TMA4215

Numerical Mathematics

Compendium for Numerical Mathematics

Author

Trym Sæther

Trym Sæther

Email: trym.saether@ntnu.no

Semester

Spring 2025

Last updated: December 2, 2025

Norwegian University of Science and Technology

Department of Mathematical Sciences

Contents

Part I

Introduction to Numerical Mathematics

Chapter 1

Preliminaries

1.1 Linear Algebra

1.1.1 Vectors and Matrices

Operations with Matrices

Let $A = (a_{ij})$ and $B = (b_{ij})$ be matrices in $\mathbb{K}^{m\times n}$, where $\mathbb{K}$ is a field (e.g., $\mathbb{R}$ or $\mathbb{C}$). We say $A = B$ if and only if $a_{ij} = b_{ij}$ for all $i = 1, \dots, m$ and $j = 1, \dots, n$.

Definition 1.1: Matrix Addition and Scalar Multiplication

- **Matrix sum:** $(A+B)_{ij} = a_{ij} + b_{ij}$ for all i, j. The additive identity is the *zero matrix* 0, with all entries zero.
- **Scalar multiplication:** For $\lambda \in \mathbb{K}$, $(\lambda A)_{ij} = \lambda a_{ij}$.

Example 1. Matrix Addition and Scalar Multiplication

Let $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $B = \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix}$, and $\lambda = 2$.

$$A + B = \begin{pmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 10 & 12 \end{pmatrix}, \quad 2A = \begin{pmatrix} 2\cdot 1 & 2\cdot 2 \\ 2\cdot 3 & 2\cdot 4 \end{pmatrix} = \begin{pmatrix} 2 & 4 \\ 6 & 8 \end{pmatrix}$$

Definition 1.2: Matrix Multiplication

Let $A \in \mathbb{K}^{m\times p}$ and $B \in \mathbb{K}^{p\times n}$. The *matrix product* $C = AB \in \mathbb{K}^{m\times n}$ is defined by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}, \quad \text{for } i = 1, \dots, m,\ j = 1, \dots, n.$$

Example 2. Matrix Multiplication

Let $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$, $B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$.

$$AB = \begin{pmatrix} 1\cdot 0 + 2\cdot 1 & 1\cdot 1 + 2\cdot 0 \\ 3\cdot 0 + 4\cdot 1 & 3\cdot 1 + 4\cdot 0 \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 4 & 3 \end{pmatrix}$$

Property 1: Matrix Operations

- Matrix addition is commutative and associative.
- Matrix multiplication is associative and distributive over addition, but **not** commutative in general: $AB \neq BA$.
- The *identity matrix* $I_n = (\delta_{ij})$ (where δ_{ij} is the Kronecker delta) satisfies $AI_n = I_nA = A$ for all $A \in \mathbb{K}^{n\times n}$.
- A *diagonal matrix* is $D = \mathrm{diag}(d_1, \dots, d_n)$, i.e., $D_{ij} = d_i$ if $i = j$, 0 otherwise.
- For $A \in \mathbb{K}^{n\times n}$ and $p \in \mathbb{N}$, A^p denotes the p-fold product of A with itself, with $A^0 = I_n$.

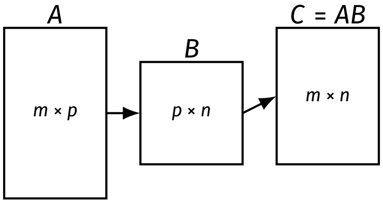


Figure 1.1: Matrix multiplication: A ($m \times p$) times B ($p \times n$) yields C ($m \times n$).

Remark 1. Commutativity and Special Matrices

Square matrices $A, B \in \mathbb{K}^{n\times n}$ are called *commutative* if $AB = BA$. In general, matrix multiplication is not commutative.

Elementary Row Operations and Matrices

Elementary row operations on $A \in \mathbb{K}^{m\times n}$ can be represented as multiplication by special matrices on the left:

- **Scaling row i by $\alpha \neq 0$:** Pre-multiply by $D = \mathrm{diag}(1, \dots, 1, \alpha, 1, \dots, 1)$, where α is in the i-th position.
- **Swapping rows i and j:** Pre-multiply by the *elementary permutation matrix* $P_{(i,j)}$, which is the identity with rows i and j swapped.
- **Adding α times row j to row i:** Pre-multiply by $E = I + \alpha E_{ij}$, where E_{ij} is the matrix with 1 in position (i, j) and 0 elsewhere.

Example 3. Permutation Matrix

or $n = 3$, swapping rows 1 and 2:

$$P_{(1,2)} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Special Types of Matrices

Definition 1.3: Special Matrices

Let $A \in \mathbb{R}^{n \times n}$ or $\mathbb{C}^{n \times n}$.
Identity matrix: $I_n = (\delta_{ij})$, where δ_{ij} is the Kronecker delta.
Diagonal matrix: $D = \operatorname{diag}(d_1, \dots, d_n)$, all off-diagonal entries are zero.
Symmetric: $A = A^\top$.
Antisymmetric (skew-symmetric): $A = -A^\top$.
Orthogonal: $A^\top A = AA^\top = I_n$ (so $A^{-1} = A^\top$).
Permutation matrix: obtained by permuting the rows of I_n.
Hermitian: $A = A^H$, where $A^H = \overline{A}^\top$.
Unitary: $A^H A = AA^H = I_n$ (so $A^{-1} = A^H$).
Normal: $A^H A = AA^H$ (complex) or $A^\top A = AA^\top$ (real matrices).

Example 4. Orthogonal and Unitary Matrices

- $Q = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ is orthogonal: $Q^\top Q = I$.
- $U = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$ is unitary: $U^H U = I$.

1.1.2 Matrix Inverse and Transpose

Definition 1.4: Inverse of a Matrix

A square matrix $A \in \mathbb{K}^{n \times n}$ is said to be *invertible* (or *nonsingular*) if there exists a matrix $A^{-1} \in \mathbb{K}^{n \times n}$ such that

$$AA^{-1} = A^{-1}A = I_n,$$

where I_n is the $n \times n$ identity matrix. The matrix A^{-1} is called the *inverse* of A.

Property 2: Invertibility and Linear Independence

A matrix $A \in \mathbb{K}^{n \times n}$ is invertible if and only if its columns (or, equivalently, its rows) form a linearly independent set in $\mathbb{K}^n$. Equivalently, $\det(A) \neq 0$.

Remark 2. Inverse of a Product
If $A, B \in \mathbb{K}^{n \times n}$ are invertible, then

$$(AB)^{-1} = B^{-1}A^{-1}.$$

More generally, for invertible matrices $A_1, A_2, \dots, A_k$,

$$(A_1 A_2 \cdots A_k)^{-1} = A_k^{-1} \cdots A_2^{-1} A_1^{-1}.$$

Definition 1.5: Transpose and Conjugate Transpose

Let $A = (a_{ij}) \in \mathbb{K}^{m\times n}$.

- The *transpose* of A is the matrix $A^\top \in \mathbb{K}^{n\times m}$ defined by

$$(A^\top)_{ij} = a_{ji}.$$

- If $\mathbb{K} = \mathbb{C}$, the *conjugate transpose* (or *adjoint*) is $A^H = \overline{A}^\top$, i.e.,

$$(A^H)_{ij} = \overline{a_{ji}}.$$

Property 3: Transpose Properties

Let $A \in \mathbb{K}^{m\times n}$, $B \in \mathbb{K}^{n\times p}$, and $\lambda \in \mathbb{K}$.

- $(A^\top)^\top = A$
- $(A + B)^\top = A^\top + B^\top$
- $(AB)^\top = B^\top A^\top$
- $(\lambda A)^\top = \lambda A^\top$
- If A invertible, $(A^\top)^{-1} = (A^{-1})^\top$

Analogous properties hold for A^H.

Example 5. Symmetric, Skew-Symmetric, Hermitian, Unitary

- $A = \begin{pmatrix} 2 & 3 \\ 3 & 2 \end{pmatrix}$ is symmetric.
- $B = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}$ is skew-symmetric.
- $C = \begin{pmatrix} 2 & i \\ -i & 3 \end{pmatrix}$ is Hermitian.
- $U = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & i \\ i & 1 \end{pmatrix}$ is unitary.

Remark 3. Diagonal Entries of Hermitian Matrices

If A is Hermitian, then $a_{ii} \in \mathbb{R}$ for all i.

Algebraic and Geometric Multiplicity

Definition 1.6: Algebraic and Geometric Multiplicity

Let $A \in \mathbb{C}^{n\times n}$ and λ be an eigenvalue of A.

- The *algebraic multiplicity* of λ, denoted $m_a(\lambda)$, is its multiplicity as a root of the characteristic polynomial $\det(A - \lambda I) = 0$.
- The *geometric multiplicity* of λ, denoted $m_g(\lambda)$, is the dimension of the eigenspace $\ker(A - \lambda I)$, i.e., the number of linearly independent eigenvectors associated with λ.
- Always $1 \le m_g(\lambda) \le m_a(\lambda)$.

Example 6. Algebraic vs. Geometric Multiplicity

Consider

$$A = \begin{pmatrix} 4 & 1 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 2 \end{pmatrix}.$$

The characteristic polynomial is

$$p(\lambda) = (4-\lambda)^2(2-\lambda)^1 = 0,$$

so $\lambda = 4$ has algebraic multiplicity 2, $\lambda = 2$ has algebraic multiplicity 1.
For $\lambda = 4$:

$$A - 4I = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -2 \end{pmatrix}$$

The eigenspace is all vectors of the form $\mathbf{v} = (v_1, 0, 0)^\top$, so the geometric multiplicity is 1.
For $\lambda = 2$:

$$A - 2I = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

The eigenspace is all vectors of the form $(0, 0, v_3)^\top$, so the geometric multiplicity is 1.

- $\lambda = 4$: $m_a = 2$, $m_g = 1$
- $\lambda = 2$: $m_a = 1$, $m_g = 1$

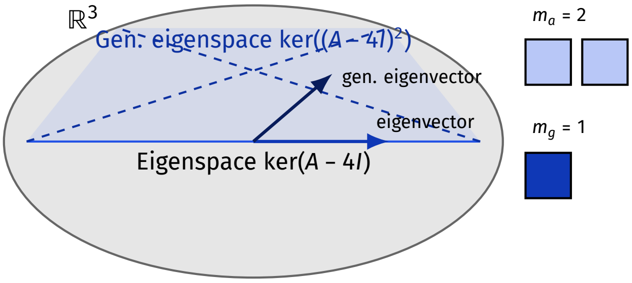


Figure 1.2: Algebraic multiplicity m_a (number of boxes) vs. geometric multiplicity m_g (number of independent eigenvectors) for $\lambda = 4$. The eigenspace is a line, the generalized eigenspace is a plane.

Remark 4. Interpretation

The algebraic multiplicity counts how many times an eigenvalue appears as a root; the geometric multiplicity counts the number of independent directions (eigenvectors) for that eigenvalue. If $m_g(\lambda) < m_a(\lambda)$, the matrix is not diagonalizable and the Jordan block for λ will be larger than 1 × 1.

