completely wrong first component, lost to the subtraction of a large multiple of row 1. Exchanging the rows first, so the pivot is 1, gives the multiplier 0.0001 and the correct result $\mathbf{x} = (1.00, 1.00)$. The matrix was never ill-conditioned; the small pivot alone caused the failure, and partial pivoting removes it.

# Exercises

**Exercise 3.1.** Solve $\varepsilon x_1 + x_2 = 1,\ x_1 + x_2 = 2$ with $\varepsilon = 10^{-3}$ both by naive elimination and by partial pivoting, carrying four significant digits throughout, and compare the computed solutions with the exact one.

**Exercise 3.2.** Write down the permutation matrix $\mathbf{P}$ corresponding to interchanging rows 1 and 3 of a $3 \times 3$ matrix, and verify that $\mathbf{P}^{-1} = \mathbf{P}^{\mathsf{T}}$.

**Exercise 3.3.** Show that for any matrix produced by partial pivoting every multiplier satisfies $|m_{ik}| \le 1$, and explain why this need not hold without pivoting.

**Exercise 3.4.** Give an example of a nonsingular $2 \times 2$ matrix for which naive elimination fails at the first step but partial pivoting succeeds.

**Exercise 3.5.** Modify `gauss_elim` to perform partial pivoting, interchanging rows of both `A` and `b` at each stage.

**Exercise 3.6.** ⋆Construct a $3 \times 3$ system in which two different pivot choices lead to computed solutions of differing accuracy, and identify which choice partial pivoting makes.

# Chapter 4

# Vector and Matrix Norms

To quantify the size of an error in a vector or a matrix, and to measure the sensitivity of a linear system to perturbations, one needs a notion of magnitude. That notion is a norm. Vector norms, matrix norms, and the spectral radius measure the size of vectors and matrices and underlie the error analysis of linear systems.

## 4.1 Vector norms

**Definition 4.1.** A *norm* on $\mathbb{R}^n$ is a function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ such that for all vectors $\mathbf{x}, \mathbf{y}$ and scalars $\alpha$:

(i) $\|\mathbf{x}\| \geq 0$, with equality if and only if $\mathbf{x} = \mathbf{0}$;

(ii) $\|\alpha\mathbf{x}\| = |\alpha| \ \|\mathbf{x}\|$;

(iii) $\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\|$.

The norms used in practice are the *p-norms*

$$\|\mathbf{x}\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p} \quad (1 \leq p < \infty), \qquad \|\mathbf{x}\|_\infty = \max_{1 \leq i \leq n} |x_i| \, .$$

The three cases that occur constantly are $p = 1$, the sum of absolute values; $p = 2$, the Euclidean length; and $p = \infty$, the largest component. Their unit balls $\{\mathbf{x} : \|\mathbf{x}\| \leq 1\}$ in the plane are, respectively, a diamond, a disk, and a square, a picture that makes the inequalities below geometrically plain.

## 4.2 Equivalence of norms

On $\mathbb{R}^n$ all norms are *equivalent*: for any two norms $\|\cdot\|_a$ and $\|\cdot\|_b$ there are positive constants $c$ and $C$ with $c\,\|\mathbf{x}\|_a \le \|\mathbf{x}\|_b \le C\,\|\mathbf{x}\|_a$ for every $\mathbf{x}$. For the standard norms the sharp constants are

$$\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \sqrt{n}\,\|\mathbf{x}\|_\infty\,, \qquad \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1 \le \sqrt{n}\,\|\mathbf{x}\|_2\,.$$

Equivalence means that a vector small in one norm is small in every other, so for questions of convergence the choice of norm is a matter of convenience rather than substance. The dimension $n$ does enter the constants, however, which can matter when $n$ is large.

## 4.3 Inner products and the Euclidean norm

The Euclidean norm arises from the *inner product* $\langle \mathbf{x}, \mathbf{y}\rangle = \mathbf{x}^{\mathsf{T}}\mathbf{y} = \sum_i x_i y_i$ through $\|\mathbf{x}\|_2 = \sqrt{\langle \mathbf{x}, \mathbf{x}\rangle}$. The inner product obeys the *Cauchy–Schwarz inequality*

$$|\langle \mathbf{x}, \mathbf{y}\rangle| \le \|\mathbf{x}\|_2\;\|\mathbf{y}\|_2\,,$$

from which the triangle inequality for $\|\cdot\|_2$ follows by expanding $\|\mathbf{x}+\mathbf{y}\|_2^2 = \|\mathbf{x}\|_2^2 + 2\,\langle \mathbf{x}, \mathbf{y}\rangle + \|\mathbf{y}\|_2^2$ and bounding the middle term. The angle $\theta$ between two nonzero vectors is defined by $\cos\theta = \langle \mathbf{x}, \mathbf{y}\rangle \,/ (\|\mathbf{x}\|_2\,\|\mathbf{y}\|_2)$, so orthogonality $\langle \mathbf{x}, \mathbf{y}\rangle = 0$ is the case $\theta = \pi/2$.

## 4.4 Matrix norms and induced norms

A measure of the size of a matrix should also respect multiplication. A *matrix norm* is a norm on $n \times n$ matrices that is *submultiplicative*, $\|\mathbf{AB}\| \le \|\mathbf{A}\|\;\|\mathbf{B}\|$. The *Frobenius norm* $\|\mathbf{A}\|_F = \big(\sum_{i,j} a_{ij}^2\big)^{1/2}$, the Euclidean length of the matrix regarded as a vector of its entries, is one such norm. The most useful matrix norms are those *induced* by a vector norm,

$$\|\mathbf{A}\| = \max_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|} = \max_{\|\mathbf{x}\|=1} \|\mathbf{A}\mathbf{x}\|\,,$$

which measures the greatest factor by which $\mathbf{A}$ can lengthen a vector. An induced norm automatically satisfies $\|\mathbf{A}\mathbf{x}\| \le \|\mathbf{A}\|\;\|\mathbf{x}\|$, is submultiplicative, and gives $\|\mathbf{I}\| = 1$.

**Theorem 4.2.** *The matrix norms induced by the vector* 1*-,* $\infty$*-, and* 2*-norms are*

$$\|\mathbf{A}\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}| \quad \textit{(largest column sum)}, \qquad \|\mathbf{A}\|_\infty = \max_i \sum_{j=1}^{n} |a_{ij}| \quad \textit{(largest row sum)},$$

$$\|\mathbf{A}\|_2 = \sqrt{\lambda_{\max}(\mathbf{A}^{\mathsf{T}}\mathbf{A})} \quad \textit{(largest singular value).}$$

*Proof for the $\infty$-norm.* For $\|\mathbf{x}\|_\infty = 1$ every $|x_j| \le 1$, so

$$|(\mathbf{A}\mathbf{x})_i| = \left|\sum_j a_{ij}x_j\right| \le \sum_j |a_{ij}|\,,$$

whence $\|\mathbf{A}\mathbf{x}\|_\infty \le \max_i \sum_j |a_{ij}|$. The bound is attained by choosing $x_j = \operatorname{sgn}(a_{Ij})$ for the row $I$ achieving the maximum, which makes $(\mathbf{A}\mathbf{x})_I = \sum_j |a_{Ij}|$. The 1-norm result is proved the same way with the roles of rows and columns exchanged, and the 2-norm formula follows from the spectral decomposition of the symmetric positive semidefinite matrix $\mathbf{A}^{\mathsf{T}}\mathbf{A}$. $\square$

The 1- and $\infty$-norms are read off a matrix by inspection, while the 2-norm requires an eigenvalue computation, which is why error bounds are often stated in the $\infty$-norm even when the 2-norm is the quantity of ultimate interest.

## 4.5 The spectral radius

The *spectral radius* of $\mathbf{A}$ is the largest magnitude among its eigenvalues,

$$\rho(\mathbf{A}) = \max_i |\lambda_i|\,.$$

It is bounded by every induced norm: if $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$, then $|\lambda|\,\|\mathbf{v}\| = \|\mathbf{A}\mathbf{v}\| \le \|\mathbf{A}\|\,\|\mathbf{v}\|$, so $|\lambda| \le \|\mathbf{A}\|$ and therefore

$$\rho(\mathbf{A}) \le \|\mathbf{A}\|\,.$$

When $\mathbf{A}$ is symmetric, equality holds in the 2-norm, $\|\mathbf{A}\|_2 = \rho(\mathbf{A})$, because the eigenvalues of $\mathbf{A}^{\mathsf{T}}\mathbf{A} = \mathbf{A}^2$ are the squares of those of $\mathbf{A}$. The spectral radius governs the convergence of iterative processes: an iteration whose error vector is multiplied by $\mathbf{A}$ at each step decays to zero precisely when $\rho(\mathbf{A}) < 1$.

**Example 4.3.** For $\mathbf{A} = \left(\begin{smallmatrix} 3 & 1 \\ 0 & -2 \end{smallmatrix}\right)$ the column sums are 3 and 3, so $\|\mathbf{A}\|_1 = 3$; the row sums are 4 and 2, so $\|\mathbf{A}\|_\infty = 4$; and the eigenvalues are 3 and $-2$, so $\rho(\mathbf{A}) = 3$. The inequality $\rho(\mathbf{A}) \le \|\mathbf{A}\|$ holds for each induced norm, with equality for neither because $\mathbf{A}$ is not symmetric.

**Example 4.4.** For $\mathbf{A} = \left(\begin{smallmatrix} 1 & -2 \\ 3 & 4 \end{smallmatrix}\right)$ the four common norms are

| | |
|---|---|
| $\|\mathbf{A}\|_1 = \max(4, 6) = 6$ | (largest absolute column sum) |
| $\|\mathbf{A}\|_\infty = \max(3, 7) = 7$ | (largest absolute row sum) |
| $\|\mathbf{A}\|_F = \sqrt{1+4+9+16} = \sqrt{30} \approx 5.48$ | (Frobenius) |
| $\|\mathbf{A}\|_2 = \sqrt{26.18} \approx 5.12$ | (largest singular value) |

The 2-norm comes from the eigenvalues 26.18 and 3.82 of $\mathbf{A}^{\mathsf{T}}\mathbf{A} = \left(\begin{smallmatrix} 10 & 10 \\ 10 & 20 \end{smallmatrix}\right)$; its square root is the largest singular value. The norms differ but all lie within the equivalence constants of one another.

# Exercises

**Exercise 4.1.** For $\mathbf{x} = (3, -4, 12)^{\mathsf{T}}$ compute $\|\mathbf{x}\|_1$, $\|\mathbf{x}\|_2$, and $\|\mathbf{x}\|_\infty$, and verify the inequalities $\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2 \le \|\mathbf{x}\|_1$.

**Exercise 4.2.** Prove that $\|\mathbf{x}\|_\infty \le \|\mathbf{x}\|_2$ and $\|\mathbf{x}\|_2 \le \sqrt{n}\,\|\mathbf{x}\|_\infty$ for every $\mathbf{x} \in \mathbb{R}^n$, and exhibit vectors attaining equality in each.

**Exercise 4.3.** For $\mathbf{A} = \left(\begin{smallmatrix} 1 & -2 \\ 3 & 4 \end{smallmatrix}\right)$ compute $\|\mathbf{A}\|_1$ and $\|\mathbf{A}\|_\infty$, and the Frobenius norm $\|\mathbf{A}\|_F$.

**Exercise 4.4.** Show that any induced matrix norm satisfies $\|\mathbf{I}\| = 1$, and deduce that the Frobenius norm is not induced by any vector norm when $n > 1$.

**Exercise 4.5.** Prove that $\rho(\mathbf{A}) \le \|\mathbf{A}\|$ for every induced norm, and give a nonsymmetric $2 \times 2$ matrix for which the inequality is strict in the 2-norm.

**Exercise 4.6.** ⋆Show that for a symmetric matrix $\mathbf{A}$ one has $\|\mathbf{A}\|_2 = \rho(\mathbf{A})$, using the existence of an orthonormal basis of eigenvectors.

# Chapter 5

# Eigenvalues and Eigenvectors

The eigenvalues of a matrix control the growth or decay of the iterative and time-stepping processes built from it, and they enter the conditioning analysis through the spectral radius and the 2-norm.

## 5.1 The eigenvalue problem

**Definition 5.1.** A nonzero vector $\mathbf{v}$ is an *eigenvector* of the $n \times n$ matrix $\mathbf{A}$ with *eigenvalue* $\lambda$ if $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$.

The equation $(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$ has a nonzero solution exactly when $\mathbf{A} - \lambda\mathbf{I}$ is singular, so the eigenvalues are the roots of the *characteristic polynomial*

$$p(\lambda) = \det(\mathbf{A} - \lambda\mathbf{I}),$$

a polynomial of degree $n$. Counted with multiplicity it has $n$ roots in $\mathbb{C}$, so an $n \times n$ matrix has $n$ eigenvalues, possibly complex and possibly repeated. For each eigenvalue the eigenvectors are the nonzero solutions of the singular system $(\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = \mathbf{0}$.

**Example 5.2.** For $\mathbf{A} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$, $p(\lambda) = (2-\lambda)^2 - 1 = (\lambda - 3)(\lambda - 1)$, so the eigenvalues are 3 and 1, with eigenvectors $(1, 1)^{\mathsf{T}}$ and $(1, -1)^{\mathsf{T}}$.

## 5.2 Properties of eigenvalues

Expanding the characteristic polynomial links the eigenvalues to two scalars already in use: the sum of the eigenvalues equals the trace and their product equals the determinant,

$$\sum_i \lambda_i = \operatorname{tr} \mathbf{A}, \qquad \prod_i \lambda_i = \det \mathbf{A}.$$

In particular $\mathbf{A}$ is singular if and only if 0 is an eigenvalue. Eigenvalues are invariant under similarity: if $\mathbf{B} = \mathbf{S}^{-1}\mathbf{A}\mathbf{S}$, then $\mathbf{A}$ and $\mathbf{B}$ share a characteristic polynomial and hence the same eigenvalues. The eigenvalues of a triangular matrix are exactly its diagonal entries.

## 5.3 Diagonalization

If $\mathbf{A}$ has $n$ linearly independent eigenvectors $\mathbf{v}_1, \dots, \mathbf{v}_n$, they form the columns of an invertible matrix $\mathbf{S}$ satisfying $\mathbf{A}\mathbf{S} = \mathbf{S}\mathbf{\Lambda}$ with $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$, so that $\mathbf{A} = \mathbf{S}\mathbf{\Lambda}\mathbf{S}^{-1}$. A matrix with $n$ distinct eigenvalues is always diagonalizable. Diagonalization makes powers transparent, $\mathbf{A}^k = \mathbf{S}\mathbf{\Lambda}^k\mathbf{S}^{-1}$, so the growth of $\mathbf{A}^k$ is governed by the powers $\lambda_i^k$ of the eigenvalues, which determines whether an iteration $\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k$ grows or decays.

## 5.4 Symmetric matrices and the spectral theorem

Symmetric matrices have the most favorable eigenstructure possible.

**Theorem 5.3** (Spectral theorem)**.** *A real symmetric matrix has only real eigenvalues, and its eigenvectors can be chosen to form an orthonormal basis of $\mathbb{R}^n$; equivalently $\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{\mathsf{T}}$ with $\mathbf{Q}$ orthogonal and $\mathbf{\Lambda}$ real diagonal.*

Two parts of this are short computations. If $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$, the scalar $\lambda \|\mathbf{v}\|_2^2 = \mathbf{v}^{\mathsf{T}}\mathbf{A}\mathbf{v}$ is real because the quadratic form of a symmetric matrix is real, so $\lambda$ is real. If also $\mathbf{A}\mathbf{u} = \mu\mathbf{u}$ with $\mu \neq \lambda$, then $\lambda\,\mathbf{u}^{\mathsf{T}}\mathbf{v} = \mathbf{u}^{\mathsf{T}}\mathbf{A}\mathbf{v} = \mu\,\mathbf{u}^{\mathsf{T}}\mathbf{v}$ forces $\mathbf{u}^{\mathsf{T}}\mathbf{v} = 0$, so eigenvectors for distinct eigenvalues are orthogonal. For a symmetric matrix the 2-norm equals the spectral radius, $\|\mathbf{A}\|_2 = \rho(\mathbf{A})$, and the condition number reduces to $|\lambda_{\max}| \,/\, |\lambda_{\min}|$.

## 5.5 Gershgorin's circle theorem

Computing eigenvalues exactly requires finding polynomial roots, but their location can be read directly from the entries.

**Theorem 5.4** (Gershgorin)**.** *Every eigenvalue of* $\mathbf{A}$ *lies in at least one of the disks in the complex plane centered at* $a_{ii}$ *with radius equal to the sum of the off-diagonal magnitudes in that row,*

$$|\lambda - a_{ii}| \le \sum_{j \ne i} |a_{ij}| \,.$$

*Proof.* Let $\mathbf{A}\mathbf{v} = \lambda \mathbf{v}$ and let $i$ be an index for which $|v_i|$ is largest. The $i$-th equation reads $(\lambda - a_{ii})v_i = \sum_{j \ne i} a_{ij} v_j$; dividing by the nonzero $v_i$ and taking absolute values, $|\lambda - a_{ii}| \le \sum_{j \ne i} |a_{ij}| \, |v_j / v_i| \le \sum_{j \ne i} |a_{ij}|$. □

A strictly diagonally dominant matrix, one with $|a_{ii}| > \sum_{j \ne i} |a_{ij}|$ for every $i$, has no Gershgorin disk reaching the origin, so 0 is not an eigenvalue and the matrix is nonsingular—a fact used to guarantee solvability without computing a determinant.

## 5.6 The power method

The simplest numerical eigenvalue method computes the *dominant* eigenvalue, the one of largest magnitude. Suppose $\mathbf{A}$ is diagonalizable with eigenvalues ordered $|\lambda_1| > |\lambda_2| \ge \cdots \ge |\lambda_n|$, and expand a starting vector in the eigenbasis as $\mathbf{x}_0 = \sum_i c_i \mathbf{v}_i$ with $c_1 \ne 0$. Then

$$\mathbf{A}^k \mathbf{x}_0 = \sum_i c_i \lambda_i^k \mathbf{v}_i = \lambda_1^k \Big( c_1 \mathbf{v}_1 + \sum_{i \ge 2} c_i \big(\lambda_i / \lambda_1\big)^k \mathbf{v}_i \Big).$$

Because $|\lambda_i / \lambda_1| < 1$ for $i \ge 2$, the sum in parentheses tends to $c_1 \mathbf{v}_1$, so repeated multiplication by $\mathbf{A}$ drives any vector toward the dominant eigenvector. Normalizing each iterate keeps it bounded, and the scaling factor recovers $\lambda_1$.

**Algorithm 5.1 (Power method).**

1. Choose $\mathbf{x}_0$ with $\|\mathbf{x}_0\|_\infty = 1$.
2. For $k = 0, 1, 2, \ldots$: set $\mathbf{y} = \mathbf{A}\mathbf{x}_k$, let $\mu_k$ be the entry of $\mathbf{y}$ of largest magnitude, and set $\mathbf{x}_{k+1} = \mathbf{y} / \mu_k$.
3. The values $\mu_k$ converge to $\lambda_1$ and the vectors $\mathbf{x}_k$ to a dominant eigenvector.

The error decreases by the factor $|\lambda_2 / \lambda_1|$ per step, so convergence is linear and is rapid only when the dominant eigenvalue is well separated from the rest.

**Example 5.5.** For $\mathbf{A} = \left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$, with dominant eigenvalue 3 and eigenvector $(1, 1)^{\mathsf{T}}$, the power method from $\mathbf{x}_0 = (1, 0)^{\mathsf{T}}$ produces the scaling factors

| $k$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| $\mu_k$ | 2.000 | 2.500 | 2.800 | 2.929 | 2.976 |

converging to 3, while the normalized iterates approach $(1,1)^{\mathsf{T}}$. The error decreases by the factor $|\lambda_2/\lambda_1| = 1/3$ per step, the linear rate predicted by the separation of the eigenvalues.

```
function [lambda, x] = power_method(A, x, tol, maxit)
% POWER_METHOD Dominant eigenvalue and eigenvector by power iteration.
x = x / norm(x, inf);
lambda = 0;
for k = 1:maxit
    y = A*x;
    [~, j] = max(abs(y)); % index of largest-magnitude entry
    mu = y(j);
    x = y / mu;
    if abs(mu - lambda) < tol
        lambda = mu; return;
    end
    lambda = mu;
end
end
```

# Exercises

**Exercise 5.1.** Find the eigenvalues and eigenvectors of $\left(\begin{smallmatrix} 3 & -1 \\ -1 & 3 \end{smallmatrix}\right)$, and verify that their sum is the trace and their product the determinant.

**Exercise 5.2.** Show that the eigenvalues of a triangular matrix are its diagonal entries.

**Exercise 5.3.** Use Gershgorin's theorem to bound the eigenvalues of $\left(\begin{smallmatrix} 5 & 1 & 0 \\ 1 & 6 & 1 \\ 0 & 1 & 7 \end{smallmatrix}\right)$, and conclude that the matrix is nonsingular.

**Exercise 5.4.** For a symmetric matrix, prove that eigenvectors belonging to distinct eigenvalues are orthogonal.

**Exercise 5.5.** Carry out two steps of the power method on $\left(\begin{smallmatrix} 2 & 1 \\ 1 & 2 \end{smallmatrix}\right)$ starting from $\mathbf{x}_0 = (1,0)^{\mathsf{T}}$, and compare the estimate $\mu_1$ with the dominant eigenvalue 3.

**Exercise 5.6.** ⋆Explain how the power method fails when the two largest eigenvalues have equal magnitude but opposite sign, and describe the behavior of the iterates in that case.

# Chapter 6

# Conditioning and Error Analysis

The norms of Chapter 4 make it possible to answer the central practical question about a linear system: when the data $\mathbf{A}$ and $\mathbf{b}$ are known only approximately, or are perturbed by rounding, how large can the change in the solution be? The answer is governed by a single number, the condition number.

## 6.1 Absolute, relative, forward, and backward error

For an approximation $\hat{x}$ to a number $x$ the *absolute error* is $|x - \hat{x}|$ and the *relative error* is $|x - \hat{x}| \,/\, |x|$; for vectors the absolute value is replaced by a norm. Relative error is the more meaningful measure: it is scale-independent and counts significant digits, since a relative error near $10^{-t}$ corresponds to about $t$ correct significant digits. For a linear system the *forward error* $\|\mathbf{x} - \hat{\mathbf{x}}\| \,/\, \|\mathbf{x}\|$ is the quantity of ultimate interest, while the *backward error* measures how nearly $\hat{\mathbf{x}}$ solves the original problem, as recorded by the residual $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$.

## 6.2 The condition number

**Definition 6.1.** The *condition number* of a nonsingular matrix $\mathbf{A}$, relative to a matrix norm, is $\operatorname{cond}(\mathbf{A}) = \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|$.

Since $1 = \|\mathbf{I}\| = \|\mathbf{A}\mathbf{A}^{-1}\| \le \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\|$, the condition number is always at least 1, with $\operatorname{cond}(\mathbf{I}) = 1$, and it is unchanged by scaling, $\operatorname{cond}(c\mathbf{A}) = \operatorname{cond}(\mathbf{A})$. In the 2-norm it is the ratio $\sigma_{\max}/\sigma_{\min}$ of the largest to the smallest singular value, and for a symmetric matrix this is $|\lambda_{\max}| \,/\, |\lambda_{\min}|$. A matrix with a large condition number is *ill-conditioned*, one with a condition number near 1 *well-conditioned*.

## 6.3 Perturbation of the right-hand side

**Theorem 6.2.** *If* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *and* $\mathbf{A}(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b} + \delta\mathbf{b}$ *with* $\mathbf{b} \neq \mathbf{0}$*, then*

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \operatorname{cond}(\mathbf{A}) \, \frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

*Proof.* Subtracting the two systems gives $\mathbf{A}\,\delta\mathbf{x} = \delta\mathbf{b}$, so $\delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b}$ and $\|\delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \, \|\delta\mathbf{b}\|$. From $\mathbf{b} = \mathbf{A}\mathbf{x}$ comes $\|\mathbf{b}\| \leq \|\mathbf{A}\| \, \|\mathbf{x}\|$, that is $1/\|\mathbf{x}\| \leq \|\mathbf{A}\| \, / \|\mathbf{b}\|$. Multiplying the two inequalities gives the bound. □

The relative error in the data is amplified by at most $\operatorname{cond}(\mathbf{A})$ in the solution: the condition number is the worst-case magnification factor.

## 6.4 Perturbation of the matrix

**Theorem 6.3.** *If* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *and* $(\mathbf{A} + \delta\mathbf{A})(\mathbf{x} + \delta\mathbf{x}) = \mathbf{b}$*, then*

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x} + \delta\mathbf{x}\|} \leq \operatorname{cond}(\mathbf{A}) \, \frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}.$$

*Proof.* Expanding and using $\mathbf{A}\mathbf{x} = \mathbf{b}$ gives $\mathbf{A}\,\delta\mathbf{x} = -\delta\mathbf{A}(\mathbf{x}+\delta\mathbf{x})$, so $\|\delta\mathbf{x}\| \leq \|\mathbf{A}^{-1}\| \, \|\delta\mathbf{A}\| \, \|\mathbf{x} + \delta\mathbf{x}\|$. Dividing by $\|\mathbf{x} + \delta\mathbf{x}\|$ and writing $\|\mathbf{A}^{-1}\| = \operatorname{cond}(\mathbf{A})/\|\mathbf{A}\|$ gives the bound. □

When both $\mathbf{A}$ and $\mathbf{b}$ are perturbed, the relative change in the solution is bounded, to first order, by $\operatorname{cond}(\mathbf{A})$ times the sum of the relative perturbations in the data.

## 6.5 Residual and error

The residual $\mathbf{r} = \mathbf{b} - \mathbf{A}\hat{\mathbf{x}}$ is computable, while the error $\mathbf{x} - \hat{\mathbf{x}}$ is not; the condition number relates them. From $\mathbf{r} = \mathbf{A}\mathbf{x} - \mathbf{A}\hat{\mathbf{x}} = \mathbf{A}(\mathbf{x} - \hat{\mathbf{x}})$ the error is $\mathbf{x} - \hat{\mathbf{x}} = \mathbf{A}^{-1}\mathbf{r}$, and the argument of Theorem 6.2 gives

$$\frac{\|\mathbf{x} - \hat{\mathbf{x}}\|}{\|\mathbf{x}\|} \leq \operatorname{cond}(\mathbf{A}) \, \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}.$$

A small relative residual guarantees a small relative error *only* when $\mathbf{A}$ is well-conditioned. For an ill-conditioned matrix a computed solution can have a tiny residual and still be wrong in every digit, so the residual by itself is not a certificate of accuracy.

## 6.6 Ill-conditioning in practice

The condition number predicts the number of digits lost in a solve: in arithmetic with unit roundoff $u$ the computed solution typically has relative error of order $\operatorname{cond}(\mathbf{A})\,u$, so about $\log_{10}\operatorname{cond}(\mathbf{A})$ significant decimal digits are lost. With double precision ($u \approx 10^{-16}$) a matrix with $\operatorname{cond}(\mathbf{A}) \approx 10^{10}$ leaves roughly six correct digits, and one with $\operatorname{cond}(\mathbf{A}) \approx 10^{16}$ may leave none.

**Example 6.4.** The $n \times n$ Hilbert matrix $H_{ij} = 1/(i+j-1)$ is the standard ill-conditioned example. Its condition number grows explosively: $\operatorname{cond}_2(H)$ is about $10^3$ at $n = 4$ and exceeds $10^{10}$ by $n = 8$. Solving $\mathbf{H}\mathbf{x} = \mathbf{b}$ in double precision is hopeless past about $n = 12$, even though $\mathbf{H}$ is symmetric positive definite and nominally nonsingular. Ill-conditioning, not singularity, is the practical obstacle to an accurate solution.

**Example 6.5.** The matrix $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1.0001 \end{smallmatrix}\right)$ has $\mathbf{A}^{-1} = \left(\begin{smallmatrix} 10001 & -10000 \\ -10000 & 10000 \end{smallmatrix}\right)$, so $\|\mathbf{A}\|_\infty = 2.0001$, $\|\mathbf{A}^{-1}\|_\infty = 20001$, and $\operatorname{cond}_\infty(\mathbf{A}) \approx 4 \times 10^4$. The effect is concrete: the solution of $\mathbf{A}\mathbf{x} = (2,\, 2.0001)^{\mathsf{T}}$ is $(1, 1)$, while changing the right-hand side to $(2, 2)^{\mathsf{T}}$—a relative change of about $5 \times 10^{-5}$—moves the solution to $(2, 0)$, a relative change of order one. The condition number forecast exactly this: a small relative perturbation in the data, magnified by $\operatorname{cond}(\mathbf{A})$, produces a large relative change in the answer.