Fundamental Subspaces

Definition 1.7: Column Space (Range)

The *column space* (or *range*) of a matrix $A \in \mathbb{R}^{m\times n}$ is the set of all linear combinations of its columns:

$$\text{range}(A) = \{\mathbf{y} \in \mathbb{R}^m : \mathbf{y} = A\mathbf{x} \text{ for some } \mathbf{x} \in \mathbb{R}^n\}.$$

The dimension of the column space is the *rank* of A:

$$\text{rank}(A) = \dim(\text{range}(A)).$$

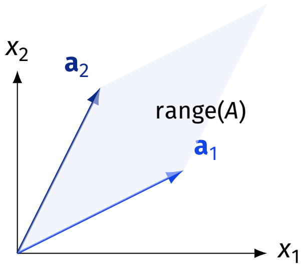


Figure 1.3: The column space is the span of the columns of A.

Definition 1.8: Row Space

The *row space* of $A \in \mathbb{R}^{m\times n}$ is the subspace of $\mathbb{R}^n$ spanned by the rows of A:

$$\text{row}(A) = \{\mathbf{z} \in \mathbb{R}^n : \mathbf{z} = \mathbf{y}A \text{ for some } \mathbf{y} \in \mathbb{R}^m\}.$$

Equivalently, the row space of A is the column space of A^T:

$$\text{row}(A) = \text{col}(A^T).$$

The dimension of the row space is also the rank of A:

$$\text{rank}(A) = \dim(\text{row}(A)).$$

Definition 1.9: Rank

The *rank* of a matrix $A \in \mathbb{R}^{m\times n}$ is the maximum number of linearly independent columns (or rows) of A:

$$\text{rank}(A) = \dim(\text{col}(A)) = \dim(\text{row}(A)).$$

The rank measures the dimension of the space spanned by the columns (or rows) of A, indicating the amount of independent information in the matrix.

Definition 1.10: Null Space (Kernel)

The *null space* (or *kernel*) of $A \in \mathbb{R}^{m\times n}$ is the set of all vectors $\mathbf{x} \in \mathbb{R}^n$ such that $A\mathbf{x} = \mathbf{0}$:

$$\ker(A) = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{0}\}.$$

The dimension of the null space is called the *nullity* of A:

$$\text{nullity}(A) = \dim(\ker(A)).$$

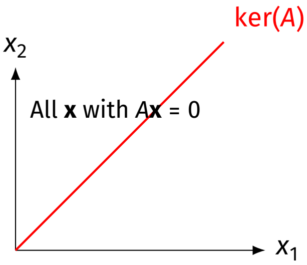


Figure 1.4: The null space consists of all solutions to $A\mathbf{x} = 0$.

Theorem 1.11: Fundamental Theorem of Linear Algebra

For any $A \in \mathbb{R}^{m\times n}$:

- The column space and null space are subspaces of $\mathbb{R}^m$ and $\mathbb{R}^n$, respectively.
- The row space and left null space (null space of A^T) are subspaces of $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively.
- **Rank-nullity theorem:**
$$\operatorname{rank}(A) + \operatorname{nullity}(A) = n.$$

Subspace	**Definition**	**Dimension**
Column space	$\{\mathbf{y} = A\mathbf{x}\}$	$\operatorname{rank}(A)$
Row space	$\{\mathbf{z} = \mathbf{y}A\}$	$\operatorname{rank}(A)$
Null space	$\{\mathbf{x} : A\mathbf{x} = 0\}$	$\operatorname{nullity}(A)$
Left null space	$\{\mathbf{y} : \mathbf{y}A = 0\}$	$m - \operatorname{rank}(A)$

Table 1.1: Summary of fundamental subspaces of a matrix.

1.1.3 Eigenvalues and Eigenvectors

Definition 1.12: Eigenvalues and Eigenvectors

Let $A \in \mathbb{R}^{n\times n}$. A scalar $\lambda \in \mathbb{C}$ is an *eigenvalue* of A if there exists a nonzero vector $\mathbf{v} \in \mathbb{C}^n$ such that

$$A\mathbf{v} = \lambda\mathbf{v}.$$

The vector $\mathbf{v}$ is called an *eigenvector* of A corresponding to λ.

- The set of all eigenvalues of A is called the *spectrum* of A, denoted $\sigma(A)$.
- The equation $\det(A - \lambda I) = 0$ is called the *characteristic equation*; its roots are the eigenvalues.
- The *eigenspace* for λ is $\ker(A - \lambda I)$, the set of all eigenvectors (plus $\mathbf{0}$) for λ.
- The *algebraic multiplicity* of λ is its multiplicity as a root of the characteristic polynomial.
- The *geometric multiplicity* of λ is $\dim\ker(A - \lambda I)$.

Remark 5. Geometric Interpretation

An eigenvector $\mathbf{v}$ is a direction that is only stretched or shrunk (by λ) by A, not rotated. If A is real but has complex eigenvalues, the action involves rotation and scaling in a plane.

Example 7. Eigenvalues and Eigenvectors of a 2×2 Matrix

Let

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

The characteristic polynomial is

$$\det(A - \lambda I) = (2 - \lambda)^2 - 1 = \lambda^2 - 4\lambda + 3 = (\lambda - 3)(\lambda - 1).$$

Thus, the eigenvalues are $\lambda_1 = 3$ and $\lambda_2 = 1$. For $\lambda_1 = 3$:

$$(A - 3I)\mathbf{v} = 0 \implies \begin{pmatrix} -1 & 1 \\ 1 & -1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0,$$

so $v_1 = v_2$. Any nonzero multiple of $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$ is an eigenvector for $\lambda = 3$. For $\lambda_2 = 1$:

$$(A - I)\mathbf{v} = 0 \implies \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 0,$$

so $v_1 = -v_2$. Any nonzero multiple of $\begin{pmatrix} 1 \\ -1 \end{pmatrix}$ is an eigenvector for $\lambda = 1$.

Remark 6. Diagonalization

If A has n linearly independent eigenvectors, then A is diagonalizable: $A = PDP^{-1}$, where D is diagonal with the eigenvalues of A and P has the eigenvectors as columns.

1.1.4 Jordan Canonical Form

Definition 1.13: Jordan Canonical Form

Let $A \in \mathbb{C}^{n \times n}$. The *Jordan canonical form* (or *Jordan normal form*) of A is a block-diagonal matrix J such that

$$A = PJP^{-1}$$

for some invertible matrix P, where J is composed of *Jordan blocks* along the diagonal. Each Jordan block $J_k(\lambda)$ associated with eigenvalue λ is of the form

$$J_k(\lambda) = \begin{pmatrix} \lambda & 1 & & & 0 \\ 0 & \lambda & 1 & & \\ \vdots & \ddots & \ddots & \ddots & \\ 0 & \cdots & 0 & \lambda & 1 \\ 0 & \cdots & \cdots & 0 & \lambda \end{pmatrix} \in \mathbb{C}^{k \times k}$$

where λ is on the diagonal, 1's are on the superdiagonal, and 0 elsewhere.

Existence and Uniqueness of Jordan Form

For any square matrix $A \in \mathbb{C}^{n \times n}$, there exists an invertible matrix P such that

$$A = PJP^{-1},$$

where J is in Jordan canonical form. The Jordan form J is unique up to the ordering of its Jordan blocks; that is, the sizes and number of blocks associated with each eigenvalue are uniquely determined by A.

The Jordan canonical form generalizes diagonalization: A is diagonalizable if and only if all Jordan blocks are 1×1. Larger blocks indicate the presence of generalized eigenvectors and the failure of diagonalizability.

Remark 7. Intuition and Geometric Meaning

The Jordan form captures the structure of A in terms of its eigenvalues and generalized eigenvectors. Each Jordan block corresponds to an eigenvalue λ and describes how A acts on the space

spanned by its eigenvectors and generalized eigenvectors.

- If A is diagonalizable, it stretches vectors along independent directions (eigenvectors).
- If not, the superdiagonal 1's in a Jordan block represent a ”shear” in addition to scaling.
- The size of the Jordan blocks (the number of 1's on the superdiagonal) shows how many generalized eigenvectors are needed to fully describe the action of A.

Example 8. Jordan Block Chain of Generalized Eigenvectors

For a Jordan block of size k for eigenvalue λ, there is a chain $\{\mathbf{v}_1, \dots, \mathbf{v}_k\}$ with

$$(A - \lambda I)\mathbf{v}_1 = 0, \quad (A - \lambda I)\mathbf{v}_j = \mathbf{v}_{j-1} \ (j \geq 2).$$

Here, $\mathbf{v}_1$ is an eigenvector, and each $\mathbf{v}_j$ ($j > 1$) is a generalized eigenvector. A acts by scaling on $\mathbf{v}_1$, and by scaling plus a shift (shear) along the chain for $\mathbf{v}_j$.

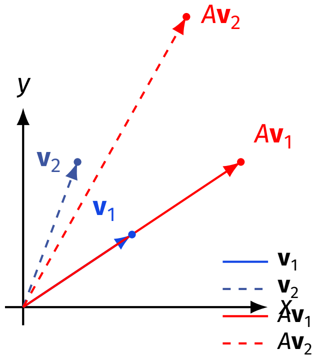


Figure 1.5: Jordan block: $A\mathbf{v}_1 = \lambda\mathbf{v}_1$, $A\mathbf{v}_2 = \lambda\mathbf{v}_2 + \mathbf{v}_1$.

Generalized Eigenvectors and Chains

A *generalized eigenvector* $\mathbf{v}$ of A for eigenvalue λ satisfies

$$(A - \lambda I)^k \mathbf{v} = 0$$

for some $k \geq 1$. The smallest such k is called the *length of the chain*. The set of all generalized eigenvectors for λ forms the *generalized eigenspace*:

$$\mathcal{G}_\lambda = \{\mathbf{v} \in \mathbb{C}^n : (A - \lambda I)^m \mathbf{v} = 0 \text{ for some } m \geq 1\}.$$

Example 9. Generalized Eigenvector Chain

et

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

The only eigenvalue is $\lambda = 2$. The vector $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ is an eigenvector:

$$(A - 2I)\mathbf{v}_1 = 0.$$

Now, $\mathbf{v}_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ is a generalized eigenvector of order 2:

$$(A - 2I)\mathbf{v}_2 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \neq 0, \quad (A - 2I)^2 \mathbf{v}_2 = 0.$$

Similarly, $\mathbf{v}_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$ is a generalized eigenvector of order 3:

$$(A - 2I)\mathbf{v}_3 = \mathbf{v}_2, \quad (A - 2I)^2 \mathbf{v}_3 = \mathbf{v}_1, \quad (A - 2I)^3 \mathbf{v}_3 = 0.$$

Thus, $\{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3\}$ forms a chain of generalized eigenvectors for $\lambda = 2$.

Structure of Jordan Form

For each eigenvalue λ:

- The number of Jordan blocks equals the geometric multiplicity (dimension of $\ker(A - \lambda I)$).
- The sum of the sizes of the Jordan blocks equals the algebraic multiplicity (multiplicity of λ as a root of the characteristic polynomial).

Example 10. Jordan Form Example

Suppose A has characteristic polynomial $(x-2)^3$ but only one linearly independent eigenvector for $\lambda = 2$. Then

$$J = \begin{pmatrix} 2 & 1 & 0 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{pmatrix}$$

is the Jordan form: a single 3×3 block for $\lambda = 2$.

Why Jordan Form Matters

- It provides a complete classification of matrices up to similarity.
- It is essential for understanding the behavior of A^k, $\exp(A)$, and solutions to differential equations $\dot{\mathbf{x}} = A\mathbf{x}$.
- It reveals the "obstructions" to diagonalization: the off-diagonal 1's indicate nontrivial chains of generalized eigenvectors.

Remark 8. Real Matrices

Every real matrix has a Jordan form over $\mathbb{C}$. Over $\mathbb{R}$, the canonical form may involve 2×2 blocks for complex conjugate eigenvalues.