## Exercises

**Exercise 6.1.** Compute $\operatorname{cond}_\infty(\mathbf{A})$ for $\mathbf{A} = \left(\begin{smallmatrix} 1 & 2 \\ 3 & 4 \end{smallmatrix}\right)$ by finding $\mathbf{A}^{-1}$ and the two $\infty$-norms.

**Exercise 6.2.** For the system $\mathbf{A}\mathbf{x} = \mathbf{b}$ with $\mathbf{A} = \left(\begin{smallmatrix} 1 & 1 \\ 1 & 1.0001 \end{smallmatrix}\right)$, estimate $\operatorname{cond}_\infty(\mathbf{A})$ and explain why small changes in $\mathbf{b}$ can produce large changes in $\mathbf{x}$.

**Exercise 6.3.** Prove that $\operatorname{cond}(\mathbf{A}) \geq 1$ for every nonsingular $\mathbf{A}$ and every induced norm, and that $\operatorname{cond}(c\mathbf{A}) = \operatorname{cond}(\mathbf{A})$ for any nonzero scalar $c$.

**Exercise 6.4.** Show that for a symmetric matrix $\operatorname{cond}_2(\mathbf{A}) = |\lambda_{\max}| \,/\, |\lambda_{\min}|$, using $\|\mathbf{A}\|_2 = \rho(\mathbf{A})$.

**Exercise 6.5.** A computed solution of a system with $\operatorname{cond}(\mathbf{A}) = 10^8$ has relative residual $10^{-15}$. What can and cannot be concluded about its relative error?

**Exercise 6.6.** ⋆Show that $\operatorname{cond}_2(\mathbf{Q}) = 1$ for any orthogonal matrix $\mathbf{Q}$, and interpret this as the statement that orthogonal transformations neither amplify nor damp relative errors.

# Chapter 7

# Polynomial Interpolation

Given the values of a function at a finite set of points, interpolation constructs a polynomial that reproduces those values and approximates the function between them. Interpolating polynomials underlie numerical differentiation, integration, and the multistep methods for differential equations.

## 7.1 The interpolation problem

Given $n+1$ distinct nodes $x_0, x_1, \dots, x_n$ and corresponding values $y_0, \dots, y_n$, the *interpolation problem* asks for a polynomial $p$ of degree at most $n$ with $p(x_i) = y_i$ for every $i$.

**Theorem 7.1.** *For distinct nodes the interpolating polynomial of degree at most $n$ exists and is unique.*

Uniqueness is immediate: two such polynomials would differ by a polynomial of degree at most $n$ with $n+1$ roots, hence the zero polynomial. Existence is established constructively by the Lagrange and Newton forms below.

## 7.2 The Lagrange form

Define the *Lagrange basis polynomials*

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j},$$

each of degree $n$ and satisfying $L_i(x_k) = \delta_{ik}$. The interpolating polynomial is then

$$p(x) = \sum_{i=0}^{n} y_i \, L_i(x),$$

since at each node $x_k$ every term vanishes except $y_k L_k(x_k) = y_k$. The Lagrange form is explicit and theoretically convenient, but adding a new node requires recomputing every basis polynomial.

**Example 7.2.** Through $(0,1),(1,3),(2,7)$ the Lagrange form gives $p(x) = 1 \cdot \frac{(x-1)(x-2)}{2} + 3 \cdot \frac{x(x-2)}{-1} + 7 \cdot \frac{x(x-1)}{2} = x^2 + x + 1$.

## 7.3 Newton's divided differences

The *Newton form* builds the polynomial incrementally and accommodates new nodes cheaply. The *divided differences* are defined recursively by $f[x_i] = y_i$ and

$$f[x_i, \ldots, x_{i+k}] = \frac{f[x_{i+1}, \ldots, x_{i+k}] - f[x_i, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}.$$

The interpolating polynomial is

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

Adding a node appends one term without disturbing those already computed, and the divided differences are conveniently organized in a triangular table.

## 7.4 The interpolation error

**Theorem 7.3.** *If $f$ has $n+1$ continuous derivatives on an interval containing the nodes and a point $x$, then there is a point $\xi$ in that interval with*

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$$

The error has two factors: the derivative $f^{(n+1)}$, which the interpolant cannot control, and the node polynomial $\prod (x - x_i)$, which depends on the placement of the nodes. Reducing the node spacing $h$ makes the product small, giving an error of order $h^{n+1}$ for fixed degree.

## 7.5 The Runge phenomenon

Increasing the polynomial degree on *equally spaced* nodes does not always improve the approximation. For $f(x) = 1/(1+25x^2)$ on $[-1,1]$ the equispaced interpolant develops violent oscillations near the endpoints that grow without bound as $n \to \infty$—the *Runge phenomenon*, caused by the node polynomial being large near the ends of the interval. Clustering the nodes toward the endpoints, as the *Chebyshev nodes* $x_k = \cos(k\pi/n)$ do, controls the node polynomial and restores convergence.

**Example 7.4.** For the data $(0,1),(1,2),(2,9),(3,28)$ the divided-difference table is

| $x_i$ | $f[\,\cdot\,]$ | first | second | third |
|---|---|---|---|---|
| 0 | 1 | | | |
| 1 | 2 | 1 | | |
| 2 | 9 | 7 | 3 | |
| 3 | 28 | 19 | 6 | 1 |

The top diagonal supplies the Newton coefficients, giving $p(x) = 1 + x + 3x(x-1) + x(x-1)(x-2)$, which expands to $x^3 + 1$. Adding a fifth point would append one row and one coefficient, leaving the rest of the table intact.

## Exercises

**Exercise 7.1.** Construct the Lagrange interpolating polynomial through $(-1,2),(0,1),(2,5)$ and simplify.

**Exercise 7.2.** Build the divided-difference table for the data of the previous exercise and write the Newton form, verifying it equals the Lagrange form.

**Exercise 7.3.** Use Theorem 7.3 to bound the error of linear interpolation ($n = 1$) of $f(x) = \sin x$ on an interval of width $h$.

**Exercise 7.4.** Show that the Lagrange basis polynomials satisfy $\sum_{i=0}^{n} L_i(x) = 1$ for all $x$.

**Exercise 7.5.** ⋆Explain, using the node polynomial in Theorem 7.3, why Chebyshev nodes reduce the maximum interpolation error compared with equally spaced nodes.

**Exercise 7.6.** Find the cubic interpolating $(0,1),(1,2),(2,5),(3,10)$ and identify the resulting polynomial.

**Exercise 7.7.** Construct the divided-difference table for the data $(1,0),(2,3),(4,5),(5,4)$ and write the Newton form.

**Exercise 7.8.** Bound the error of quadratic interpolation of $f(x) = \mathrm{e}^x$ at the equally spaced nodes $0, \frac{1}{2}, 1$, using Theorem 7.3.

**Exercise 7.9.** Show that the leading coefficient of the degree-$n$ interpolating polynomial equals the divided difference $f[x_0, \ldots, x_n]$.

**Exercise 7.10.** Compute the maximum of $|(x - x_0)(x - x_1)(x - x_2)|$ on $[-1, 1]$ for three equally spaced nodes and for the three Chebyshev nodes, and compare.

**Exercise 7.11.** Use inverse interpolation—interpolating $x$ as a function of $y$—to estimate the root of $f$ from the table $f(1) = -1$, $f(2) = 0.5$, $f(3) = 2$.

**Exercise 7.12.** ⋆Show that the Vandermonde determinant of distinct nodes equals $\prod_{i<j}(x_j - x_i) \neq 0$, giving a second proof that the interpolating polynomial exists.

# Chapter 8

# Initial Value Problems: Existence, Uniqueness, and Stability

Numerical methods for ordinary differential equations begin with the initial value problem

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0. \tag{8.1}$$

Before approximating a solution one must know that a solution exists, is unique, and depends continuously on the data; otherwise no numerical method can be meaningful.

## 8.1 The Lipschitz condition

The function $f$ is said to satisfy a *Lipschitz condition* in $y$ on a region if there is a constant $L$, the *Lipschitz constant*, with

$$|f(t, y_1) - f(t, y_2)| \le L\, |y_1 - y_2|$$

for all points $(t, y_1), (t, y_2)$ in the region. If $f$ has a bounded partial derivative $f_y$, then $L = \max |f_y|$ serves, by the mean value theorem. The Lipschitz condition limits how fast $f$ can change in $y$ and is exactly what is needed to control the divergence of nearby solutions.

## 8.2 Existence and uniqueness

**Theorem 8.1** (Picard)**.** *If $f$ is continuous on the strip $t \in [t_0, b]$, $y \in \mathbb{R}$, and satisfies a Lipschitz condition in $y$ there, then the initial value problem (8.1) has a unique solution $y(t)$ on $[t_0, b]$.*

The proof recasts (8.1) as the integral equation $y(t) = y_0 + \int_{t_0}^{t} f(s, y(s))\,ds$ and shows that the Picard iteration $y_{k+1}(t) = y_0 + \int_{t_0}^{t} f(s, y_k(s))\,ds$ is a contraction, whose unique fixed point is the solution. The Lipschitz constant is precisely the quantity that makes the iteration contract.

## 8.3 Well-posedness and stability

A problem is *well-posed* if a small change in the data produces only a correspondingly small change in the solution. For (8.1) this follows from the Lipschitz condition.

**Theorem 8.2.** *Let $y$ and $\tilde{y}$ solve $y' = f(t, y)$ with initial values $y_0$ and $\tilde{y}_0$, where $f$ is Lipschitz in $y$ with constant $L$. Then*

$$|y(t) - \tilde{y}(t)| \le \mathrm{e}^{L(t-t_0)}\,|y_0 - \tilde{y}_0|\,.$$

Two solutions can diverge no faster than exponentially, at a rate set by $L$. The bound is the continuous analogue of the error recurrence that governs the numerical methods: it shows that perturbations—whether in the data or introduced by a numerical step—are amplified by at most $\mathrm{e}^{L(t-t_0)}$ over the interval.

**Example 8.3.** For $y' = \lambda y$ the constant is $L = |\lambda|$ and two solutions differ by $\mathrm{e}^{\lambda t}\,|y_0 - \tilde{y}_0|$. When $\lambda < 0$ solutions converge and the problem is stable; when $\lambda > 0$ they separate exponentially, a genuine sensitivity that any numerical method must contend with.

**Example 8.4.** The Picard iteration for $y' = y$, $y(0) = 1$ makes the existence proof concrete. Starting from $y_0(t) \equiv 1$,

$$y_1 = 1 + \int_0^t 1\,ds = 1 + t, \quad y_2 = 1 + \int_0^t (1 + s)\,ds = 1 + t + \tfrac{t^2}{2}, \quad y_3 = 1 + t + \tfrac{t^2}{2} + \tfrac{t^3}{6},$$

the partial sums of $\mathrm{e}^t$. The iterates converge to the exact solution $\mathrm{e}^t$ on any bounded interval, exactly as the contraction argument guarantees.

## Exercises

**Exercise 8.1.** Find a Lipschitz constant for $f(t, y) = t^2 + y^2$ on the rectangle $|t| \le 1$, $|y| \le 2$.

**Exercise 8.2.** Show that $f(t, y) = \sqrt{|y|}$ is not Lipschitz near $y = 0$, and that the initial value problem $y' = \sqrt{|y|}$, $y(0) = 0$ has more than one solution.

**Exercise 8.3.** Carry out two Picard iterations for $y' = y$, $y(0) = 1$, starting from $y_0(t) \equiv 1$, and identify the emerging series for $\mathrm{e}^{t}$.

**Exercise 8.4.** Use Theorem 8.2 to bound the effect on the solution at $t = 1$ of changing the initial value of $y' = 2y$, $y(0) = 1$ by $10^{-3}$.

**Exercise 8.5.** ⋆Prove Theorem 8.2 by applying Gronwall's inequality to $u(t) = |y(t) - \tilde{y}(t)|$.

**Exercise 8.6.** Find a Lipschitz constant for $f(t, y) = ty$ on $|t| \le 2$, $|y| \le 3$.

**Exercise 8.7.** Determine whether the problem $y' = y^{1/3}$, $y(0) = 0$ has a unique solution, and relate the answer to the Lipschitz condition.

**Exercise 8.8.** Compute three Picard iterates for $y' = t + y$, $y(0) = 0$, starting from $y_0(t) \equiv 0$.

**Exercise 8.9.** Use Theorem 8.2 to bound the separation at $t = 2$ of two solutions of $y' = 3y$ whose initial values differ by $10^{-4}$.

**Exercise 8.10.** Show that $f(t, y) = |y|$ satisfies a Lipschitz condition with $L = 1$.

**Exercise 8.11.** Verify that $y' = y^2$, $y(0) = 1$ has the solution $y = 1/(1 - t)$, which exists only for $t < 1$, illustrating that existence may be merely local.

**Exercise 8.12.** ⋆State Gronwall's inequality and use it to bound $|y(t)|$ for a solution of $y' = a(t)y$ with $|a(t)| \le M$.

# Chapter 9

# Euler's Method

## 9.1 Derivation

Partition $[t_0, b]$ into steps of size $h$ with $t_n = t_0 + nh$. Replacing the derivative in (8.1) by a forward difference, $y'(t_n) \approx [y(t_{n+1}) - y(t_n)]/h$, and using the differential equation $y'(t_n) = f(t_n, y(t_n))$, gives *Euler's method*

$$y_{n+1} = y_n + h\, f(t_n, y_n), \qquad y_0 \text{ given},$$

where $y_n$ denotes the numerical approximation to $y(t_n)$. Geometrically, each step follows the tangent line to the solution through the current point for a time $h$.

## 9.2 Local truncation error

The accuracy of a one-step method $y_{n+1} = y_n + h\,\phi(t_n, y_n, h)$ is measured by its *local truncation error*, the amount by which the exact solution fails to satisfy the method,

$$\tau_{n+1} = \frac{y(t_{n+1}) - y(t_n)}{h} - \phi\big(t_n, y(t_n), h\big).$$

For Euler's method $\phi = f$, and Taylor expansion of $y(t_{n+1})$ gives

$$\tau_{n+1} = \frac{h}{2}\, y''(\xi_n) = \mathcal{O}(h).$$

A method is *consistent of order* $p$ if $\tau = \mathcal{O}(h^p)$; Euler's method is consistent of order 1. The single-step error in $y_{n+1}$ itself is $h\tau_{n+1} = \mathcal{O}(h^2)$, and these single-step errors accumulate over the $\mathcal{O}(1/h)$ steps to give a global error of order $h$.

**Example 9.1.** Applied to $y' = y$, $y(0) = 1$ with $h = 0.1$, Euler's method gives $y_1 = 1.1$, $y_2 = 1.21$, …, $y_{10} = 1.1^{10} \approx 2.594$, against the exact $\mathrm{e} \approx 2.71828$. Halving $h$ roughly halves the error, the hallmark of first-order accuracy.

**Example 9.2.** For $y' = t - y$, $y(0) = 1$, whose exact solution is $y(t) = t - 1 + 2\mathrm{e}^{-t}$, Euler's method with $h = 0.2$ gives the values below.

| $t_n$ | $y_n$ (Euler) | $y(t_n)$ | error |
|---|---|---|---|
| 0.0 | 1.0000 | 1.0000 | 0.0000 |
| 0.2 | 0.8000 | 0.8375 | 0.0375 |
| 0.4 | 0.6800 | 0.7406 | 0.0606 |
| 0.6 | 0.6240 | 0.6976 | 0.0736 |
| 0.8 | 0.6192 | 0.6987 | 0.0795 |
| 1.0 | 0.6554 | 0.7358 | 0.0804 |

The error grows roughly in proportion to the number of steps, reaching about 0.08 at $t = 1$. Halving the step to $h = 0.1$ roughly halves it, confirming the first-order accuracy; obtaining even three correct digits this way would require a very small step.

```
function [t, y] = euler(f, t0, y0, h, N)
% EULER N steps of Euler's method for y' = f(t,y), y(t0)=y0.
t = t0 + (0:N)'*h;
y = zeros(N+1, 1); y(1) = y0;
for n = 1:N
    y(n+1) = y(n) + h*f(t(n), y(n));
end
end
```

# Exercises

**Exercise 9.1.** Apply Euler's method with $h = 0.25$ to $y' = t - y$, $y(0) = 1$, computing $y_1, \dots, y_4$.

**Exercise 9.2.** Derive Euler's method from the integral form $y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(s, y(s))\, ds$ by approximating the integral with the left-endpoint rule.

**Exercise 9.3.** Show that the local truncation error of Euler's method is $\frac{h}{2} y''(\xi_n)$ by Taylor expansion.

**Exercise 9.4.** For $y' = -y$, $y(0) = 1$, find the Euler approximation $y_n$ explicitly as a power of $1 - h$, and discuss its behavior for $h < 1$ and $h > 2$.

**Exercise 9.5.** ⋆*Backward* (implicit) Euler is $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$. Apply it to $y' = -y$ and show that the approximation decays for every $h > 0$, unlike the explicit method.

**Exercise 9.6.** Apply Euler's method with $h = 0.1$ to $y' = y - t$, $y(0) = 2$, computing $y_1, y_2, y_3$.

**Exercise 9.7.** Apply Euler with $h = 0.25$ to $y' = t^2$, $y(0) = 0$, and compare $y_4$ with the exact value at $t = 1$.

**Exercise 9.8.** Show that Euler applied to $y' = ay$ gives $y_n = (1 + ah)^n y_0$.

**Exercise 9.9.** Find the local truncation error of Euler's method for $y' = \cos t$.

**Exercise 9.10.** For $y' = -2y$, determine the range of step sizes $h$ for which the Euler iterates decay in magnitude.

**Exercise 9.11.** Compute the Euler global error at $t = 1$ for $y' = y$, $y(0) = 1$ with $h = 0.1$ and $h = 0.05$, and confirm the first-order ratio.

**Exercise 9.12.** ⋆Show that backward Euler applied to $y' = \lambda y$ with $\operatorname{Re}\lambda < 0$ produces decaying iterates for every $h > 0$.

# Chapter 10

# Taylor Methods

## 10.1 Higher-order Taylor expansion

Euler's method keeps only the first-order term of the Taylor expansion of the solution. Retaining more terms raises the order. Differentiating the equation $y' = f(t, y)$ gives the higher derivatives of the solution in terms of $f$,

$$y'' = f_t + f_y\, f, \qquad y''' = f_{tt} + 2f_{ty} f + f_{yy} f^2 + f_y(f_t + f_y f),$$

and so on, each computed by the chain rule. The *Taylor method of order p* uses the truncated Taylor series,

$$y_{n+1} = y_n + h\, f + \frac{h^2}{2}\, y'' + \cdots + \frac{h^p}{p!}\, y^{(p)},$$

with all derivatives evaluated at $(t_n, y_n)$.

## 10.2 Order and the role of derivatives

The Taylor method of order $p$ has local truncation error $\tau_{n+1} = \frac{h^p}{(p+1)!} y^{(p+1)}(\xi_n) = \mathcal{O}(h^p)$, hence global error $\mathcal{O}(h^p)$. It is the natural high-order method, but it has a serious practical drawback: it requires the analytic derivatives $y'', y''', \ldots$, that is the partial derivatives of $f$, which must be derived by hand and re-derived for each new equation. This expense is what the Runge–Kutta methods avoid.

**Example 10.1.** For $y' = t + y$ the second derivative is $y'' = 1 + y' = 1 + t + y$, so the order-two Taylor method is $y_{n+1} = y_n + h(t_n + y_n) + \frac{h^2}{2}(1 + t_n + y_n)$, with local truncation error $\mathcal{O}(h^2)$ and global error $\mathcal{O}(h^2)$, one order better than Euler at the cost of forming $y''$.

**Example 10.2.** For $y' = t + y$, $y(0) = 1$, with exact solution $y(t) = 2\mathrm{e}^{t} - t - 1$, the order-two Taylor method $y_{n+1} = y_n + h(t_n + y_n) + \frac{h^2}{2}(1 + t_n + y_n)$ with $h = 0.1$ gives

| $t_n$ | $y_n$ (Taylor 2) | $y(t_n)$ | error |
|---|---|---|---|
| 0.1 | 1.110000 | 1.110342 | $3.4 \times 10^{-4}$ |
| 0.2 | 1.242050 | 1.242806 | $7.6 \times 10^{-4}$ |
| 0.3 | 1.398465 | 1.399718 | $1.3 \times 10^{-3}$ |

The error is of order $10^{-3}$, an order of magnitude better than Euler at the same step, and shrinks by a factor of about four when $h$ is halved, confirming second-order accuracy.

## Exercises

**Exercise 10.1.** For $y' = y - t^2 + 1$ compute $y''$ and $y'''$ and write the order-three Taylor method.

**Exercise 10.2.** Apply the order-two Taylor method with $h = 0.2$ to $y' = t + y$, $y(0) = 1$, computing $y_1$ and $y_2$.

**Exercise 10.3.** State the local truncation error of the order-$p$ Taylor method and deduce its global order.

**Exercise 10.4.** Explain why the Taylor method of order 1 is exactly Euler's method.

**Exercise 10.5.** ⋆For an equation $y' = f(t)$ depending on $t$ alone, show that the order-$p$ Taylor method reduces to a Taylor approximation of the integral $\int f$, and relate it to a quadrature rule.

**Exercise 10.6.** Write the order-three Taylor method for $y' = y^2$.

**Exercise 10.7.** Apply the order-two Taylor method with $h = 0.1$ to $y' = t - y$, $y(0) = 1$, computing $y_1$ and $y_2$.

**Exercise 10.8.** Compute $y''$ and $y'''$ for the equation $y' = ty$.

**Exercise 10.9.** Show that the local truncation error of the order-two Taylor method is $\frac{h^2}{6} y'''(\xi_n)$.

**Exercise 10.10.** For an equation $y' = f(t)$ depending on $t$ alone, show that the order-$p$ Taylor method is the Taylor expansion of $\int f$.

**Exercise 10.11.** Compare the work of the order-two Taylor method, which needs $f$, $f_t$, and $f_y$, with that of the midpoint Runge–Kutta method.

**Exercise 10.12.** ⋆Express $y^{(4)}$ in terms of $f$ and its partial derivatives.

# Chapter 11

# Runge–Kutta Methods

## 11.1 The idea

Runge–Kutta methods attain the accuracy of a Taylor method without computing any derivatives of $f$, by sampling $f$ at several points within each step and combining the samples so as to match the Taylor expansion to the desired order.

## 11.2 Second-order methods

A two-stage method has the form

$$k_1 = f(t_n, y_n), \qquad k_2 = f(t_n + ch,\, y_n + ch\, k_1), \qquad y_{n+1} = y_n + h(ak_1 + bk_2).$$

Expanding $k_2$ by Taylor's theorem and matching the result with the order-two Taylor method forces the *order conditions* $a + b = 1$ and $bc = \tfrac{1}{2}$. Two standard choices solve them: the *midpoint method*, $a = 0$, $b = 1$, $c = \tfrac{1}{2}$,

$$y_{n+1} = y_n + h\, f\big(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2} f(t_n, y_n)\big)\,,$$

and *Heun's method*, $a = b = \tfrac{1}{2}$, $c = 1$. Each has local truncation error $\mathcal{O}(h^2)$.

## 11.3 The classical fourth-order method

The most widely used method is the four-stage, fourth-order scheme

$$k_1 = f(t_n, y_n), \qquad k_2 = f\!\left(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_1\right),$$
$$k_3 = f\!\left(t_n + \tfrac{h}{2},\, y_n + \tfrac{h}{2}k_2\right), \quad k_4 = f(t_n + h,\, y_n + hk_3),$$

$$y_{n+1} = y_n + \frac{h}{6}\big(k_1 + 2k_2 + 2k_3 + k_4\big),$$

with local truncation error $\mathcal{O}(h^4)$ and global error $\mathcal{O}(h^4)$. Its balance of accuracy, stability, and simplicity—four function evaluations per step and no derivatives—makes it the default method for nonstiff problems. The coefficients of a Runge–Kutta method are conveniently displayed in a *Butcher tableau*, and the algebra of matching Taylor terms gives the *order conditions* that the coefficients must satisfy.

**Example 11.1.** Applied to $y' = y$, $y(0) = 1$ with the comparatively large step $h = 0.25$, the classical fourth-order method produces

| $t_n$ | $y_n$ (RK4) | $\mathrm{e}^{t_n}$ | error |
|---|---|---|---|
| 0.00 | 1.000000 | 1.000000 | — |
| 0.25 | 1.284017 | 1.284025 | $8 \times 10^{-6}$ |
| 0.50 | 1.648699 | 1.648721 | $2.2 \times 10^{-5}$ |
| 0.75 | 2.116949 | 2.117000 | $5.1 \times 10^{-5}$ |
| 1.00 | 2.718214 | 2.718282 | $6.8 \times 10^{-5}$ |

With the same number of steps that left Euler's method with an error near 0.08, RK4 attains five correct digits. This accuracy per step, at the cost of four function evaluations and no derivatives, is what makes the method the standard choice.

```
function [t, y] = rk4(f, t0, y0, h, N)
% RK4 N steps of the classical fourth-order Runge-Kutta method.
t = t0 + (0:N)'*h;
y = zeros(N+1, 1); y(1) = y0;
for n = 1:N
    k1 = f(t(n), y(n));
    k2 = f(t(n)+h/2, y(n)+h/2*k1);
    k3 = f(t(n)+h/2, y(n)+h/2*k2);
    k4 = f(t(n)+h, y(n)+h*k3);
    y(n+1) = y(n) + h/6*(k1 + 2*k2 + 2*k3 + k4);
end
end
```

## Exercises

**Exercise 11.1.** Apply the midpoint method with $h = 0.2$ to $y' = t + y$, $y(0) = 1$, and compare $y_1$ with the order-two Taylor result.

**Exercise 11.2.** Verify that Heun's method, $a = b = \frac{1}{2}$, $c = 1$, satisfies the order-two conditions $a + b = 1$ and $bc = \frac{1}{2}$.

**Exercise 11.3.** Take one step of the classical RK4 method on $y' = y$, $y(0) = 1$ with $h = 0.5$, and compare $y_1$ with $\mathrm{e}^{0.5}$.

**Exercise 11.4.** Count the function evaluations per step of Euler, the midpoint method, and RK4, and relate the count to the order of each.

**Exercise 11.5.** ⋆Show that for $y' = f(t)$ the classical RK4 method reduces to Simpson's rule for $\int_{t_n}^{t_{n+1}} f(t)\,dt$.

**Exercise 11.6.** Apply the midpoint method with $h = 0.2$ to $y' = ty$, $y(0) = 1$, taking two steps.

**Exercise 11.7.** Verify directly that the midpoint method satisfies the order-two conditions $a + b = 1$ and $bc = \frac{1}{2}$.

**Exercise 11.8.** Take one step of classical RK4 on $y' = -y$, $y(0) = 1$ with $h = 0.5$ and compare $y_1$ with $\mathrm{e}^{-0.5}$.

**Exercise 11.9.** Write the Butcher tableau of the classical fourth-order method.

**Exercise 11.10.** Show that for $y' = f(t)$ the classical RK4 method reduces to Simpson's rule.

**Exercise 11.11.** Count the function evaluations needed to integrate to $T$ in $N$ steps by Euler, the midpoint method, and RK4.

**Exercise 11.12.** ⋆Derive the two-stage order conditions $a + b = 1$, $bc = \frac{1}{2}$ by expanding $k_2$ and matching the order-two Taylor method.

# Chapter 12

# Convergence of One-Step Methods

## 12.1 Local and global error

The local truncation error measures the error committed in a single step, assuming exact data at its start. The quantity of real interest is the *global error* $e_n = y(t_n) - y_n$, the accumulated discrepancy after $n$ steps. The central theorem of the subject relates the two: consistency, together with a stability property that one-step methods possess automatically, implies convergence at the same order.

## 12.2 The convergence theorem

**Theorem 12.1.** *Consider a one-step method $y_{n+1} = y_n + h\,\phi(t_n, y_n, h)$ whose increment function $\phi$ is Lipschitz in $y$ with constant $\Lambda$, and whose local truncation error satisfies $|\tau_n| \le C h^p$. If $y_0$ is exact, then the global error is bounded by*

$$|e_n| \le \frac{C h^p}{\Lambda}\Big(\mathrm{e}^{\Lambda (t_n - t_0)} - 1\Big) = \mathcal{O}(h^p),$$

*so the method converges with order $p$.*

*Proof.* The exact solution satisfies $y(t_{n+1}) = y(t_n) + h\phi(t_n, y(t_n), h) + h\tau_{n+1}$, while $y_{n+1} = y_n + h\phi(t_n, y_n, h)$. Subtracting and using the Lipschitz bound on $\phi$,

$$|e_{n+1}| \le |e_n| + h\Lambda\,|e_n| + h\,|\tau_{n+1}| = (1 + h\Lambda)\,|e_n| + hCh^p.$$

With $e_0 = 0$, induction gives $|e_n| \le hCh^p \sum_{k=0}^{n-1} (1+h\Lambda)^k = \frac{Ch^p}{\Lambda}\big((1+h\Lambda)^n - 1\big)$, and $(1+h\Lambda)^n \le \mathrm{e}^{nh\Lambda} = \mathrm{e}^{\Lambda(t_n - t_0)}$ yields the stated bound. $\square$

The order of convergence equals the order of consistency: Euler converges as $\mathcal{O}(h)$, the second-order Runge–Kutta and Taylor methods as $\mathcal{O}(h^2)$, and classical RK4 as $\mathcal{O}(h^4)$. The factor $\mathrm{e}^{\Lambda(t_n - t_0)}$ is the discrete counterpart of the well-posedness bound of Chapter 8.