1.1.5 Matrix and Vector Norms

Norms formalize the notion of size or length for vectors and matrices, providing a way to measure distances and magnitudes in linear algebra.

Definition 1.14: Vector Norm

A *norm* on $\mathbb{R}^n$ is a function $\|\cdot\| : \mathbb{R}^n \to [0, \infty)$ such that, for all $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ and $\alpha \in \mathbb{R}$:

1. **Definiteness:** $\|\mathbf{x}\| = 0 \iff \mathbf{x} = \mathbf{0}$
2. **Homogeneity:** $\|\alpha\mathbf{x}\| = |\alpha|\,\|\mathbf{x}\|$
3. **Triangle inequality:** $\|\mathbf{x} + \mathbf{y}\| \le \|\mathbf{x}\| + \|\mathbf{y}\|$

Common norms for $\mathbf{x} = (x_1, \dots, x_n)^\mathsf{T}$:

$$\begin{aligned} p\text{-norm:} \quad & \|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}, \quad 1 \le p < \infty \\ \text{Infinity norm:} \quad & \|\mathbf{x}\|_\infty = \max_{1 \le i \le n} |x_i| \\ \text{Euclidean norm:} \quad & \|\mathbf{x}\|_2 = \left(\sum_{i=1}^n |x_i|^2\right)^{1/2} \end{aligned}$$

Definition 1.15: Matrix Norm

A *matrix norm* $\|\cdot\|$ on $\mathbb{R}^{m\times n}$ is a function $\|\cdot\| : \mathbb{R}^{m\times n} \to [0,\infty)$ satisfying properties analogous to vector norms.

An *induced* (operator) norm, subordinate to a vector norm $\|\cdot\|$, is defined by

$$\|A\| = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|} = \sup_{\|\mathbf{x}\|=1} \|A\mathbf{x}\|.$$

Common induced norms for $A = (a_{ij}) \in \mathbb{R}^{m\times n}$**:**

$$\begin{aligned}
\text{1-norm:} \quad & \|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{m} |a_{ij}| \quad \text{(maximum column sum)} \\
\text{Infinity norm:} \quad & \|A\|_\infty = \max_{1\le i\le m} \sum_{j=1}^{n} |a_{ij}| \quad \text{(maximum row sum)} \\
\text{2-norm (spectral norm):} \quad & \|A\|_2 = \sup_{\|\mathbf{x}\|_2=1} \|A\mathbf{x}\|_2 = \sqrt{\lambda_{\max}(A^\top A)}
\end{aligned}$$

where $\lambda_{\max}(A^\top A)$ denotes the largest eigenvalue of $A^\top A$.

1.1.6 Spectral Radius

The spectral radius of a matrix measures the largest "scaling" by which the matrix can stretch a vector, as determined by its eigenvalues.

Definition 1.16: Spectral Radius

The *spectral radius* of a square matrix $A \in \mathbb{R}^{n\times n}$ is the largest modulus among its eigenvalues:

$$\rho(A) = \max\{|\lambda| : \lambda \in \sigma(A)\},$$

where $\sigma(A)$ denotes the set of all eigenvalues of A. That is, if A has eigenvalues $\lambda_1, \lambda_2, \dots, \lambda_n$, then

$$\rho(A) = \max_{1\le i\le n} |\lambda_i|.$$

Intuitively, it describes the long-term behavior of repeated matrix multiplication: if the spectral radius is less than one, powers of the matrix tend to zero; if greater than one, they grow without bound.

1.1.7 Condition Number

The condition number of a matrix measures how sensitive the solution of $A\mathbf{x} = \mathbf{b}$ is to small changes in A or $\mathbf{b}$. A large condition number indicates that the system is *ill-conditioned*: small errors in the data can cause large errors in the solution. If the condition number is close to 1, the system is *well-conditioned*.

Definition 1.17: Condition Number

Let $A \in \mathbb{R}^{n\times n}$ be an invertible matrix, and let $\|\cdot\|$ be a matrix norm. The *condition number* of A with respect to $\|\cdot\|$ is defined as:

$$\kappa(A) = \|A\| \cdot \|A^{-1}\|.$$

Property 4: Condition Number and Singular Values

The condition number varies with the matrix norm used:

$$
\begin{aligned}
\text{2-norm:} \quad & \kappa_2(A) = \frac{\sigma_{\max}(A)}{\sigma_{\min}(A)} \\
\text{1-norm:} \quad & \kappa_1(A) = \|A\|_1 \cdot \|A^{-1}\|_1 \\
\infty\text{-norm:} \quad & \kappa_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty
\end{aligned}
$$

where $\sigma_{\max}(A)$ and $\sigma_{\min}(A)$ denote the largest and smallest singular values of A, respectively.

Remark 9. Condition Number for Symmetric Positive Definite Matrices

If $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite, the singular values of A are equal to the absolute values of its eigenvalues. Thus, the condition number in the 2-norm simplifies to:

$$\kappa_2(A) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)},$$

where $\lambda_{\max}(A)$ and $\lambda_{\min}(A)$ are the largest and smallest eigenvalues of A, respectively.

Remark 10. Geometric Interpretation of Condition Number

The condition number describes how much A stretches the unit sphere into an ellipsoid. A large condition number indicates that the ellipsoid is highly elongated, meaning some directions are stretched significantly more than others. This implies that the matrix A is *ill-conditioned*, and small perturbations in the input can lead to large changes in the output.

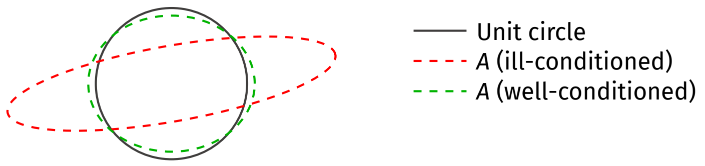


Figure 1.6: Effect of A on the unit circle: well-conditioned (nearly circular) vs. ill-conditioned (stretched).

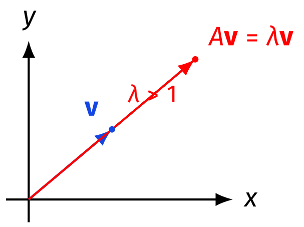


Figure 1.7: An eigenvector $\mathbf{v}$ is only stretched (not rotated) by A: $A\mathbf{v} = \lambda\mathbf{v}$

1.2 Complex Numbers and Fields

1.2.1 Complex Numbers

Definition 1.18: Complex Numbers

The set of complex numbers is

$$\mathbb{C} = \{z = x + iy : x, y \in \mathbb{R},\ i^2 = -1\}.$$

The real part is $\Re(z) = x$, the imaginary part is $\Im(z) = y$, and the complex conjugate is $\overline{z} = x - iy$.

Remark 11. Connection to Linear Algebra

Complex numbers extend the field of real numbers, allowing us to define vector spaces and matrices over $\mathbb{C}$ as well as $\mathbb{R}$. Many linear algebra concepts, such as eigenvalues and diagonalization, naturally generalize to complex vector spaces.

Property 5: Modulus and Argument

The modulus of z is $|z| = \sqrt{x^2 + y^2}$. The argument is $\arg(z) = \arctan 2(y, x)$. Euler's formula: $z = re^{i\theta}$, where $r = |z|$, $\theta = \arg(z)$.

1.2.2 Fields

Definition 1.19: Field

A *field* $\mathbb{K}$ is a set with two operations, addition and multiplication, such that:

- $(\mathbb{K}, +)$ is an abelian group with identity 0.
- $(\mathbb{K} \setminus \{0\}, \cdot)$ is an abelian group with identity 1.
- Multiplication is distributive over addition.

Examples: $\mathbb{R}$, $\mathbb{C}$, $\mathbb{Q}$.

Remark 12. Fields in Linear Algebra

Linear algebra is built on the concept of vector spaces over a field. The choice of field (e.g., $\mathbb{R}$ or $\mathbb{C}$) determines the properties of the vector spaces and matrices we study.

1.3 Polynomials

Definition 1.20: Polynomial

A polynomial of degree n over a field $\mathbb{K}$ is

$$p(x) = a_0 + a_1 x + \cdots + a_n x^n, \quad a_n \neq 0,\ a_i \in \mathbb{K}.$$

Property 6: Roots and Factorization

A polynomial of degree n has at most n roots in $\mathbb{K}$. Over $\mathbb{C}$, every polynomial of degree n has exactly n roots (counting multiplicities).

Definition 1.21: Multiplicity of a Root

The root r of $p(x)$ has multiplicity k if $(x-r)^k$ divides $p(x)$, but $(x-r)^{k+1}$ does not.

1.4 Systems of Linear Equations

1.4.1 Solving Linear Systems

Definition 1.22: Linear System

A system of m linear equations in n variables:

$$A\mathbf{x} = \mathbf{b}, \quad A \in \mathbb{R}^{m\times n},\ \mathbf{x} \in \mathbb{R}^n,\ \mathbf{b} \in \mathbb{R}^m.$$

- If A is square and invertible, the unique solution is $\mathbf{x} = A^{-1}\mathbf{b}$.
- If A is not invertible or not square, solutions may not exist or may not be unique.

1.4.2 Gaussian Elimination

Definition 1.23: Gaussian Elimination

A systematic procedure to solve $A\mathbf{x} = \mathbf{b}$ by reducing A to row echelon form using elementary row operations.

Remark 13. Pivoting

Pivoting (row exchanges) is used to avoid division by zero and reduce numerical errors.

1.5 Basic Set Theory and Functions

1.5.1 Sets and Mappings

Definition 1.24: Set

A set is a collection of distinct objects. Notation: $A = \{a_1, a_2, \ldots\}$.

Definition 1.25: Function (Mapping)

A function $f : X \to Y$ assigns to each $x \in X$ a unique $y = f(x) \in Y$.

- f is injective (one-to-one) if $f(x_1) = f(x_2) \implies x_1 = x_2$.
- f is surjective (onto) if for every $y \in Y$, there exists $x \in X$ with $f(x) = y$.
- f is bijective if it is both injective and surjective.

1.6 Basic Topology in $\mathbb{R}^n$

1.6.1 Open and Closed Sets

Definition 1.26: Open Set

A set $U \subset \mathbb{R}^n$ is open if for every $x \in U$, there exists $\varepsilon > 0$ such that the ball $B(x, \varepsilon) \subset U$.

Definition 1.27: Closed Set

A set $F \subset \mathbb{R}^n$ is closed if its complement is open, or equivalently, if it contains all its limit points.

1.6.2 Boundedness and Compactness

Definition 1.28: Bounded Set

$S \subset \mathbb{R}^n$ is bounded if there exists $M > 0$ such that $\|x\| < M$ for all $x \in S$.

Definition 1.29: Compact Set

$K \subset \mathbb{R}^n$ is compact if it is closed and bounded (Heine-Borel theorem).

1.7 Sequences and Limits

Definition 1.30: Sequence

A sequence in $\mathbb{R}^n$ is a function $x : \mathbb{N} \to \mathbb{R}^n$, written as (x_k).

Definition 1.31: Limit of a Sequence

(x_k) converges to x if for every $\varepsilon > 0$, there exists N such that $k \geq N \implies \|x_k - x\| < \varepsilon$.

1.8 Basic Calculus in Several Variables

1.8.1 Continuity and Differentiability

Definition 1.32: Continuous Function

$f : \mathbb{R}^n \to \mathbb{R}^m$ is continuous at x_0 if for every $\varepsilon > 0$, there exists $\delta > 0$ such that $\|x - x_0\| < \delta \implies \|f(x) - f(x_0)\| < \varepsilon$.