## 12.3 Roundoff and the optimal step size

In finite precision each step also incurs a rounding error of size about $u$, the unit roundoff, which contributes a term of order $u/h$ to the total error—it grows as $h$ shrinks. The total error therefore behaves like $C_1 h^p + C_2 u/h$, which is minimized at a step size $h_{\mathrm{opt}}$ of order $u^{1/(p+1)}$. Reducing $h$ below this point increases the error, because accumulated roundoff overwhelms the gain in truncation accuracy.

**Example 12.2.** The first-order rate is visible in the global error of Euler's method for $y' = -y$, $y(0) = 1$ at $t = 1$, where the exact value is $\mathrm{e}^{-1} \approx 0.367879$:

| $h$ | steps | error | ratio |
|---|---|---|---|
| 0.2 | 5 | $4.02 \times 10^{-2}$ | — |
| 0.1 | 10 | $1.92 \times 10^{-2}$ | 2.09 |
| 0.05 | 20 | $9.39 \times 10^{-3}$ | 2.04 |
| 0.025 | 40 | $4.65 \times 10^{-3}$ | 2.02 |

Each halving of $h$ halves the error, the ratio tending to 2, which is the defining behavior of a first-order method and matches the $\mathcal{O}(h)$ bound of the convergence theorem.

## Exercises

**Exercise 12.1.** State the global order of convergence of Euler, the midpoint method, and RK4, and the factor by which the error changes when $h$ is halved for each.

**Exercise 12.2.** In the convergence theorem, verify the summation $\sum_{k=0}^{n-1} (1 + h\Lambda)^k = \frac{(1+h\Lambda)^n - 1}{h\Lambda}$.

**Exercise 12.3.** For $y' = -y$, $y(0) = 1$, compute the Euler global error at $t = 1$ for $h = 0.1$ and $h = 0.05$, and confirm the first-order ratio.

**Exercise 12.4.** Using the model error $C_1 h + C_2 u/h$ for Euler, find the step size that minimizes the total error and the corresponding minimal error.

**Exercise 12.5.** ⋆Explain why the convergence theorem requires only that $\phi$ be Lipschitz, and verify that the RK4 increment function inherits a Lipschitz constant from that of $f$.

**Exercise 12.6.** State the global order of Euler, the midpoint method, and RK4, and the factor by which the error changes when $h$ is halved.

**Exercise 12.7.** Verify the geometric sum $\sum_{k=0}^{n-1}(1+h\Lambda)^k = \dfrac{(1+h\Lambda)^n - 1}{h\Lambda}$ used in the convergence proof.

**Exercise 12.8.** Compute the Euler global error at $t = 1$ for $y' = -y$, $y(0) = 1$ at $h = 0.1$ and $h = 0.05$, confirming $\mathcal{O}(h)$ behavior.

**Exercise 12.9.** Minimize the model total error $C_1 h + C_2 u/h$ and find the optimal step size and minimal error.

**Exercise 12.10.** Show that the increment function of RK4 inherits a Lipschitz constant from that of $f$.

**Exercise 12.11.** For $y' = \lambda y$, find the amplification factor of RK4 and the interval of absolute stability on the negative real axis.

**Exercise 12.12.** ⋆Prove the discrete lemma: if $|e_{n+1}| \le (1+h\Lambda)\,|e_n| + h\delta$ and $e_0 = 0$, then $|e_n| \le \dfrac{\delta}{\Lambda}\big(\mathrm{e}^{\Lambda(t_n - t_0)} - 1\big)$.

# Chapter 13

# Multistep Methods: Adams–Bashforth and Adams–Moulton

## 13.1 The idea

A one-step method discards all information once a step is complete. A *multistep* method instead reuses several previously computed values to take the next step, achieving high order with only one new function evaluation. The Adams methods are derived by integrating the differential equation over one step,

$$y(t_{n+1}) = y(t_n) + \int_{t_n}^{t_{n+1}} f(t, y(t))\, dt,$$

and replacing $f$ by the polynomial that interpolates its known past values.

## 13.2 Adams–Bashforth methods

Interpolating $f$ through the points $t_n, t_{n-1}, \ldots, t_{n-k+1}$ (all in the past) and integrating gives the explicit *Adams–Bashforth* formulas. The two- and three-step members are

$$y_{n+1} = y_n + \frac{h}{2}\big(3f_n - f_{n-1}\big), \qquad y_{n+1} = y_n + \frac{h}{12}\big(23f_n - 16f_{n-1} + 5f_{n-2}\big),$$

where $f_j = f(t_j, y_j)$. The $k$-step method has order $k$. Being explicit, each step is a single evaluation of $f$, but the method needs $k$ starting values, which are supplied by a one-step method such as RK4.

## 13.3 Adams–Moulton methods

Including the as-yet-unknown point $t_{n+1}$ among the interpolation nodes gives the implicit *Adams–Moulton* formulas. The one- and two-step members are

$$y_{n+1} = y_n + \frac{h}{2}\bigl(f_{n+1} + f_n\bigr), \qquad y_{n+1} = y_n + \frac{h}{12}\bigl(5f_{n+1} + 8f_n - f_{n-1}\bigr).$$

The first is the *trapezoidal method*. Because $f_{n+1} = f(t_{n+1}, y_{n+1})$ appears on both sides, each step requires solving an equation for $y_{n+1}$, but for the same order an Adams–Moulton method is more accurate and has better stability than the corresponding Adams–Bashforth method.

**Example 13.1.** For $y' = y$, $y(0) = 1$ with $h = 0.1$, taking the first step by Euler ($y_1 = 1.1$) and continuing with the two-step Adams–Bashforth method gives

| $t_n$ | $y_n$ (AB2) | $\mathrm{e}^{t_n}$ | error |
|---|---|---|---|
| 0.1 | 1.100000 | 1.105171 | $5.2 \times 10^{-3}$ |
| 0.2 | 1.215000 | 1.221403 | $6.4 \times 10^{-3}$ |
| 0.3 | 1.342250 | 1.349859 | $7.6 \times 10^{-3}$ |
| 0.4 | 1.482838 | 1.491825 | $9.0 \times 10^{-3}$ |
| 0.5 | 1.638151 | 1.648721 | $1.06 \times 10^{-2}$ |

The error is held to about $10^{-2}$ with a single function evaluation per step after the start. Part of it traces to the first-order Euler starting value; a second-order starting step would reduce the early error and let the method's $\mathcal{O}(h^2)$ accuracy show more cleanly.

```
function [t, y] = ab2(f, t0, y0, h, N)
% AB2 Two-step Adams-Bashforth; first step by Euler to start.
t = t0 + (0:N)'*h;
y = zeros(N+1, 1); y(1) = y0;
fprev = f(t(1), y(1));
y(2) = y(1) + h*fprev; % Euler starting step
for n = 2:N
    fn = f(t(n), y(n));
    y(n+1) = y(n) + h/2*(3*fn - fprev);
    fprev = fn;
end
end
```

# Exercises

**Exercise 13.1.** Apply the two-step Adams–Bashforth method to $y' = y$, $y(0) = 1$ with $h = 0.1$, using Euler for the first step, and compute $y_2$ and $y_3$.

**Exercise 13.2.** Derive the trapezoidal method by integrating $y' = f$ over $[t_n, t_{n+1}]$ with the trapezoidal rule.

**Exercise 13.3.** Explain why an Adams–Bashforth method needs starting values and how they are obtained.

**Exercise 13.4.** For the trapezoidal method applied to $y' = \lambda y$, solve the implicit relation explicitly for $y_{n+1}$ in terms of $y_n$.

**Exercise 13.5.** ⋆Show that the two-step Adams–Bashforth method has order 2 by expanding its local truncation error in powers of $h$.

**Exercise 13.6.** Apply the two-step Adams–Bashforth method to $y' = t + y$, $y(0) = 1$ with $h = 0.1$, using Euler for the first step, and compute $y_2$ and $y_3$.

**Exercise 13.7.** Derive the two-step Adams–Bashforth formula by integrating the linear polynomial interpolating $f$ at $t_n$ and $t_{n-1}$.

**Exercise 13.8.** Solve the trapezoidal method explicitly for $y_{n+1}$ when applied to $y' = \lambda y$.

**Exercise 13.9.** Determine the order of the three-step Adams–Bashforth method from the leading term of its local truncation error.

**Exercise 13.10.** Explain why a multistep method needs starting values and how they are generated.

**Exercise 13.11.** Compare the number of function evaluations per step of the two-step Adams–Bashforth and two-step Adams–Moulton methods.

**Exercise 13.12.** ⋆Show that the trapezoidal method has local truncation error $-\frac{h^3}{12} y'''(\xi_n)$.

# Chapter 14

# Predictor–Corrector Methods

## 14.1 Combining explicit and implicit methods

An Adams–Moulton method is more accurate and more stable than the Adams–Bashforth method of the same order, but it is implicit: each step requires solving an equation for $y_{n+1}$. A *predictor–corrector* method sidesteps that solve by using an explicit method to predict $y_{n+1}$ and then the implicit formula, evaluated at the predicted value, to correct it. The implicit equation is never solved; the predicted value plays the role of the unknown on the right.

## 14.2 The PECE scheme

Pairing an Adams–Bashforth predictor with an Adams–Moulton corrector of the same order gives the standard cycle, abbreviated PECE for its four operations. Using the two-step predictor and the trapezoidal corrector,

$$\text{(P)}\quad y^*_{n+1} = y_n + \frac{h}{2}\big(3f_n - f_{n-1}\big), \qquad \text{(E)}\quad f^*_{n+1} = f\big(t_{n+1}, y^*_{n+1}\big),$$

$$\text{(C)}\quad y_{n+1} = y_n + \frac{h}{2}\big(f^*_{n+1} + f_n\big), \qquad \text{(E)}\quad f_{n+1} = f\big(t_{n+1}, y_{n+1}\big).$$

The predictor supplies the value at which the corrector evaluates $f$; the final evaluation prepares $f_{n+1}$ for the next step. The cost is two function evaluations per step, far less than the iteration an implicit solve would require, while retaining most of the corrector's accuracy and stability.

## 14.3 Iterating the corrector and estimating error

The corrector may be applied more than once, re-evaluating $f$ at each new $y_{n+1}$; the iteration converges to the true implicit solution when $h\,|\beta f_y| < 1$, which holds for small $h$. In practice a single correction suffices. The difference between predictor and corrector, $y_{n+1} - y^*_{n+1}$, estimates the local truncation error and is the basis of the step-size control of Chapter 16.

**Example 14.1.** Take $y' = y$, $y(0) = 1$ with $h = 0.1$ and the starting values $y_0 = 1$, $y_1 = \mathrm{e}^{0.1} \approx 1.105171$. The two-step Adams–Bashforth predictor gives

$$y^*_2 = y_1 + \tfrac{h}{2}(3f_1 - f_0) = 1.220947,$$

and the trapezoidal corrector, using $f^*_2 = y^*_2$, gives

$$y_2 = y_1 + \tfrac{h}{2}(f^*_2 + f_1) = 1.221477.$$

Against the exact $\mathrm{e}^{0.2} \approx 1.221403$, the predictor is in error by $4.6 \times 10^{-4}$ and the corrected value by only $7.4 \times 10^{-5}$. Their difference, $5.3 \times 10^{-4}$, estimates the error of the predictor and drives the step-size logic.

## Exercises

**Exercise 14.1.** Carry out one PECE step for $y' = t + y$, $y(0) = 1$ with $h = 0.1$, taking the first step by Euler to provide $f_{n-1}$.

**Exercise 14.2.** Explain why a predictor–corrector pair uses two function evaluations per step, and contrast this with solving the implicit corrector exactly.

**Exercise 14.3.** Show that for $y' = \lambda y$ the corrector iteration converges when $|h\lambda| < 2$.

**Exercise 14.4.** Describe how $y_{n+1} - y^*_{n+1}$ serves as a local error estimate, and why predictor and corrector should have the same order for this to work.

**Exercise 14.5.** ⋆Write the PECE scheme using the three-step Adams–Bashforth predictor and the two-step Adams–Moulton corrector, both of order three.

**Exercise 14.6.** Explain how the difference between predictor and corrector yields a local error estimate (Milne's estimate).

**Exercise 14.7.** Count the function evaluations per step in the PECE and PECECE schemes.

# Chapter 15

# Consistency, Stability, and Convergence of Multistep Methods

## 15.1 Linear multistep methods

A general *linear multistep method* has the form

$$\sum_{j=0}^{k} \alpha_j \, y_{n+j} = h \sum_{j=0}^{k} \beta_j \, f_{n+j},$$

with coefficients $\alpha_j, \beta_j$ and $\alpha_k \neq 0$; it is explicit if $\beta_k = 0$ and implicit otherwise. The Adams methods are the special case $\alpha_k = 1$, $\alpha_{k-1} = -1$, all other $\alpha_j = 0$. Two polynomials encode the method,

$$\rho(\xi) = \sum_{j=0}^{k} \alpha_j \xi^j, \qquad \sigma(\xi) = \sum_{j=0}^{k} \beta_j \xi^j.$$

## 15.2 Consistency

The method is *consistent* of order at least one if its local truncation error vanishes as $h \to 0$, which is equivalent to the two conditions

$$\rho(1) = 0, \qquad \rho'(1) = \sigma(1).$$

The first says the method reproduces constant solutions; the second matches the first derivative. Higher order is obtained by matching further terms, expressed as additional linear conditions on the coefficients.

## 15.3 Zero-stability and the root condition

Consistency alone does not guarantee that the numerical solution stays close to the true one; the method must also not amplify small errors as the step count grows. This is *zero-stability*, governed by the roots of $\rho$.

**Theorem 15.1** (Root condition)**.** *A linear multistep method is zero-stable if and only if every root of $\rho$ satisfies $|\xi| \le 1$, and every root with $|\xi| = 1$ is simple.*

The roots of $\rho$ govern the solutions of the difference equation in the limit $h \to 0$; if any root exceeds 1 in magnitude, or a root on the unit circle is repeated, the corresponding parasitic solution grows and swamps the true one.