Definition 1.33: Differentiable Function

$f : \mathbb{R}^n \to \mathbb{R}^m$ is differentiable at x_0 if there exists a linear map $A : \mathbb{R}^n \to \mathbb{R}^m$ such that

$$\lim_{h \to 0} \frac{\|f(x_0 + h) - f(x_0) - Ah\|}{\|h\|} = 0.$$

The matrix A is the Jacobian of f at x_0.

1.9 Basic Notation and Conventions

- $\mathbb{N}$: natural numbers, $\mathbb{Z}$: integers, $\mathbb{Q}$: rationals, $\mathbb{R}$: reals, $\mathbb{C}$: complex numbers.
- Vectors are columns by default; $\mathbf{x}$ denotes a vector, A a matrix.
- $\|\cdot\|$ denotes a norm; $\langle \cdot, \cdot \rangle$ denotes an inner product.
- I_n: $n \times n$ identity matrix; 0: zero vector or matrix (context-dependent).

Chapter 2

Numerical Principles and Error Analysis

2.1 Mathematical Framework: From Problems to Algorithms

Before diving into error analysis, we must understand the fundamental structure of computational mathematics. Every numerical computation follows a pipeline from real-world phenomena to algorithmic solutions.

2.1.1 Direct and Inverse Problems

At the heart of numerical mathematics lies the general problem:

$$\boxed{F(x; d) = 0}$$

where:

- d represents the *data* on which the solution depends
- x is the *solution* we seek
- F models the *relationship* between data and solution

> **Example 11. Linear Systems**
>
> For $F(x; b) = Ax - b = 0$, we solve the linear system $Ax = b$. Here A is part of the model F, b is the data, and x is the unknown solution.

Classification of Problems

Direct Problem Given F and d, find x such that $F(x; d) = 0$.
Inverse Problem Given F and x, determine the data d.
Identification Problem Given x and d, determine the model F.

2.1.2 From Problems to Algorithms

Definition 2.1: Algorithm

An algorithm is a sequence of deterministic instructions consisting of elementary operations, designed to solve a problem and guaranteed to terminate in finitely many steps.

We measure algorithms by three key criteria:

Accuracy How close do we get to the exact solution? How does error decrease as we invest more computational effort?

Reliability How likely is it that the global error remains small? (Requires testing and statistical analysis)

Efficiency What computational complexity is needed to achieve a desired accuracy?

2.2 Well-Posedness: The Foundation of Stable Computation

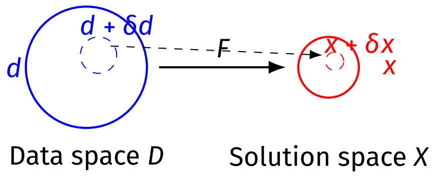


Figure 2.1: Well-posed problem: small changes in data d lead to small changes in solution x.

Definition 2.2: Well-Posedness (Hadamard)

The problem $F(x; d) = 0$ is *well-posed* if:

1. **Existence:** A solution x exists
2. **Uniqueness:** The solution is unique
3. **Continuous dependence:** The solution x depends continuously on the data d

If any condition fails, the problem is *ill-posed* or *unstable*.

2.2.1 Rigorous Definition of Continuous Dependence

Let D be the set of admissible data. Suppose we measure perturbed data $d + \delta d \in D$ instead of the true data $d \in D$. This yields a perturbed solution satisfying $F(x + \delta x; d + \delta d) = 0$.

Definition 2.3: Stability

A problem is *stable* if there exist constants $\eta_0 = \eta_0(d) > 0$ and $K_0 = K_0(d)$ such that:

$$\|\delta d\| \leq \eta_0 \quad \Rightarrow \quad \|\delta x\| \leq K_0 \|\delta d\|$$

In words: small changes in data should yield proportionally small changes in the solution.

2.3 Condition Numbers: Quantifying Sensitivity

2.3.1 Relative and Absolute Condition Numbers

Definition 2.4: Condition Numbers

For the problem $F(x; d) = 0$:

Relative condition number:

$$\kappa(d) = \sup_{\substack{\delta d \neq 0 \\ d+\delta d \in D}} \frac{\|\delta x\| / \|x\|}{\|\delta d\| / \|d\|}$$

Absolute condition number (when $d = 0$ or $x = 0$):

$$\kappa_{\text{abs}}(d) = \sup_{\substack{\delta d \neq 0 \\ d+\delta d \in D}} \frac{\|\delta x\|}{\|\delta d\|}$$

2.3.2 Computing Condition Numbers via Linearization

Since $F(x; d) = 0$ has a unique solution, we can define the solution map $G : d \mapsto x$ implicitly via $F(G(d); d) = 0$.

For the error, $G(d + \delta d) = x + \delta x$. Assuming G is differentiable with Jacobian $G'(d)$:

$$G(d + \delta d) = G(d) + G'(d)\delta d + o(\|\delta d\|) \quad \text{as } \delta d \to 0$$

This gives us:

$$\boxed{\begin{aligned} \kappa(d) &\approx \|G'(d)\| \frac{\|d\|}{\|G(d)\|} \\ \kappa_{\text{abs}}(d) &\approx \|G'(d)\| \end{aligned}}$$

Example 12. Square Root Problem - Ill-Conditioning

Consider solving $x^2 - 2px + 1 = 0$ for $p \geq 1$. The solutions are $x_\pm = p \pm \sqrt{p^2 - 1}$.
For the solution map $G_-(p) = p - \sqrt{p^2 - 1}$:

$$G_-'(p) = 1 - \frac{p}{\sqrt{p^2 - 1}}$$

The condition number becomes:

$$\kappa(p) \approx \frac{|p|}{|p - \sqrt{p^2 - 1}|} \cdot \left| 1 - \frac{p}{\sqrt{p^2 - 1}} \right| = \frac{p}{\sqrt{p^2 - 1}}$$

For p close to 1, $\kappa(p) \to \infty$ — the problem becomes ill-conditioned!

Example 13. Square Root Problem - Reformulation

Reformulate using $t = p + \sqrt{p^2 - 1}$, so $p = \frac{1+t^2}{2t}$.

The new problem: $F(x, t) = x^2 - \frac{x(1+t^2)}{t} + 1 = 0$

Solutions: $G_{-}(t) = t$ and $G_{+}(t) = \frac{1}{t}$

Derivatives: $G'_{-}(t) = 1$ and $G'_{+}(t) = -\frac{1}{t^2}$

Now $\kappa(t) \approx 1$ for any t — much better conditioning!

Remark 14. Key Insight

A large condition number might indicate that you need to reformulate your problem, not just use higher precision arithmetic.

2.4 Why Worry About Error?

Imagine steering a ship with a compass that is *almost* correct. Each glance introduces a tiny angular mistake; over time these slips can push the vessel far off course. Numerical algorithms face the same danger: every arithmetic operation or approximation injects a micro-error that may *cancel*, *accumulate*, or *amplify*.

Error analysis is the navigator's sea-chart. It answers:

- where the error comes from (sources),
- how large it is now (bounds),
- how it propagates (stability),
- and how fast it shrinks as we refine the mesh or increase precision (convergence).

2.5 Stability of Numerical Methods

When we solve $F(x; d) = 0$ approximately through iterative methods, we work with sequences:

$$F_n(x_n; d_n) = 0, \quad n = 1, 2, \ldots$$

where $x_n \to x$ and $d_n \to d$ as $n \to \infty$.

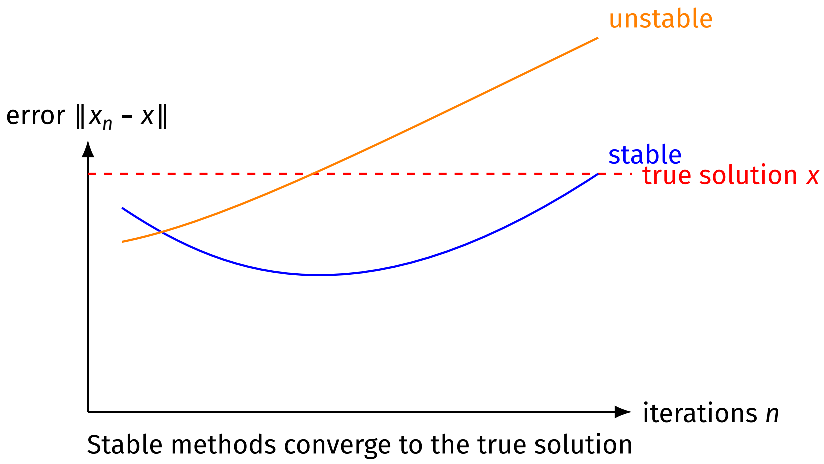


Figure 2.2: Stable vs. unstable numerical methods: error behavior over iterations.

2.5.1 Consistency, Stability, and Convergence

Definition 2.5: Consistency

A numerical method F_n is *consistent* with the continuous problem F if:

$$F_n(x; d) - F(x; d) \to 0 \quad \text{as } n \to \infty$$

for any admissible data $d \in D$ and its exact solution x.

Definition 2.6: Convergence

A numerical method is *convergent* if for all $\varepsilon > 0$, there exist $n_0 = n_0(\varepsilon)$ and $\delta = \delta(n_0, \varepsilon)$ such that:

$$n > n_0 \text{ and } \|\delta d_n\| \leq \delta \quad \Rightarrow \quad \|x(d) - x_n(d + \delta d_n)\| \leq \varepsilon$$

Theorem 2.7: Lax-Richtmyer Equivalence

For well-posed linear problems, a consistent finite difference method is convergent if and only if it is stable.

This fundamental theorem establishes the theoretical foundation for numerical analysis:

$$\boxed{\text{Consistency} + \text{Stability} \iff \text{Convergence}}$$

Interpretation

- **Consistency** ensures each step approximates the differential equation correctly
- **Stability** prevents local errors from amplifying catastrophically
- **Convergence** guarantees the numerical solution approaches the exact solution as the mesh is refined

The theorem reveals that consistency alone is insufficient—without stability, small local truncation errors can grow exponentially, destroying convergence regardless of the method's formal order of accuracy.

Example 14. Third-Order Adams-Bashforth

Coefficients:

$$\beta_0 = \frac{5}{12}, \quad \beta_1 = -\frac{4}{3}, \quad \beta_2 = \frac{23}{12}$$

give:

$$u_{n+3} = u_{n+2} + h\left(\frac{23}{12} f_{n+2} - \frac{4}{3} f_{n+1} + \frac{5}{12} f_n\right)$$

Local error A fourth-order Taylor expansion yields:

$$\tau_{n+3} = \frac{3}{8} h^4 y^{(4)}(\xi_n) = \mathcal{O}(h^4),$$

so the method is consistent with order $p = 3$.

Zero-stability $\rho(r) = r^3 - r^2 = r^2(r-1)$ has roots $0, 0, 1$; all lie inside or on the unit circle and the unit root is simple.

> **Convergence** By Dahlquist's theorem (the ODE analogue of Lax-Richtmyer):
>
> $$\max_{0\le n\le T/h} |u_n - y(t_n)| = \mathcal{O}(h^3)$$

2.5.2 A Priori vs. A Posteriori Error Analysis

A Priori Analysis Estimates error bounds *before* computation:

- **Forward Analysis:** Bounds $\|\delta x_n\|$ due to $\|\delta d\|$ or method properties
- **Backward Analysis:** Given computed $\hat{x}_n$, find δd such that $F_n(\hat{x}_n; d + \delta d) = 0$

A Posteriori Analysis Estimates actual error using computed results:

- Estimate $\|\hat{x}_n - x\|$ using residual $r_n = F(\hat{x}_n; d)$
- Adaptive methods use this for step size control

2.6 Sources of Error

Understanding error sources is crucial for developing reliable numerical methods. Errors propagate through a computational pipeline from the real world to our final answer.

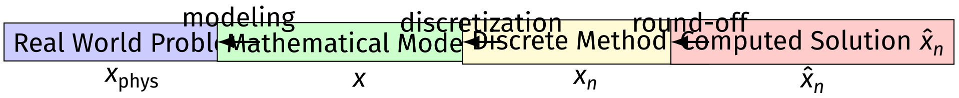


Figure 2.3: Error propagation pipeline: from physics to computation.

The total error decomposes as:

$$\boxed{e_{\text{total}} = \underbrace{(x_{\text{phys}} - x)}_{\text{modeling}} + \underbrace{(x - x_n)}_{\text{discretization}} + \underbrace{(x_n - \hat{x}_n)}_{\text{round-off}}}$$

2.6.1 Modeling Error

The mismatch between the *mathematical model* and reality (e.g., replacing the Navier-Stokes equations by potential flow). No amount of numerical refinement can undo this bias.

> **Example 15. Modeling Error**
>
> Using the linear heat equation $u_t = \alpha u_{xx}$ to model heat conduction neglects:
>
> - Nonlinear temperature-dependent conductivity
> - Radiation heat transfer
> - Convective effects

2.6.2 Data Error

Inexact initial or boundary data, coefficients, or forcing terms. For linear multistep ODE solvers these perturbations behave like round-off: they are injected at start-up and spread according to the method's stability.

2.6.3 Discretization (Truncation) Error

Approximating an infinite object by a finite stencil/series. A forward difference illustrates the idea:

$$f'(x) - \frac{f(x+h) - f(x)}{h} = \frac{h}{2} f''(\xi) = \mathcal{O}(h)$$

The residual that remains after *one* step is called the *local truncation error (LTE)*.

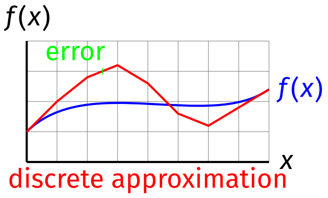


Figure 2.4: Discretization error: approximating a smooth function on a finite grid.

2.6.4 Floating-Point (Round-Off) Error

Computers represent reals as:

$$x = \pm m \times 2^e, \qquad m, e \in \mathbb{Z}$$

so most numbers are stored only approximately.

For every $\circ \in \{+, -, \times, /\}$:

$$\mathrm{fl}(x \circ y) = (x \circ y)(1 + \delta), \qquad |\delta| \le \varepsilon_{\text{mach}}$$

The *machine epsilon* is the smallest $\varepsilon_{\text{mach}}$ with $1 + \varepsilon_{\text{mach}} > 1$; for IEEE double precision $\varepsilon_{\text{mach}} = 2^{-53} \approx 1.1 \times 10^{-16}$.

Remark 15. Catastrophic Cancellation

When subtracting nearly equal numbers, relative error can explode:

$$\frac{(a + \delta a) - (b + \delta b)}{a - b} = 1 + \frac{\delta a - \delta b}{a - b}$$

If $a \approx b$, then $\frac{\delta a - \delta b}{a - b}$ can be huge!

2.7 Measuring Error

2.7.1 Local Truncation Error (LTE)

For a k-step multistep method:

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

Insert the *exact* solution y:

$$\tau_{n+k} = \frac{1}{h}\Big(\sum_{j=0}^{k} \alpha_j y(t_{n+j}) - h \sum_{j=0}^{k} \beta_j f(t_{n+j}, y(t_{n+j}))\Big)$$

If $\tau_{n+k} = \mathcal{O}(h^{p+1})$, the method has *order* p.

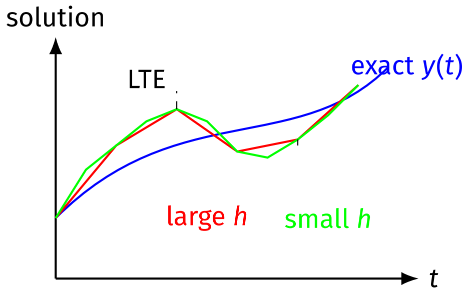


Figure 2.5: Local truncation error decreases as step size h becomes smaller.

2.7.2 Global Discretization Error (GDE)

After many steps:

$$e_n = u_n - y(t_n), \qquad n = 0, 1, \dots, N$$

Under zero-stability:

$$\max_{0 \le n \le N} |e_n| \le C \max_{0 \le n \le N} |\tau_n|$$

Thus for multistep methods, global and local orders coincide; for Runge-Kutta, global order is one lower than local order.

2.8 Stability Types in Numerical Methods

2.8.1 Zero-Stability

Zero-stability is fundamental for *linear multistep methods*. Unlike one-step methods, multistep methods use several past values, so errors can persist through the recurrence relation.

For a k-step method with characteristic polynomial $\rho(r) = \sum_{j=0}^{k} \alpha_j r^j$, zero-stability requires:

- All roots satisfy $|r| \le 1$
- Any roots on the unit circle are simple:

$$\begin{aligned} \rho(r) &= 0 \quad \text{for } |r| = 1 \\ \rho'(r) &\neq 0 \quad \text{for } |r| = 1 \end{aligned}$$

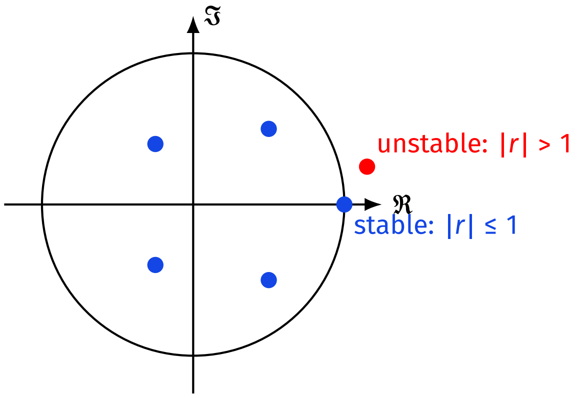
2.8.2 *A*-Stability and *L*-Stability

Let $\Phi(z)$ be the stability function for $y' = \lambda y$.

Definition 2.8: *A*-Stability and *L*-Stability

A method is:

- ***A*-stable** if $|\Phi(z)| \le 1$ whenever $\Re z < 0$
- ***L*-stable** if, in addition, $\Phi(z) \to 0$ as $z \to -\infty$



Zero-stable: all roots inside or on unit circle

Figure 2.6: Root condition for zero-stability of multistep methods.

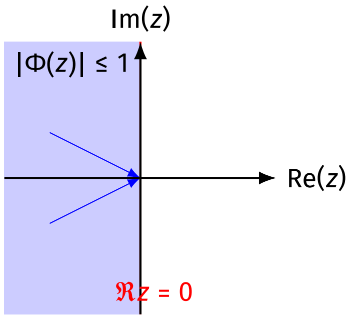


Figure 2.7: A-stability region: the entire left half-plane.

2.8.3 Comparison of Stability Types

Stability	Methods	Stiffness-safe	Fast-mode damping
Zero-stable	Multistep	No	No
A-stable	One-step, Multistep	Yes	No
L-stable	One-step, Multistep	Yes	Yes

Which to use?

- **Non-stiff:** Zero-stable multistep is sufficient
- **Stiff:** Choose an A-stable method for larger, stable steps
- **Very stiff:** Use an L-stable scheme for rapid damping of stiff modes

2.9 Advanced Topics in Error Analysis

2.9.1 Conditioning vs. Stability

It's crucial to distinguish between:

Well-conditioned problem The mathematical problem $F(x; d) = 0$ has a small condition number

Stable algorithm The numerical method produces solutions that are insensitive to perturbations

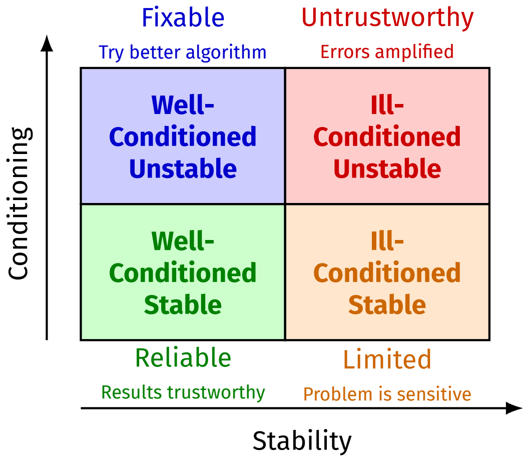


Figure 2.8: Conditioning (problem sensitivity) vs. stability (algorithm robustness): Only the lower-left is truly reliable. If the algorithm is unstable, try a better method; if the problem is ill-conditioned, no algorithm can fully fix it.

An ill-conditioned *problem* means even the best algorithm will struggle; an unstable *algorithm* can often be improved (e.g., by using pivoting or a more robust method).

2.9.2 Backward Error

Instead of asking ”How wrong is my solution?” ask ”Which *nearby* problem did I solve *exactly*?”

A p-th order Runge-Kutta method solves:

$$y' = f(t, y) + h^p g_h(t, y), \qquad g_h = \mathcal{O}(1)$$

Long-time behavior (e.g., energy conservation in symplectic integrators) is often easier to explain in this *modified equation* picture.

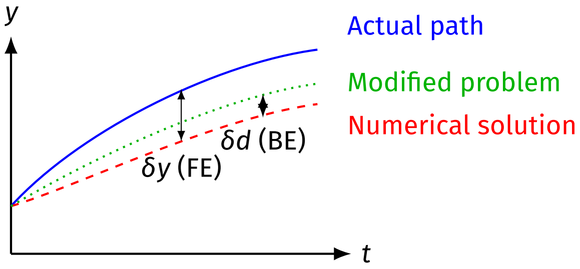


Figure 2.9: Backward error: the computed solution solves a slightly perturbed problem exactly—often closer than the forward error.

2.9.3 Controlling Error

Effective error control follows a systematic approach:

1. **Problem Reformulation:** Address ill-conditioning at the mathematical level
2. **Method Selection:** Choose algorithms suited to problem characteristics

3. **Adaptive Control:** Automatically adjust computational parameters
4. **Post-processing:** Apply error reduction techniques
5. **Verification:** Validate results through multiple approaches

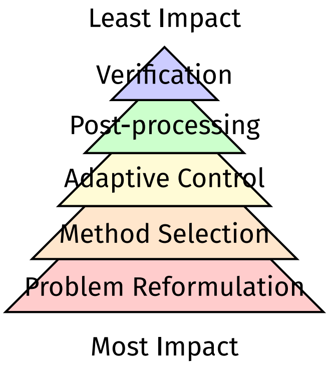


Figure 2.10: Error control hierarchy: address fundamental issues first.