## 15.4 The Dahlquist equivalence theorem

The two properties together are exactly what convergence requires.

**Theorem 15.2** (Dahlquist)**.** *A linear multistep method is convergent if and only if it is both consistent and zero-stable.*

This equivalence is the central structural result for multistep methods: it reduces the analytic question of convergence to two algebraic checks on the polynomials $\rho$ and $\sigma$. The Adams methods satisfy it because $\rho(\xi) = \xi^k - \xi^{k-1}$ has the single root $\xi = 1$ on the unit circle and all others at the origin.

## 15.5 Absolute stability and stiffness

Zero-stability concerns the limit $h \to 0$; for a fixed positive $h$ the relevant notion is *absolute stability*, studied on the test equation $y' = \lambda y$ with $\operatorname{Re}\lambda < 0$, whose true solution decays. Substituting into the method yields a polynomial in $\xi$ whose coefficients depend on $h\lambda$; the set of $h\lambda$ for which all roots satisfy $|\xi| \le 1$ is the *region of absolute stability*. A method is useful on the test problem only when $h\lambda$ lies in this region, which bounds the step size from above.

Explicit methods have bounded stability regions, so they force impractically small steps on *stiff* problems—those with widely separated decay rates, where the fastest-decaying component, though negligible in the solution, dictates the step size. Implicit methods can have unbounded regions: the backward Euler and trapezoidal methods are *A-stable*, stable for every $h\lambda$ in the left half-plane, which is why stiff problems are solved with implicit methods despite the cost of the implicit solve.

**Example 15.3.** The method $y_{n+2} - 3y_{n+1} + 2y_n = -hf_n$ is consistent: $\rho(\xi) = \xi^2 - 3\xi + 2$ satisfies $\rho(1) = 0$ and $\rho'(1) = -1 = \sigma(1)$. Yet $\rho$ factors as $(\xi - 1)(\xi - 2)$, with a root $\xi = 2$ outside the unit disk, so the root condition fails. Applied to the trivial problem $y' = 0$, $y \equiv 1$, the recurrence $y_{n+2} = 3y_{n+1} - 2y_n$ has general solution $A + B \cdot 2^n$; an initial rounding error of size $\epsilon$ in $y_1$ excites the $2^n$ mode, and by $n = 20$ it has grown to about $10^6\epsilon$. Consistency without zero-stability gives a divergent method, which is what Dahlquist's theorem forbids.

# Exercises

**Exercise 15.1.** Write $\rho$ and $\sigma$ for the two-step Adams–Bashforth method and verify the consistency conditions $\rho(1) = 0$ and $\rho'(1) = \sigma(1)$.

**Exercise 15.2.** Show that the Adams methods are zero-stable by locating the roots of $\rho(\xi) = \xi^k - \xi^{k-1}$.

**Exercise 15.3.** Determine the interval of absolute stability of Euler's method on the real axis by applying it to $y' = \lambda y$.

**Exercise 15.4.** Explain what makes a problem stiff and why explicit methods are inefficient for it.

**Exercise 15.5.** ⋆Apply the backward Euler method to $y' = \lambda y$ and show that its region of absolute stability contains the entire left half-plane, so the method is A-stable.

**Exercise 15.6.** Determine whether the leapfrog method $y_{n+1} = y_{n-1} + 2hf_n$ is zero-stable by examining the roots of its $\rho$.

**Exercise 15.7.** ⋆Show that the region of absolute stability of the trapezoidal method is exactly the left half-plane.

# Chapter 16

# Adaptive Step Size and Error Control

## 16.1 Why vary the step size

A fixed step size is wasteful: it is too small where the solution is smooth and too large where the solution changes rapidly. An *adaptive* method adjusts $h$ from step to step to keep the local error near a prescribed tolerance, taking large steps in placid regions and small steps where the solution is active. This requires, at each step, an estimate of the local error and a rule for changing $h$ in response.

## 16.2 Estimating the local error

The local error is estimated by computing the step two ways and comparing. One approach takes a single step of size $h$ and two steps of size $h/2$ and uses their difference; a more efficient approach uses an *embedded* pair of Runge–Kutta methods of orders $p$ and $p+1$ that share their function evaluations. The Runge–Kutta–Fehlberg method (RKF45) is the classic example: a fourth-order and a fifth-order formula built from the same six stages, whose difference

$$E = \left| y_{n+1}^{(p+1)} - y_{n+1}^{(p)} \right|$$

estimates the local error of the lower-order result at almost no extra cost.

## 16.3 Controlling the step size

Because the local error of an order-$p$ method scales as $h^{p+1}$, an estimate $E$ at step size $h$ predicts the step size that would meet a tolerance $\tau$. Setting the predicted error equal to $\tau$

gives the new step size

$$h_{\text{new}} = \theta\, h \left(\frac{\tau}{E}\right)^{1/(p+1)},$$

where $\theta \approx 0.9$ is a safety factor. If $E \le \tau$ the step is accepted and $h$ is enlarged for the next step; if $E > \tau$ the step is rejected and retried with the smaller $h_{\text{new}}$.

**Algorithm 16.1 (Adaptive step with error control).**

1. From $(t_n, y_n)$ and step $h$, compute the order-$p$ and order-$(p+1)$ estimates and the error $E$.
2. If $E \le \tau$, accept: set $t_{n+1} = t_n + h$, $y_{n+1}$ to the higher-order value.
3. Whether accepted or rejected, update $h \leftarrow \theta\, h(\tau/E)^{1/(p+1)}$, optionally limiting the growth and shrinkage factors.

Adaptive control lets a single integration handle solutions with both smooth and rapidly varying regions efficiently, and is the mechanism behind production differential-equation solvers.

**Example 16.1.** Suppose an order-four method ($p = 4$) with tolerance $\tau = 10^{-6}$ takes a step of size $h = 0.1$ and the embedded estimate returns $E = 5 \times 10^{-7}$. Since $E < \tau$, the step is accepted, and the next step is enlarged to $h_{\text{new}} = 0.9 \cdot 0.1 \cdot (10^{-6}/5 \times 10^{-7})^{1/5} \approx 0.103$. If that step then returns $E = 3 \times 10^{-6} > \tau$, it is rejected and retried with $h_{\text{new}} = 0.9 \cdot 0.103 \cdot (10^{-6}/3 \times 10^{-6})^{1/5} \approx 0.075$. The controller shrinks the step where the solution is active and lets it grow where the solution is smooth.

# Exercises

**Exercise 16.1.** Given an order-four method with local error estimate $E = 2 \times 10^{-6}$ at step $h = 0.1$ and tolerance $\tau = 10^{-6}$, compute the recommended new step size.

**Exercise 16.2.** Explain how step doubling (one step of $h$ versus two of $h/2$) estimates the local error, and how many function evaluations it costs.

**Exercise 16.3.** Why is an embedded Runge–Kutta pair more efficient than step doubling for error estimation?

**Exercise 16.4.** Describe what happens to the step size near a point where the solution suddenly changes rapidly.

**Exercise 16.5.** ⋆Derive the step-size formula $h_{\text{new}} = h(\tau/E)^{1/(p+1)}$ from the assumption that the local error behaves like $Ch^{p+1}$.

**Exercise 16.6.** For an order-four method with $E = 4 \times 10^{-6}$ at $h = 0.1$ and tolerance $\tau = 10^{-6}$, compute the recommended step size.

**Exercise 16.7.** ⋆In RKF45 the fifth-order value is often used to advance the solution; explain this local extrapolation and its effect on the error estimate.

# Chapter 17

# Systems and Higher-Order Equations

The methods of the preceding chapters were stated for a single first-order equation, but the same formulas apply without change to systems of equations and, through a simple reduction, to equations of higher order.

## 17.1 Systems of first-order equations

A system of $m$ first-order equations is written in vector form as

$$\mathbf{y}'(t) = \mathbf{f}(t, \mathbf{y}), \qquad \mathbf{y}(t_0) = \mathbf{y}_0,$$

with $\mathbf{y}(t) \in \mathbb{R}^m$ and $\mathbf{f} : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}^m$. Every method of Chapters 9–16 carries over verbatim with scalars replaced by vectors. Euler's method is $\mathbf{y}_{n+1} = \mathbf{y}_n + h\,\mathbf{f}(t_n, \mathbf{y}_n)$, and the classical Runge–Kutta method uses vector stages $\mathbf{k}_1, \dots, \mathbf{k}_4$ combined by the same weights. The convergence theory is unchanged, the Lipschitz condition being stated in a vector norm, $\|\mathbf{f}(t,\mathbf{u}) - \mathbf{f}(t,\mathbf{v})\| \le L\,\|\mathbf{u}-\mathbf{v}\|$, and the global error remaining $\mathcal{O}(h^p)$ for a method of order $p$.

## 17.2 Reduction of higher-order equations

An $m$-th order equation $y^{(m)} = g\big(t, y, y', \dots, y^{(m-1)}\big)$ is converted to a first-order system by naming the derivatives as new unknowns. Setting $u_1 = y$, $u_2 = y'$, $\dots$, $u_m = y^{(m-1)}$ gives

$$u_1' = u_2, \quad u_2' = u_3, \quad \dots, \quad u_{m-1}' = u_m, \quad u_m' = g(t, u_1, \dots, u_m),$$

a first-order system for $\mathbf{u} = (u_1, \dots, u_m)^{\mathsf{T}}$. The numerical methods apply to $\mathbf{u}$, and the first component of the result is the solution $y$.

**Example 17.1.** The damped oscillator $y'' + cy' + ky = 0$ becomes, with $u_1 = y$ and $u_2 = y'$, the linear system $\mathbf{u}' = \left(\begin{smallmatrix} 0 & 1 \\ -k & -c \end{smallmatrix}\right)\mathbf{u}$. The eigenvalues of the coefficient matrix decide whether the motion decays or oscillates, and a numerical method reproduces that behavior when $h$ keeps the products $h\lambda$ inside the region of absolute stability.

## 17.3 Numerical solution of a system

The implementation needs only treat the state as a column vector and the right-hand side as returning a column vector; the loop body is identical to the scalar case.

```
function [t, Y] = rk4_system(f, t0, y0, h, N)
% RK4_SYSTEM Classical RK4 for the system y' = f(t,y); y0 a column vector.
t = t0 + (0:N)'*h;
Y = zeros(N+1, numel(y0));
y = y0(:); Y(1,:) = y';
for n = 1:N
    k1 = f(t(n), y);
    k2 = f(t(n)+h/2, y + h/2*k1);
    k3 = f(t(n)+h/2, y + h/2*k2);
    k4 = f(t(n)+h, y + h*k3);
    y = y + h/6*(k1 + 2*k2 + 2*k3 + k4);
    Y(n+1,:) = y';
end
end
```

**Example 17.2.** The predator–prey system $x' = ax - bxy$, $y' = -cy + dxy$ is solved by writing $\mathbf{u} = (x, y)^{\mathsf{T}}$ with $\mathbf{f}(t, \mathbf{u}) = (ax - bxy,\ -cy + dxy)^{\mathsf{T}}$; the routine above produces the closed orbits the model predicts.

**Example 17.3.** The system $x' = y$, $y' = -x$ has the circular solution $(\cos t, -\sin t)$ from $(1, 0)$, on which $\|\mathbf{u}\|_2 \equiv 1$. Euler's method with $h = 0.1$ gives

| $n$ | $x_n$ | $y_n$ | $\|\mathbf{u}_n\|_2$ |
|---|---|---|---|
| 0 | 1.000 | 0.000 | 1.0000 |
| 1 | 1.000 | $-0.100$ | 1.0050 |
| 2 | 0.990 | $-0.200$ | 1.0100 |
| 3 | 0.970 | $-0.299$ | 1.0150 |

The norm grows by about 0.5% per step: Euler spirals slowly outward, injecting energy the true motion conserves. The coefficient matrix has eigenvalues $\pm i$, on the imaginary axis and

outside Euler's region of absolute stability, so no step size cures the drift—a higher-order or implicit method is needed for oscillatory systems.

## Exercises

**Exercise 17.1.** Convert $y''' - 2y'' + y' - y = 0$ into a first-order system of three equations.

**Exercise 17.2.** Write Euler's method in vector form for $x' = y$, $y' = -x$, and take two steps from $(x_0, y_0) = (1, 0)$ with $h = 0.1$.

**Exercise 17.3.** Reduce the pendulum equation $\theta'' + \sin\theta = 0$ to a first-order system and identify its vector field.

**Exercise 17.4.** Explain why the convergence theory of one-step methods extends to systems with no change beyond using a vector norm.

**Exercise 17.5.** $^\star$For $\mathbf{u}' = \mathbf{A}\mathbf{u}$, show that Euler's method gives $\mathbf{u}_n = (\mathbf{I} + h\mathbf{A})^n \mathbf{u}_0$ and is stable precisely when $\rho(\mathbf{I} + h\mathbf{A}) \le 1$.

# Chapter 18

# The Bisection Method

Root-finding solves a single nonlinear equation $f(x) = 0$. The bisection method is the most elementary root-finder: slow but utterly reliable, and the standard against which faster methods are measured.

## 18.1 The intermediate value theorem

If $f$ is continuous on $[a, b]$ and $f(a)$ and $f(b)$ have opposite signs, the intermediate value theorem guarantees a root in $(a, b)$. Bisection exploits this by repeatedly halving the bracket: the midpoint $c = (a + b)/2$ is computed, and the half on which $f$ changes sign—either $[a, c]$ or $[c, b]$—is kept as the new, smaller bracket. Each step encloses a root in an interval half as wide.

## 18.2 Convergence

After $n$ steps the bracket has width $(b - a)/2^n$, and taking its midpoint as the estimate gives the error bound

$$|x_n - x^*| \leq \frac{b - a}{2^{n+1}}.$$

The error is halved at every step regardless of $f$, so convergence is *linear* with rate $\frac{1}{2}$. To guarantee an error below a tolerance $\varepsilon$ requires

$$n \geq \log_2 \frac{b - a}{\varepsilon}$$

steps, a count known in advance—a property no faster method shares.

**Algorithm 18.1 (Bisection).**

1. Given $[a, b]$ with $f(a)f(b) < 0$ and a tolerance $\varepsilon$.
2. Repeat: set $c = (a + b)/2$; if $f(a)f(c) < 0$ set $b = c$, else set $a = c$.
3. Stop when $(b - a)/2 < \varepsilon$ and return $c$.

```
function c = bisection(f, a, b, tol)
% BISECTION Root of f in [a,b] with f(a)*f(b) < 0, to tolerance tol.
fa = f(a);
while (b - a)/2 > tol
    c = (a + b)/2;
    fc = f(c);
    if fa*fc < 0
        b = c;
    else
        a = c; fa = fc;
    end
end
c = (a + b)/2;
end
```

**Example 18.1.** For $f(x) = x^2 - 2$ on $[1, 2]$, bisection produces $1.5, 1.25, 1.375, \ldots$, converging to $\sqrt{2} \approx 1.41421$; reaching an error below $10^{-6}$ takes $\lceil \log_2(1/10^{-6}) \rceil = 20$ steps.