2.9.4 Method Selection Guidelines

Problem	**Recommended Methods**	**Considerations**
Smooth ODEs	High-order Runge-Kutta, Adams methods	Efficiency over stability
Stiff ODEs	Implicit methods (BDF, RADAU)	A- or L-stability required
Oscillatory	Symplectic, exponential integrators	Preserve energy/oscillations
Conservation laws	Geometric integrators	Structure preservation
Rough data	Robust methods, regularization	Stability over high order

2.9.5 Algorithmic Best Practices

Numerical Hygiene

- Avoid subtracting nearly equal numbers
- Use stable algorithms (QR vs. normal equations)
- Scale problems to avoid extreme magnitudes
- Check intermediate results for sanity

Error Monitoring

- Track residuals and error estimates
- Use multiple precision levels for comparison
- Implement convergence tests with multiple criteria
- Monitor conservation quantities when applicable

2.10 Case Studies in Error Analysis

2.10.1 Case Study 1: The Quadratic Formula

Consider solving $ax^2 + bx + c = 0$ with $a = 1$, $b = -2000$, $c = 1$.

Standard Formula (Poor):

$$x = \frac{2000 \pm \sqrt{4000000 - 4}}{2} = \frac{2000 \pm 1999.999}{2}$$

One root suffers catastrophic cancellation: $x_1 = \frac{2000-1999.999}{2} = 0.0005$

Improved Formula (Good): Use $x_1 x_2 = c/a = 1$, so:

$$x_1 = \frac{2c}{-b - \sqrt{b^2 - 4ac}} = \frac{2}{2000 + 1999.999} = \frac{1}{1999.9995}$$

2.10.2 Case Study 2: Computing $e^x - 1$ for Small x

Direct Computation (Poor): For $x = 10^{-8}$: $e^x - 1 = 1.00000001 - 1 = 10^{-8}$ (loses digits)

Series Expansion (Good):

$$e^x - 1 = x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

For small x, truncate after a few terms.

Built-in Function (Best): Use `expm1(x)` which is designed for this purpose.

2.11 Summary and Final Insights

Fundamental Principles

1. **Error is inevitable.** Every computation introduces approximation; the goal is to understand and control it.
2. **Stability dominates accuracy.** A stable method with modest accuracy often outperforms an unstable high-order scheme.
3. **Conditioning reveals problem difficulty.** Large condition numbers signal fundamental limitations, not implementation flaws.
4. **Structure preservation matters.** For long-time integration, qualitative correctness can be more important than pointwise accuracy.
5. **Multiple perspectives illuminate.** Forward, backward, and probabilistic error analyses each reveal different aspects of computational behavior.

2.11.1 The Error Analysis Toolkit

Theoretical Tools Condition numbers, stability analysis, convergence theory
Computational Tools Adaptive methods, error estimation, residual monitoring
Diagnostic Tools Multiple precision, alternative algorithms, conservation checks
Preventive Tools Problem reformulation, method selection, numerical hygiene

Remark 17. Final Thought

Error analysis is not just about finding mistakes—it's about understanding the fundamental limits and possibilities of computation.
Master these principles, and you'll develop an intuition for when numerical results can be trusted

> and when they require deeper investigation.

Part II

Approximation Methods

Chapter 3

Orthogonal Polynomials

Orthogonal polynomials underpin Gaussian quadrature, spectral methods for differential equations, and least-squares data fitting. They provide numerically stable bases because their three-term recurrences avoid the ill-conditioning of raw monomials. Classical families such as Chebyshev or Legendre deliver nearly optimal approximation and interpolation properties that we exploit in later chapters.

3.1 General Setup

We consider an interval, usually $[-1,1]$, $[-\pi,\pi)$, $[0,\infty)$, or $\mathbb{R}$, and a weight function $w(x)$ with $w(x) > 0$ and integrable (possibly unbounded at endpoints). The inner product on function spaces is defined as

$$(f,g)_w := \int_{-1}^{1} f(x)g(x)w(x)\,dx.$$

This inner product is linear in the first argument, symmetric, and positive definite, and induces the norm $\|f\|_w := \sqrt{(f,f)_w}$. If $(f,g)_w = 0$, we say f and g are orthogonal with respect to w.

Example 16. Monomials

Let $f(x) = x^k$, $g(x) = x^l$ on $[-1,1]$ with $w \equiv 1$. The monomials are orthogonal if $k \neq l$.

$$(x^k, x^l)_w = \int_{-1}^{1} x^{k+l}\,dx = \begin{cases} 0 & \text{if } k+l \text{ odd,} \\ \frac{2}{k+l+1} & \text{if } k+l \text{ even.} \end{cases}$$

Thus, the monomials are orthogonal whenever $k \neq l$ because the integral vanishes when $k+l$ is odd.

Orthogonalizing the monomials by Gram-Schmidt yields bases with dramatically lower condition numbers than unscaled Vandermonde systems, making least-squares approximation numerically stable.

3.2 Orthogonal and Orthonormal Polynomials

Let $\{p_0, p_1, \ldots\}$ be a sequence of polynomials with $\deg p_k = k$. Given an interval I and weight w, the sequence is orthogonal if

$$(p_i, p_j)_w = 0 \quad \text{for } i \neq j.$$

An orthonormal system is obtained by normalizing so that $\|p_k\|_w = 1$.

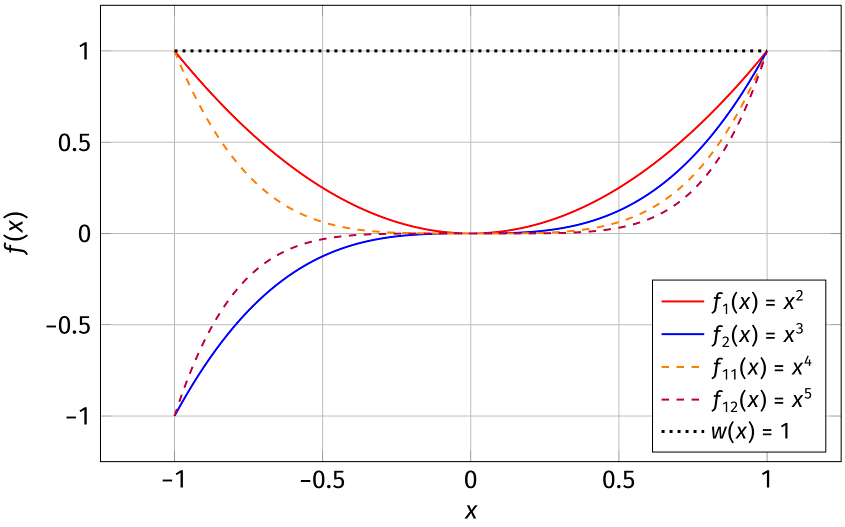


Figure 3.1: Monomials x^2, x^3, x^4, and x^5 on $[-1, 1]$ with weight $w(x) = 1$.

3.3 Construction of Orthogonal Polynomials via Gram-Schmidt

Orthogonalize $\{1, x, x^2, x^3, ...\}$ using **Gram-Schmidt**. Start with $p_0(x) = 1$.

For $p_1(x)$, we start with the most general degree-1 polynomial,

$$p_1(x) = x + a_{1,0},$$

and determine $a_{1,0}$ so that p_1 is orthogonal to p_0. Specifically, we require

$$\begin{aligned}(p_0, p_1)_w &= 0\\(1, x + a_{1,0})_w &=\\(1, x)_w + a_{1,0}(1, 1)_w &= 0.\end{aligned}$$

Solving for $a_{1,0}$ gives

$$a_{1,0} = -\frac{(1, x)_w}{(1, 1)_w}.$$

Continue this process for higher degrees, always ensuring:

- We choose the monic normalization, i.e., $p_k(x) = x^k + \cdots$ (leading coefficient1),
- Orthogonality: $(p_j, p_k)_w = 0$ for $j \neq k$,
- Degree: $\deg p_k = k$.

Thus, $\Pi_n = \text{span}\{1, x, \dots, x^n\} = \text{span}\{p_0, p_1, \dots, p_n\}$.

3.4 Generalized Fourier Series

Let $L^2_w = \{f : \|f\|^2_w = \int_{-1}^{1} f^2(x)w(x)\,dx < \infty\}$. For $f \in L^2_w$, we can expand

$$Sf = \sum_{k=0}^{\infty} \hat{f}_k p_k, \quad \text{where} \quad \hat{f}_k = \frac{(f, p_k)_w}{(p_k, p_k)_w}.$$

The partial sum $f_n(x) = S_n f(x) := \sum_{k=0}^{n} \hat{f}_k p_k$ converges in L^2_w-norm to f as $n \to \infty$. By Bessel's inequality (and Parseval's identity when the series converges), these coefficients guarantee $\|f - S_n f\|_w \to 0$ as $n \to \infty$.

Theorem 3.1: Best Approximation in L^2_w

The truncated generalized Fourier series $f_n(x) = S_n f(x) = \sum_{k=0}^{n} \hat{f}_k p_k$ is the best approximation in $\| \cdot \|_w$ with respect to P_n, i.e.,

$$\|f_n - f\|_w = \min_{q \in P_n} \|f - q\|_w.$$

Proof. For any $q \in P_n$, write $q = \sum_{k=0}^{n} \hat{q}_k p_k$. Then,

$$f - q = (f - f_n) + (f_n - q) = (f - f_n) + \sum_{k=0}^{n} (\hat{f}_k - \hat{q}_k) p_k.$$

Since $f - f_n$ is orthogonal to P_n, we have $(f - f_n, q)_w = 0$ for all $q \in P_n$. Therefore,

$$\|f - q\|_w^2 = \|f - f_n\|_w^2 + \|f_n - q\|_w^2 \geq \|f - f_n\|_w^2,$$

with equality if and only if $q = f_n$. Thus, f_n is the unique best approximation in P_n. Uniqueness follows from the strict convexity of the L^2-norm. □

3.4.1 Three-Term Recursion

Orthogonal polynomials can be constructed recursively using a simple three-term relation. Starting with $p_0(x)$ and $p_1(x)$, we compute $p_n(x)$ for $n \geq 2$ as follows:

$$p_n(x) = x p_{n-1}(x) - \sum_{k=0}^{n-1} a_{n,k} p_k(x),$$

where the coefficients $a_{n,k}$ are chosen to ensure orthogonality. Specifically, we require:

$$(p_n, p_j)_w = 0 \quad \text{for all } j < n.$$

This condition simplifies to:

$$a_{n,j} = \frac{(x p_{n-1}, p_j)_w}{(p_j, p_j)_w}, \quad j = 0, \dots, n-1.$$

In practice, most of these coefficients vanish due to orthogonality, leaving only two nonzero terms: $a_{n,n-1}$ and $a_{n,n-2}$. This leads to the efficient three-term recursion:

$$p_{n+1}(x) = (x - \alpha_n) p_n(x) - \beta_n p_{n-1}(x),$$

where the coefficients are given by:

$$\alpha_n = \frac{(x p_n, p_n)_w}{(p_n, p_n)_w}, \quad \beta_n = \frac{(p_n, p_n)_w}{(p_{n-1}, p_{n-1})_w}.$$

This recursive approach is widely used for constructing orthogonal polynomial systems efficiently.

Remark 18. Efficiency of Three-Term Recursion

Three-term recurrences compute p_{n+1} from p_n and p_{n-1} in $\mathcal{O}(n)$ arithmetic, avoiding the full Gram-Schmidt cost and rounding-error accumulation.

3.5 Zeros of Orthogonal Polynomials

Theorem 3.2: Real Zeros of Functions

Let $f \in \mathcal{C}([a,b])$, $f \not\equiv 0$, and $n \in \mathbb{N}$ be given. Assume that

$$(f,p)_w = 0 \quad \text{for all } p \in P_{n-1}.$$

(In particular, we could take $f = p_n$.) Then f changes sign at least n times on $[a,b]$; that is, f has at least n real zeros.

Proof. Let $f \in \mathcal{C}([a,b])$, $f \not\equiv 0$, and assume $(f,p)_w = 0$ for all $p \in P_{n-1}$. This means f is orthogonal to all polynomials of degree less than n with respect to the weight $w(x)$.

1. Suppose f does not change sign at least n times. Then $f(x)$ can be expressed as $f(x) = g(x)h(x)$, where $g(x) \in P_{n-1}$ and $h(x)$ is a non-negative function that does not change sign.
2. Consider the inner product:

$$(f,g)_w = \int_a^b g^2(x)h(x)w(x)\,dx.$$

3. Since $g^2(x) \geq 0$ and $h(x) \geq 0$, the integral $(f,g)_w \geq 0$.
4. This contradicts the assumption that $(f,p)_w = 0$ for all $p \in P_{n-1}$ unless $f \equiv 0$, which is excluded by hypothesis.

⇒ Therefore, f must change sign at least n times on $[a,b]$ and has at least n real zeros.

□

3.6 Chebyshev Polynomials

On $[-1,1]$ with weight $w(x) = \frac{1}{\sqrt{1-x^2}}$, the Chebyshev polynomials are defined by

$$T_k(x) = \cos(k \arccos x), \qquad x = \cos\theta.$$

They satisfy the recurrence

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x).$$

Property 7: Properties of Chebyshev Polynomials

- T_k is even for even k, odd for odd k.
- $|T_k(x)| \leq 1$ for $x \in [-1,1]$.
- Orthogonality:

$$(T_k, T_l)_w = \int_{-1}^{1} T_k(x)T_l(x)\,\frac{w(x)\,dx}{\sqrt{1-x^2}}, \qquad w(x) = \frac{1}{\sqrt{1-x^2}} = \begin{cases} 0 & k \neq l, \\ \pi & k = l = 0, \\ \frac{\pi}{2} & k = l > 0. \end{cases}$$

- *Minimax property:* T_k minimizes the uniform norm among all monic polynomials of degree k; hence $\|T_k\|_\infty = 1$ on $[-1,1]$.

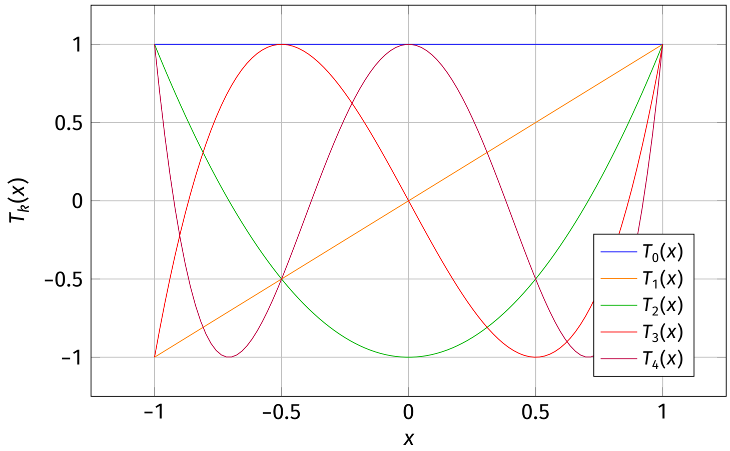


Figure 3.2: Chebyshev polynomials T_0 to T_4 on $[-1, 1]$.

3.7 Legendre Polynomials

On $[-1, 1]$ with $w(x) \equiv 1$, the Legendre polynomials satisfy

$$P_0(x) = 1, \quad P_1(x) = x,$$

and the recurrence

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x).$$

Using normalized polynomials ($P_n(1) = 1$):

$$P_{n+1}(x) = \frac{2n+1}{n+1}xP_n(x) - \frac{n}{n+1}P_{n-1}(x).$$

Property 8: Properties of Legendre Polynomials

- $P_n(x)$ is an even function for even n and an odd function for odd n.
- Orthogonality:

$$(P_k, P_l)_w = \int_{-1}^{1} P_k(x)P_l(x)\,dx = \begin{cases} 0 & \text{if } k \neq l, \\ \frac{2}{2k+1} & \text{if } k = l. \end{cases}$$

3.8 Jacobi Polynomials

A broad family of orthogonal polynomials on $[-1, 1]$ is given by the **Jacobi polynomials** $P_n^{(\alpha,\beta)}(x)$, which are orthogonal with respect to the weight

$$w(x) = (1-x)^\alpha(1+x)^\beta, \qquad \alpha, \beta > -1.$$

Many classical polynomials are special cases of Jacobi polynomials:

- **Legendre polynomials:** $\alpha = \beta = 0$,

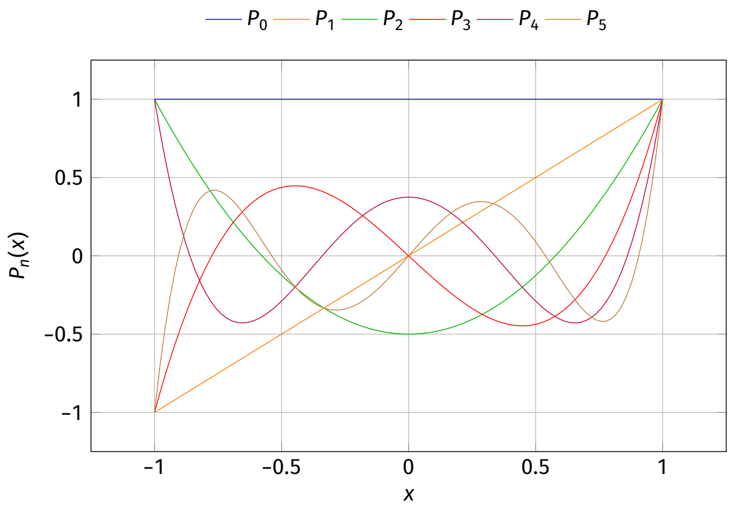


Figure 3.3: Legendre polynomials P_0 to P_5 on $[-1, 1]$.

- **Chebyshev polynomials:** $\alpha = \beta = -\frac{1}{2}$.

Jacobi polynomials satisfy a three-term recurrence relation (see subsection 3.4.1), with coefficients depending on α and β:

$$\begin{aligned}
P_0^{(\alpha,\beta)}(x) &= 1,\\
P_1^{(\alpha,\beta)}(x) &= \frac{1}{2}\Big[2(\alpha+1) + (\alpha+\beta+2)(x-1)\Big],\\
P_{n+1}^{(\alpha,\beta)}(x) &= \frac{(2n+\alpha+\beta+1)\Big[(2n+\alpha+\beta)(2n+\alpha+\beta+2)x+\alpha^2-\beta^2\Big]}{2(n+1)(n+\alpha+\beta+1)} P_n^{(\alpha,\beta)}(x) - \frac{2(n+\alpha)(n+\beta)(2n+\alpha+\beta+2)}{2(n+1)(n+\alpha+\beta+1)(2n+\alpha+\beta)} P_{n-1}^{(\alpha,\beta)}(x).
\end{aligned}$$

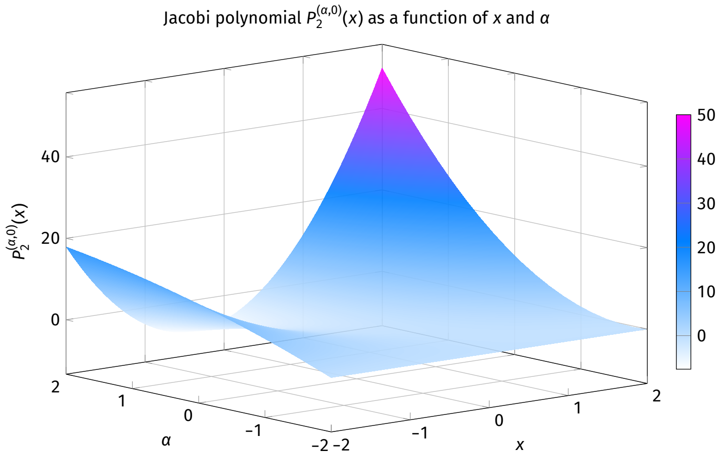


Figure 3.4: Surface plot of the Jacobi polynomial $P_2^{(\alpha,0)}(x)$ as a function of x and α.

Chapter 4

Fourier Analysis

4.1 Fourier Series

A Fourier series represents any reasonably well–behaved $2L$–periodic function $f(x)$ as an infinite linear combination of sines and cosines.

4.1.1 Orthogonal Basis of Sines and Cosines

The key idea is that the set

$$\mathcal{S} := \left\{1, \cos\frac{\pi x}{L}, \sin\frac{\pi x}{L}, \cos\frac{2\pi x}{L}, \sin\frac{2\pi x}{L}, \dots\right\}$$

forms an orthogonal basis on the interval $[-L, L]$.

Because sines and cosines of different frequencies are orthogonal, cross–terms disappear when computing the coefficients:

Proposition 1: Orthogonality of Sines and Cosines

For integers $m, n \geq 0$ on the interval $[-L, L]$, the following orthogonality relations hold:

$$\int_{-L}^{L} \cos\left(\frac{m\pi x}{L}\right)\cos\left(\frac{n\pi x}{L}\right) dx = \begin{cases} 0, & m \neq n, \\ L, & m = n \neq 0, \\ 2L, & m = n = 0, \end{cases}$$

$$\int_{-L}^{L} \sin\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi x}{L}\right) dx = \begin{cases} 0, & m \neq n, \\ L, & m = n \neq 0, \end{cases}$$

$$\int_{-L}^{L} \cos\left(\frac{m\pi x}{L}\right)\sin\left(\frac{n\pi x}{L}\right) dx = 0 \qquad \text{for all } m, n.$$

These relations show that the set of functions $\{1, \cos(\frac{n\pi x}{L}), \sin(\frac{n\pi x}{L})\}$ forms an orthogonal basis for $L^2([-L, L])$.

4.1.2 Fourier Series Definition

If we project f onto these basis functions, we obtain its Fourier expansion. The coefficients are computed by taking the inner product of f with each basis function, exploiting their orthogonality.

Definition 4.1: Fourier Series

Let f be a $2L$–periodic function integrable on $[-L, L]$. Its Fourier series is

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty}\left(a_n \cos\frac{n\pi x}{L} + b_n \sin\frac{n\pi x}{L}\right)$$

where the coefficients are the orthogonal projections

$$a_0 = \frac{1}{L}\int_{-L}^{L} f(x)\,dx$$
$$a_n = \frac{1}{L}\int_{-L}^{L} f(x)\cos\frac{n\pi x}{L}\,dx$$
$$b_n = \frac{1}{L}\int_{-L}^{L} f(x)\sin\frac{n\pi x}{L}\,dx$$

Intuitively, each coefficient a_n (resp. b_n) measures how much of the cosine (resp. sine) wave of frequency $n\pi/L$ is present in f.

The Fourier series can be understood geometrically as projecting the function f onto the orthogonal basis of trigonometric functions. Each coefficient represents the ”component” of f in the direction of a particular basis function.