**Example 18.2.** For $f(x) = x^3 - x - 2$ on $[1, 2]$, with $f(1) = -2$ and $f(2) = 4$, bisection produces the brackets below, the width halving at every step toward the root $x^* \approx 1.5214$.

| $n$ | midpoint $c_n$ | $f(c_n)$ | bracket width |
|---|---|---|---|
| 1 | 1.5000 | $-0.125$ | 0.5 |
| 2 | 1.7500 | 1.609 | 0.25 |
| 3 | 1.6250 | 0.666 | 0.125 |
| 4 | 1.5625 | 0.252 | 0.0625 |
| 5 | 1.5313 | 0.060 | 0.03125 |
| 6 | 1.5156 | $-0.034$ | 0.015625 |

Each step gains a little more than one binary digit; matching the three-digit accuracy of one Newton step takes about ten bisection steps. Reliability is bought at the price of speed.

## Exercises

**Exercise 18.1.** Perform four steps of bisection on $f(x) = x^3 - x - 2$ starting from $[1, 2]$, reporting each bracket.

**Exercise 18.2.** How many bisection steps are needed to locate a root in $[0, 1]$ to within $10^{-8}$?

**Exercise 18.3.** Explain why bisection cannot find a root of $f(x) = x^2$ even though $x = 0$ is a root.

**Exercise 18.4.** Show that bisection converges linearly with rate $\frac{1}{2}$, with the error bound halving at each step.

**Exercise 18.5.** ⋆Modify the algorithm to also stop when $|f(c)|$ is below a tolerance, and discuss when each stopping criterion is preferable.

**Exercise 18.6.** Perform four bisection steps on $f(x) = x^3 - x - 2$ starting from $[1, 2]$.

**Exercise 18.7.** How many bisection steps locate a root in $[0, 1]$ to within $10^{-8}$?

**Exercise 18.8.** Explain why bisection cannot find the double root of $f(x) = x^2$.

**Exercise 18.9.** State the error bound after $n$ bisection steps and the number of steps to reach tolerance $\varepsilon$.

**Exercise 18.10.** Apply three bisection steps to $f(x) = \cos x - x$ on $[0, 1]$.

**Exercise 18.11.** ⋆Modify the bisection algorithm to stop when $|f(c)| < \delta$, and discuss when this criterion is preferable to a small bracket.

# Chapter 19

# Newton's Method

## 19.1 Derivation

Newton's method replaces $f$ near the current estimate by its tangent line and takes the root of the tangent as the next estimate. The tangent at $x_n$ is $f(x_n) + f'(x_n)(x - x_n)$, whose root is

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)},$$

the same formula that the Taylor expansion $0 = f(x^*) \approx f(x_n) + f'(x_n)(x^* - x_n)$ produces. Each step needs both $f$ and $f'$.

## 19.2 Quadratic convergence

**Theorem 19.1.** *If $f$ has a continuous second derivative, $x^*$ is a simple root ($f'(x^*) \neq 0$), and $x_0$ is sufficiently close to $x^*$, then Newton's method converges and*

$$|x_{n+1} - x^*| \leq C \, |x_n - x^*|^2$$

*for a constant $C = |f''(x^*)| \, / (2\,|f'(x^*)|)$.*

*Sketch.* Taylor expansion about $x_n$ gives $0 = f(x^*) = f(x_n) + f'(x_n)(x^* - x_n) + \frac{1}{2} f''(\eta)(x^* - x_n)^2$. Dividing by $f'(x_n)$ and using the iteration formula yields $x^* - x_{n+1} = -\frac{f''(\eta)}{2f'(x_n)}(x^* - x_n)^2$, which gives the bound near the root. □

The error is squared at each step, so the number of correct digits roughly doubles per iteration once the iteration is near the root—far faster than the linear convergence of bisection.

## 19.3 Failure modes

Newton's method is fast but not foolproof. It breaks down if $f'(x_n) = 0$, sending the tangent off to a horizontal line; it may diverge or cycle when $x_0$ is far from a root; and at a *multiple* root, where $f'(x^*) = 0$, the convergence degrades from quadratic to linear. The speed of Newton's method is local: a good starting point, often supplied by a few bisection steps, is essential.

```
function x = newton(f, df, x0, tol, maxit)
% NEWTON Root of f by Newton's method; df is the derivative f'.
x = x0;
for k = 1:maxit
    fx = f(x);
    if abs(fx) < tol, return; end
    x = x - fx / df(x);
end
end
```

**Example 19.2.** Applied to $f(x) = x^2 - a$, Newton's method becomes $x_{n+1} = \tfrac{1}{2}\big(x_n + a/x_n\big)$, the ancient iteration for $\sqrt{a}$. For $a = 2$ and $x_0 = 1.5$ it gives $1.41\overline{6}$, $1.414215\overline{5}$, …, the correct digits doubling each step.

**Example 19.3.** The doubling of correct digits is seen directly for $f(x) = x^3 - x - 2$, whose root is $x^* \approx 1.521380$. From $x_0 = 1.5$ Newton's method gives the iterates

| $n$ | $x_n$ | $\lvert x_n - x^*\rvert$ |
|---|---|---|
| 0 | 1.500000 | $2.1 \times 10^{-2}$ |
| 1 | 1.521739 | $3.6 \times 10^{-4}$ |
| 2 | 1.521380 | $3.6 \times 10^{-7}$ |
| 3 | 1.521380 | $< 10^{-9}$ |

The error falls from $10^{-2}$ to $10^{-4}$ to $10^{-7}$: the exponent roughly doubles at each step, the signature of quadratic convergence. Four iterations already reach machine accuracy.

## Exercises

**Exercise 19.1.** Apply Newton's method to $f(x) = x^3 - x - 2$ from $x_0 = 1.5$, computing $x_1$ and $x_2$.

**Exercise 19.2.** Derive the Newton iteration for $\sqrt{a}$ and for the reciprocal $1/a$ (solving $1/x - a = 0$ without division).

**Exercise 19.3.** State and explain the quadratic convergence estimate, identifying the constant $C$.

**Exercise 19.4.** Give an example of a function and starting point for which Newton's method diverges, and explain why.

**Exercise 19.5.** ⋆Show that for a root of multiplicity $m > 1$ Newton's method converges only linearly, with rate $1 - 1/m$.

**Exercise 19.6.** Derive the Newton iteration for the cube root of $a$ (solving $x^3 - a = 0$).

**Exercise 19.7.** Identify the constant $C$ in the quadratic convergence estimate for a simple root.

**Exercise 19.8.** Apply Newton's method to solve $x = \cos x$ from $x_0 = 1$, taking two steps.

**Exercise 19.9.** ⋆Derive a division-free Newton iteration for $1/\sqrt{a}$ (solving $x^{-2} - a = 0$), useful in hardware.

# Chapter 20

# The Secant Method

## 20.1 Derivation

Newton's method requires the derivative $f'$, which may be unavailable or costly. The *secant method* replaces the derivative by the finite-difference approximation from the two most recent points,

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}},$$

giving the iteration

$$x_{n+1} = x_n - f(x_n)\,\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}.$$

Geometrically, the next estimate is the root of the secant line through the two previous points rather than of the tangent. The method needs two starting values but only one new function evaluation per step, and no derivative.

## 20.2 Order of convergence

For a simple root the secant method converges *superlinearly*, with error satisfying

$$|x_{n+1} - x^*| \le C\ |x_n - x^*|^{\varphi}, \qquad \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618,$$

the golden ratio. The order lies between the linear convergence of bisection and the quadratic convergence of Newton. Because each step costs only one function evaluation, against the two values ($f$ and $f'$) Newton requires, the secant method is often more efficient than Newton per unit of work, even though its order is lower.

**Example 20.1.** For $f(x) = x^2 - 2$ with $x_0 = 1$, $x_1 = 2$, the secant method gives $x_2 = \frac{4}{3} \approx$

1.333, $x_3 \approx 1.400$, $x_4 \approx 1.41420$, approaching $\sqrt{2}$ superlinearly.

**Example 20.2.** On $f(x) = x^3 - x - 2$ with $x_0 = 1$, $x_1 = 2$ the secant method gives the iterates below, the error falling faster than linearly but without the clean squaring of Newton's method.

| $n$ | $x_n$ | $\lvert x_n - x^* \rvert$ |
|---|---|---|
| 0 | 1.000000 | $5.2 \times 10^{-1}$ |
| 1 | 2.000000 | $4.8 \times 10^{-1}$ |
| 2 | 1.333333 | $1.9 \times 10^{-1}$ |
| 3 | 1.462687 | $5.9 \times 10^{-2}$ |
| 4 | 1.531129 | $9.7 \times 10^{-3}$ |
| 5 | 1.520911 | $4.7 \times 10^{-4}$ |
| 6 | 1.521372 | $8 \times 10^{-6}$ |

The convergence is of order $\varphi \approx 1.618$: each error is roughly the previous one raised to that power, intermediate between bisection and Newton, and achieved with a single function evaluation per step.

# Exercises

**Exercise 20.1.** Apply the secant method to $f(x) = x^3 - x - 2$ with $x_0 = 1$, $x_1 = 2$, computing $x_2$ and $x_3$.

**Exercise 20.2.** Explain how the secant method approximates Newton's method, and what is gained and lost by the approximation.

**Exercise 20.3.** Compare the cost per step and the order of convergence of bisection, the secant method, and Newton's method.

**Exercise 20.4.** Show that the secant iteration can be written symmetrically in $x_n$ and $x_{n-1}$, and that it reduces to interpolating $f$ linearly through the two points.

**Exercise 20.5.** ⋆Assuming an error relation $\lvert e_{n+1} \rvert \approx K \lvert e_n \rvert \, \lvert e_{n-1} \rvert$, derive the convergence order $\varphi = (1 + \sqrt{5})/2$.

**Exercise 20.6.** Use the secant method to approximate $\sqrt{5}$ from $x_0 = 2$, $x_1 = 3$.

**Exercise 20.7.** Show that the secant step is the root of the line interpolating $f$ at the two previous points.

**Exercise 20.8.** Derive the convergence order $\varphi = (1 + \sqrt{5})/2$ from the relation $\lvert e_{n+1} \rvert \approx K \lvert e_n \rvert \, \lvert e_{n-1} \rvert$.

**Exercise 20.9.** ⋆Describe how the secant method can fail when $f(x_n) = f(x_{n-1})$, and a safeguard against it.

# Chapter 21

# Fixed-Point Iteration

## 21.1 Fixed points

Every root-finding problem can be recast as a *fixed-point* problem: rewrite $f(x) = 0$ as $x = g(x)$ for a suitable $g$, so that a root of $f$ is a fixed point of $g$. The associated iteration is

$$x_{n+1} = g(x_n),$$

which, when it converges, converges to a fixed point. Many methods, Newton's among them, are fixed-point iterations for a particular choice of $g$.

## 21.2 The contraction mapping theorem

**Theorem 21.1.** *Let $g$ map an interval $[a, b]$ into itself and satisfy $|g'(x)| \le k < 1$ there. Then $g$ has a unique fixed point $x^*$ in $[a, b]$, the iteration $x_{n+1} = g(x_n)$ converges to it from any starting point in $[a, b]$, and*

$$|x_n - x^*| \le \frac{k^n}{1-k} \, |x_1 - x_0| \, .$$

*Sketch.* The mean value theorem gives $|x_{n+1} - x^*| = |g(x_n) - g(x^*)| \le k \, |x_n - x^*|$, so the error contracts by the factor $k < 1$ at each step; iterating gives geometric convergence, and a contraction has at most one fixed point. □

The condition $|g'| < 1$ is decisive: the iteration converges when $g$ is a contraction and diverges when $|g'(x^*)| > 1$, however close the start.

## 21.3 Order of convergence

Near the fixed point $g(x) \approx g(x^*) + g'(x^*)(x - x^*)$, so the error obeys $e_{n+1} \approx g'(x^*)e_n$: convergence is *linear* with rate $|g'(x^*)|$ when this is nonzero, and faster when it vanishes. If $g'(x^*) = 0$ but $g''(x^*) \neq 0$ the convergence is quadratic. Newton's method is exactly the fixed-point iteration with $g(x) = x - f(x)/f'(x)$, for which a short computation gives $g'(x^*) = 0$ at a simple root—the fixed-point explanation of its quadratic convergence.

**Example 21.2.** To solve $x = \cos x$, the iteration $x_{n+1} = \cos x_n$ has $|g'(x)| = |\sin x| < 1$ near the fixed point, so it converges, from $x_0 = 1$, to $x^* \approx 0.739$. Rewriting the same root as $x = x - (x - \cos x)$ or other forms can give $|g'| > 1$ and a divergent iteration, showing that the choice of $g$ controls success.

**Example 21.3.** The iteration $x_{n+1} = \cos x_n$ from $x_0 = 1$ converges to $x^* \approx 0.739085$, but slowly, the iterates oscillating about the fixed point.

| $n$ | $x_n$ | $\lvert x_n - x^* \rvert$ |
|---|---|---|
| 1 | 0.540302 | $1.99 \times 10^{-1}$ |
| 2 | 0.857553 | $1.18 \times 10^{-1}$ |
| 4 | 0.793480 | $5.4 \times 10^{-2}$ |
| 6 | 0.763960 | $2.5 \times 10^{-2}$ |
| 8 | 0.750418 | $1.1 \times 10^{-2}$ |

The error shrinks by the roughly constant factor $|g'(x^*)| = |\sin x^*| \approx 0.674$ per step, so the convergence is linear: reaching six-digit accuracy takes more than thirty iterations, against the four Newton's method needs. The price of avoiding the derivative is speed.

## Exercises

**Exercise 21.1.** Show that $g(x) = \cos x$ is a contraction near its fixed point, and perform three steps of the iteration from $x_0 = 1$.

**Exercise 21.2.** For solving $x^2 - 2 = 0$, compare the iterations $x = 2/x$ and $x = \frac{1}{2}(x + 2/x)$, and explain why only the second converges.

**Exercise 21.3.** State the contraction mapping theorem and the error bound it provides.

**Exercise 21.4.** Show that Newton's method is the fixed-point iteration with $g(x) = x - f(x)/f'(x)$, and compute $g'(x^*)$ at a simple root.

**Exercise 21.5.** ⋆Prove that if $g'(x^*) = 0$ and $g''$ is continuous, then the fixed-point iteration converges quadratically near $x^*$.

**Exercise 21.6.** Construct a convergent fixed-point iteration for $x^2 - 3 = 0$ and identify its convergence rate.

**Exercise 21.7.** If $g'(x^*) = 0.7$, estimate how many iterations reduce the error by a factor of $10^3$.

**Exercise 21.8.** $^\star$Prove that if $g'(x^*) = 0$ and $g''$ is continuous, the fixed-point iteration converges quadratically near $x^*$.

# Solutions to Selected Exercises

**Chapter 1.**

**1.2.** If $\mathbf{A}, \mathbf{B}$ are upper triangular then $(\mathbf{AB})_{ij} = \sum_k a_{ik} b_{kj}$; a term is nonzero only when $i \le k$ and $k \le j$, which requires $i \le j$, so the product is upper triangular. For $i = j$ the only surviving term is $k = i$, giving $(\mathbf{AB})_{ii} = a_{ii} b_{ii}$.

**1.3.** From $\mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$, transposing gives $(\mathbf{A}^{-1})^{\mathsf{T}} \mathbf{A}^{\mathsf{T}} = \mathbf{I}$, so $\mathbf{A}^{\mathsf{T}}$ is nonsingular with inverse $(\mathbf{A}^{-1})^{\mathsf{T}}$.

**1.5.** $\mathbf{A} = \operatorname{diag}(d_1, \dots, d_n)$ is nonsingular iff every $d_i \neq 0$, in which case $\mathbf{A}^{-1} = \operatorname{diag}(1/d_1, \dots, 1/d_n)$ and $\det \mathbf{A} = d_1 \cdots d_n$.

**Chapter 2.**

**2.2.** For $\mathbf{A} = \left(\begin{smallmatrix} 4 & 3 \\ 6 & 3 \end{smallmatrix}\right)$ the multiplier is $m_{21} = 6/4 = \tfrac{3}{2}$, giving $\mathbf{U} = \left(\begin{smallmatrix} 4 & 3 \\ 0 & -\tfrac{3}{2} \end{smallmatrix}\right)$ and $\mathbf{L} = \left(\begin{smallmatrix} 1 & 0 \\ \tfrac{3}{2} & 1 \end{smallmatrix}\right)$. Solving $\mathbf{L}\mathbf{y} = (10, 12)^{\mathsf{T}}$ gives $y_1 = 10$, $y_2 = 12 - 15 = -3$; then $\mathbf{U}\mathbf{x} = \mathbf{y}$ gives $x_2 = 2$, $x_1 = (10 - 6)/4 = 1$.

**2.4.** Since $\mathbf{A} = \mathbf{L}\mathbf{U}$ with $\mathbf{L}$ unit lower triangular, $\det \mathbf{L} = 1$ and $\det \mathbf{A} = \det \mathbf{U} = u_{11} \cdots u_{nn}$ because the determinant of a triangular matrix is the product of its diagonal entries.

**Chapter 3.**

**3.1.** With $\varepsilon = 10^{-3}$ and four-digit arithmetic, naive elimination gives $m_{21} = 1000$, the second equation $(1 - 1000)x_2 = 2 - 1000$ rounds to $-999\, x_2 = -998$, so $x_2 = 0.9990$ and then $x_1 = (1 - 0.9990)/0.001 = 1.000$; here the loss is mild. As $\varepsilon$ shrinks toward $\epsilon_{\text{mach}}$ the same computation loses all accuracy, whereas partial pivoting (interchanging the equations) keeps the multiplier equal to $\varepsilon$ and remains accurate.

**3.3.** At stage $k$ the pivot $a_{kk}$ is chosen to maximize $|a_{kk}|$ among the entries on and below the diagonal in column $k$, so $|a_{ik}| \le |a_{kk}|$ for $i > k$ and hence $|m_{ik}| = |a_{ik}/a_{kk}| \le 1$. Without pivoting $a_{kk}$ may be smaller than $a_{ik}$, making the multiplier exceed 1.

**Chapter 4.**

**4.1.** $\|\mathbf{x}\|_1 = 3 + 4 + 12 = 19$, $\|\mathbf{x}\|_2 = \sqrt{9 + 16 + 144} = 13$, $\|\mathbf{x}\|_\infty = 12$; indeed $12 \le 13 \le 19$.

**4.3.** Column sums of $\left(\begin{smallmatrix} 1 & -2 \\ 3 & 4 \end{smallmatrix}\right)$ are 4 and 6, so $\|\mathbf{A}\|_1 = 6$; row sums are 3 and 7, so $\|\mathbf{A}\|_\infty = 7$; and $\|\mathbf{A}\|_F = \sqrt{1 + 4 + 9 + 16} = \sqrt{30}$.

**4.5.** If $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$ with $\mathbf{v} \neq \mathbf{0}$, then $|\lambda| \, \|\mathbf{v}\| = \|\mathbf{A}\mathbf{v}\| \le \|\mathbf{A}\| \, \|\mathbf{v}\|$, so $|\lambda| \le \|\mathbf{A}\|$ for every eigenvalue and hence $\rho(\mathbf{A}) \le \|\mathbf{A}\|$. For $\mathbf{A} = \left(\begin{smallmatrix} 0 & 2 \\ 0 & 0 \end{smallmatrix}\right)$, $\rho(\mathbf{A}) = 0$ while $\|\mathbf{A}\|_2 = 2$, so the inequality is strict.

**Chapter 5.**

**5.1.** $p(\lambda) = (3 - \lambda)^2 - 1 = (\lambda - 2)(\lambda - 4)$, so the eigenvalues are 2 (eigenvector $(1, 1)^\mathsf{T}$) and 4 (eigenvector $(1, -1)^\mathsf{T}$). Their sum 6 is the trace and product 8 is the determinant.

**5.3.** The Gershgorin disks are centered at $5, 6, 7$ with radii $1, 2, 2$, so all eigenvalues lie in $[3, 9]$ on the real axis; since none of the disks reaches 0, the matrix is nonsingular.

**5.5.** $\mathbf{y} = \mathbf{A}\mathbf{x}_0 = (2, 1)^\mathsf{T}$, so $\mu_1 = 2$ and $\mathbf{x}_1 = (1, \frac{1}{2})^\mathsf{T}$; then $\mathbf{y} = \mathbf{A}\mathbf{x}_1 = (2.5, 2)^\mathsf{T}$, so $\mu_2 = 2.5$. The estimates $2, 2.5$ climb toward the dominant eigenvalue 3.

**Chapter 6.**

**6.1.** $\mathbf{A}^{-1} = \left(\begin{smallmatrix} -2 & 1 \\ \frac{3}{2} & -\frac{1}{2} \end{smallmatrix}\right)$, with $\|\mathbf{A}\|_\infty = 7$ (row sum $3 + 4$) and $\|\mathbf{A}^{-1}\|_\infty = 3$ (row sum $2 + 1$), so $\operatorname{cond}_\infty(\mathbf{A}) = 21$.

**6.3.** For any induced norm, $1 = \|\mathbf{I}\| = \|\mathbf{A}\mathbf{A}^{-1}\| \le \|\mathbf{A}\| \, \|\mathbf{A}^{-1}\| = \operatorname{cond}(\mathbf{A})$. For scaling, $\operatorname{cond}(c\mathbf{A}) = \|c\mathbf{A}\| \, \|(c\mathbf{A})^{-1}\| = |c| \, \|\mathbf{A}\| \cdot |c|^{-1} \, \|\mathbf{A}^{-1}\| = \operatorname{cond}(\mathbf{A})$.

**6.5.** The bound $\|\mathbf{x} - \hat{\mathbf{x}}\| \, / \, \|\mathbf{x}\| \le \operatorname{cond}(\mathbf{A}) \, \|\mathbf{r}\| \, / \, \|\mathbf{b}\|$ gives at most $10^8 \cdot 10^{-15} = 10^{-7}$, so about seven correct digits are guaranteed; the small residual certifies a small *backward* error, but nothing sharper about the forward error can be concluded from the residual alone.

**6.6.** An orthogonal $\mathbf{Q}$ has $\|\mathbf{Q}\|_2 = 1$ and $\mathbf{Q}^{-1} = \mathbf{Q}^\mathsf{T}$, also orthogonal with $\left\|\mathbf{Q}^\mathsf{T}\right\|_2 = 1$, so $\operatorname{cond}_2(\mathbf{Q}) = 1$. Solving a system with an orthogonal matrix neither amplifies nor damps relative errors—the reason orthogonal transformations are the basis of stable algorithms.

**Chapter 7.**

**7.1.** The Lagrange combination simplifies to $p(x) = x^2 + 1$, which indeed gives $2, 1, 5$ at $x = -1, 0, 2$.

**7.4.** The constant function 1 has degree $0 \le n$, so by uniqueness it is its own interpolant; writing it in Lagrange form gives $1 = \sum_i 1 \cdot L_i(x) = \sum_i L_i(x)$.

**Chapter 8.**

**8.1.** $f_y = 2y$, so $|f_y| \le 4$ on $|y| \le 2$ and $L = 4$ serves.

**8.4.** The bound gives $|y(1) - \tilde{y}(1)| \le \mathrm{e}^2 \cdot 10^{-3} \approx 7.4 \times 10^{-3}$.

**Chapter 9.**

**9.1.** $y_1 = 0.75$, $y_2 = 0.625$, $y_3 = 0.59375$, $y_4 \approx 0.6328$.

**9.4.** $y_{n+1} = (1-h)y_n$, so $y_n = (1-h)^n$: it decays for $0 < h < 1$, oscillates while decaying for $1 < h < 2$, and grows in magnitude for $h > 2$.

**Chapter 10.**

**10.2.** With $y'' = 1 + t + y$, the order-two Taylor method gives $y_1 = 1.24$ and $y_2 \approx 1.5768$.

**10.4.** Keeping only the term $hf$ in the Taylor formula is exactly $y_{n+1} = y_n + hf(t_n, y_n)$, Euler's method.

**Chapter 11.**

**11.2.** $a+b = \frac{1}{2} + \frac{1}{2} = 1$ and $bc = \frac{1}{2} \cdot 1 = \frac{1}{2}$, so Heun's method meets both order-two conditions.

**11.3.** The stages are $k_1 = 1$, $k_2 = 1.25$, $k_3 = 1.3125$, $k_4 = 1.65625$, giving $y_1 \approx 1.6484$ against $\mathrm{e}^{0.5} \approx 1.64872$.

**Chapter 12.**

**12.1.** Euler is $\mathcal{O}(h)$ (error halves), the order-two methods $\mathcal{O}(h^2)$ (error quarters), and RK4 is $\mathcal{O}(h^4)$ (error drops by 16).

**12.4.** Minimizing $C_1 h + C_2 u/h$ gives $h_{\text{opt}} = \sqrt{C_2 u / C_1}$, of order $u^{1/2}$, with minimal error of order $\sqrt{u}$.

**Chapter 13.**

**13.1.** With the Euler start $y_1 = 1.1$, the two-step Adams–Bashforth method gives $y_2 = 1.215$ and $y_3 \approx 1.3423$.

**13.4.** Solving $y_{n+1} = y_n + \frac{h}{2}(\lambda y_{n+1} + \lambda y_n)$ gives $y_{n+1} = y_n(1 + h\lambda/2)/(1 - h\lambda/2)$.

**Chapter 14.**

**14.2.** A PECE step uses one evaluation at the predicted value and one at the corrected value; an exact implicit solve would instead require iterating to convergence.

**14.3.** The corrector map for $y' = \lambda y$ has multiplier $h\lambda/2$, so the iteration converges when $|h\lambda| < 2$.

**Chapter 15.**

**15.1.** For AB2, $\rho(\xi) = \xi^2 - \xi$ and $\sigma(\xi) = \frac{3}{2}\xi - \frac{1}{2}$; then $\rho(1) = 0$ and $\rho'(1) = 1 = \sigma(1)$, so the method is consistent.

**15.5.** Backward Euler gives $y_{n+1} = y_n/(1 - h\lambda)$, and $|1 - h\lambda| \geq 1$ for every $h\lambda$ with $\mathrm{Re}(h\lambda) \leq 0$, so $|y_{n+1}| \leq |y_n|$: the method is A-stable.

**Chapter 16.**

**16.1.** $h_{\text{new}} = 0.9 \cdot 0.1 \cdot (1/2)^{1/5} \approx 0.0784$.

**16.5.** Setting $Ch_{\text{new}}^{p+1} = \tau$ and dividing by $E = Ch^{p+1}$ gives $h_{\text{new}} = h(\tau/E)^{1/(p+1)}$.

**Chapter 17.**

**17.1.** The brackets are $[1.5, 2]$, $[1.5, 1.75]$, $[1.5, 1.625]$, $[1.5, 1.5625]$, closing on the root $\approx 1.5214$.

**17.2.** $n \geq \log_2(1/10^{-8}) = 8 \log_2 10 \approx 26.6$, so 27 steps.

**Chapter 18.**

**18.1.** From $x_0 = 1.5$, $x_1 \approx 1.52174$ and $x_2 \approx 1.52138$, the correct digits doubling.

**18.2.** For $\sqrt{a}$, $x_{n+1} = \frac{1}{2}(x_n + a/x_n)$; for $1/a$, solving $1/x - a = 0$ gives $x_{n+1} = x_n(2 - ax_n)$, which uses no division.

**Chapter 19.**

**19.1.** From $x_0 = 1$, $x_1 = 2$: $x_2 = \frac{4}{3} \approx 1.333$ and $x_3 \approx 1.463$, approaching $\sqrt[3]{}$-type root $\approx 1.5214$.

**19.3.** Bisection has order 1 with one evaluation per step and always converges; the secant method has order $\approx 1.618$ with one evaluation per step; Newton has order 2 but costs two values ($f$ and $f'$) per step.

**Chapter 20.**

**20.1.** $|g'(x)| = |\sin x| < 1$ near $x^* \approx 0.739$; from $x_0 = 1$ the iterates are $0.5403$, $0.8576$, $0.6543, \ldots$, closing on $0.739$.

**20.2.** For $x = 2/x$, $g'(\sqrt{2}) = -1$, so the iteration fails to converge; for $x = \frac{1}{2}(x + 2/x)$, $g'(\sqrt{2}) = 0$, giving quadratic convergence.

**20.4.** For $g(x) = x - f(x)/f'(x)$, $g'(x) = f(x)f''(x)/f'(x)^2$, which vanishes at a simple root since $f(x^*) = 0$—hence quadratic convergence.