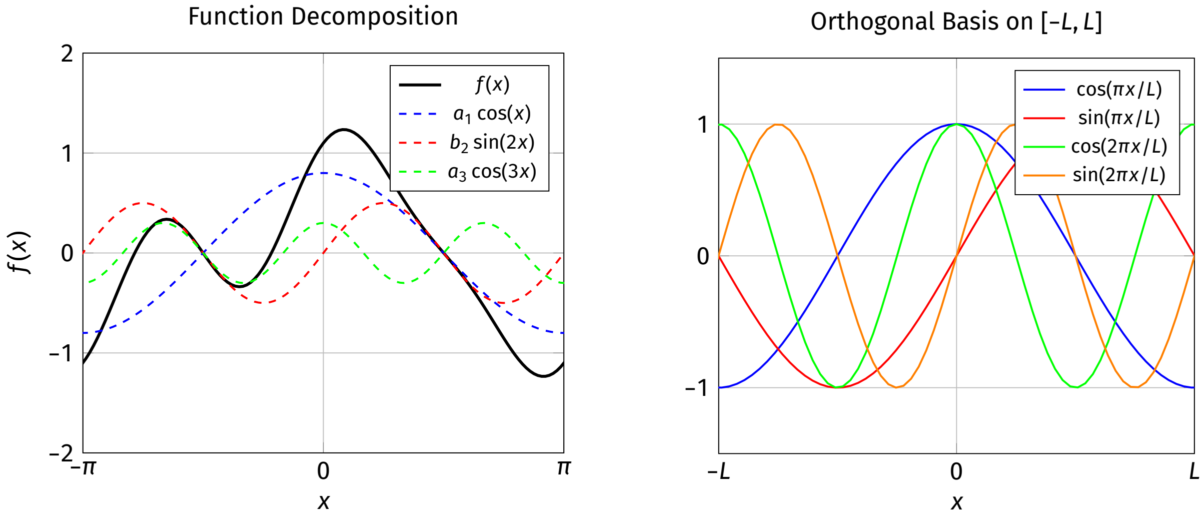


Figure 4.1: Left: Decomposition of a function $f(x)$ into its trigonometric components. Right: The orthogonal trigonometric basis functions on $[-L, L]$.

The projection formula for coefficient a_n can be visualized as:

$$a_n = \frac{\langle f, \cos(n\pi x/L)\rangle}{\|\cos(n\pi x/L)\|^2} = \frac{\int_{-L}^{L} f(x)\cos(n\pi x/L)\,dx}{\int_{-L}^{L}\cos^2(n\pi x/L)\,dx} = \frac{1}{L}\int_{-L}^{L} f(x)\cos(n\pi x/L)\,dx$$

This geometric interpretation shows that Fourier analysis is fundamentally about finding the best approximation to f in the subspace spanned by trigonometric functions, where ”best” means minimizing the L^2 error over the interval $[-L, L]$.

Definition 4.2: Complex Fourier Series

The complex exponential form of the Fourier series is:

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L}$$

where the complex coefficients are:

$$c_n = \frac{1}{2L} \int_{-L}^{L} f(x) e^{-in\pi x/L}\, dx$$

The relationship between real and complex coefficients is:

$$\begin{aligned} c_0 &= \frac{a_0}{2} \\ c_n &= \frac{a_n - ib_n}{2} \quad (n > 0) \\ c_{-n} &= \frac{a_n + ib_n}{2} = \overline{c_n} \quad (n > 0) \end{aligned}$$

4.1.3 Properties of Fourier Series

The Fourier series possesses several important properties that mirror the structure of the underlying function space.

Property 9: Properties of Fourier Series

Let $f(x)$ and $g(x)$ be $2L$-periodic functions with Fourier series, and let α, β be constants. Then:

1. **Linearity:** The Fourier series is linear:

$$\mathcal{F}[\alpha f(x) + \beta g(x)] = \alpha \mathcal{F}[f(x)] + \beta \mathcal{F}[g(x)].$$

2. **Differentiation:** If f is continuous and piecewise smooth, the Fourier series of $f'(x)$ is obtained by term-wise differentiation:

$$f'(x) = \sum_{n=1}^{\infty} \left(-a_n \frac{n\pi}{L} \sin\left(\frac{n\pi x}{L}\right) + b_n \frac{n\pi}{L} \cos\left(\frac{n\pi x}{L}\right) \right),$$

   where a_n, b_n are the Fourier coefficients of f.

3. **Integration:** The Fourier series of $f(x)$ can be integrated term by term:

$$\int_{-L}^{x} f(t)\, dt = \frac{a_0}{2}(x + L) + \sum_{n=1}^{\infty} \left[a_n \frac{L}{n\pi} \sin\left(\frac{n\pi x}{L}\right) - b_n \frac{L}{n\pi} \cos\left(\frac{n\pi x}{L}\right) + b_n \frac{L}{n\pi} \right],$$

   where the integration constant is chosen so that the result is $2L$-periodic.

Parseval's Theorem

Parseval's theorem relates the L^2 norm of a function to the sum of the squares of its Fourier coefficients, providing a powerful tool for analyzing the energy content of signals.

Theorem 4.3: Parseval's Theorem

Let f be a $2L$-periodic, square-integrable function on $[-L, L]$, with complex Fourier coefficients

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-in\pi x/L}\,dx.$$

Define the inner product and norm on $\mathrm{L}^2([-L, L])$ by

$$\langle f, g\rangle = \int_{-L}^{L} f(x)\cdot\overline{g(x)}\,dx, \qquad \|f\|^2 = \langle f, f\rangle = \int_{-L}^{L} |f(x)|^2\,dx.$$

Then Parseval's theorem states:

$$\|f\|^2 = \int_{-L}^{L} |f(x)|^2\,dx = 2L\sum_{n=-\infty}^{\infty} |c_n|^2.$$

This expresses the completeness and orthogonality of the exponential basis in $L^2([-L, L])$.

Proof. Let $f(x) = \sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L}$ be the complex Fourier series of f, converging in $L^2([-L, L])$. The coefficients are given by

$$c_n = \frac{1}{2L}\int_{-L}^{L} f(x)e^{-in\pi x/L}\,dx.$$

The exponential functions $\{e^{in\pi x/L}\}_{n\in\mathbb{Z}}$ form an orthogonal basis for $L^2([-L, L])$ with

$$\int_{-L}^{L} e^{in\pi x/L}\overline{e^{im\pi x/L}}\,dx = \int_{-L}^{L} e^{i(n-m)\pi x/L}\,dx = \begin{cases} 2L, & n = m, \\ 0, & n \neq m. \end{cases}$$

Therefore,

$$\begin{aligned}
\|f\|^2 &= \int_{-L}^{L} |f(x)|^2\,dx \\
&= \int_{-L}^{L}\left(\sum_{n=-\infty}^{\infty} c_n e^{in\pi x/L}\right)\overline{\left(\sum_{m=-\infty}^{\infty} c_m e^{im\pi x/L}\right)}\,dx \\
&= \int_{-L}^{L}\sum_n\sum_m c_n\overline{c_m}e^{i(n-m)\pi x/L}\,dx \\
&= \sum_n\sum_m c_n\overline{c_m}\int_{-L}^{L} e^{i(n-m)\pi x/L}\,dx \\
&= \sum_n\sum_m c_n\overline{c_m}\cdot 2L\,\delta_{nm} \\
&= 2L\sum_{n=-\infty}^{\infty} |c_n|^2.
\end{aligned}$$

This completes the proof. □

Gibbs Phenomenon

When approximating a function with discontinuities using Fourier series, the partial sums exhibit an overshoot near the discontinuities that does not disappear as more terms are added. This overshoot is approximately 9% of the jump discontinuity and is known as the **Gibbs phenomenon**.

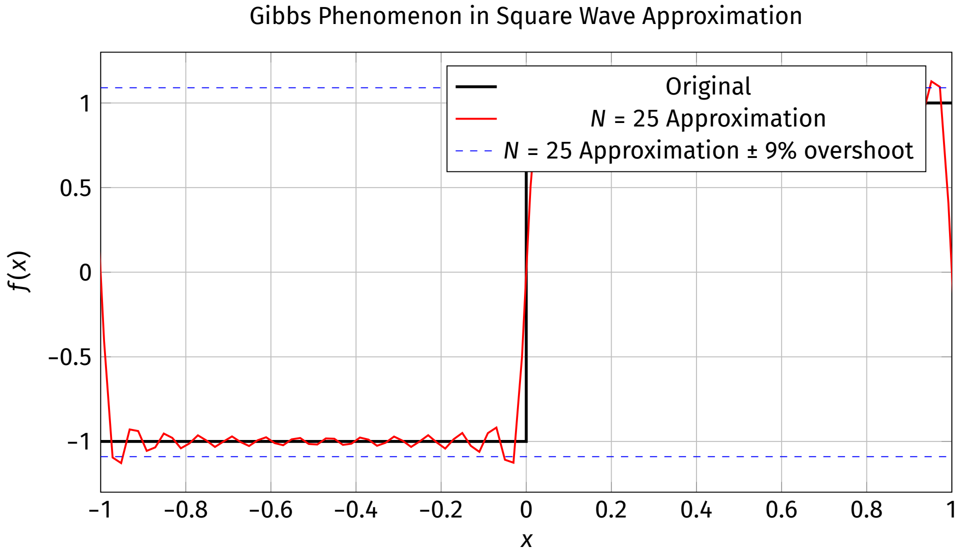


Figure 4.2: Gibbs phenomenon showing persistent overshoot near discontinuities

Convergence of Fourier Series

A natural question is: *When does the Fourier series of a function converge to the function itself?* Not every periodic function has a Fourier series that converges everywhere to the original function. The answer is provided by the **Dirichlet conditions**, which give sufficient (but not necessary) criteria for pointwise convergence of the Fourier series.

Theorem 4.4: Dirichlet Conditions for Fourier Series Convergence

Let f be a $2L$-periodic, piecewise smooth function on $\mathbb{R}$. If the following conditions hold:

1. f is absolutely integrable over one period: $\int_{-L}^{L} |f(x)|\,dx < \infty$.
2. f has only finitely many discontinuities (all of finite size) and finitely many extrema in any period.

Then, at every point x where f is continuous, its Fourier series converges to $f(x)$. At a point of discontinuity x_0, the Fourier series converges to the average of the right and left limits:

$$\lim_{N\to\infty} S_N(x_0) = \frac{f(x_0^+) + f(x_0^-)}{2}$$

where $S_N(x)$ is the N-th partial sum of the Fourier series, and $f(x_0^+)$, $f(x_0^-)$ are the right and left limits of f at x_0.

- Dirichlet conditions are sufficient (not necessary) for pointwise convergence.
- At discontinuities, the series converges to the average of left and right limits.
- Convergence is pointwise, not uniform; Gibbs phenomenon may appear near jumps.

4.1.4 Examples of Fourier Series

Example 17. Square Wave

Consider the square wave function defined on

$$f(x) = \begin{cases} 1 & \text{if } 0 < x < \pi \\ -1 & \text{if } -\pi < x < 0 \\ 0 & \text{if } x = 0, \pm\pi \end{cases}$$

Since the function is odd, all cosine terms vanish ($a_n = 0$), and we only have sine terms:

$$\begin{aligned} b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin(nx)\,dx \\ &= \frac{2}{\pi}\int_{0}^{\pi} \sin(nx)\,dx \\ &= \frac{2}{\pi}\left[-\frac{\cos(nx)}{n}\right]_0^{\pi} \\ &= \frac{2}{n\pi}(1-\cos(n\pi)) \\ &= \begin{cases} \frac{4}{n\pi} & \text{if } n \text{ is odd} \\ 0 & \text{if } n \text{ is even} \end{cases} \end{aligned}$$

Therefore, the Fourier series is:

$$f(x) = \frac{4}{\pi}\sum_{n=1,3,5,\ldots} \frac{\sin(nx)}{n} = \frac{4}{\pi}\left(\sin x + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + \cdots\right)$$

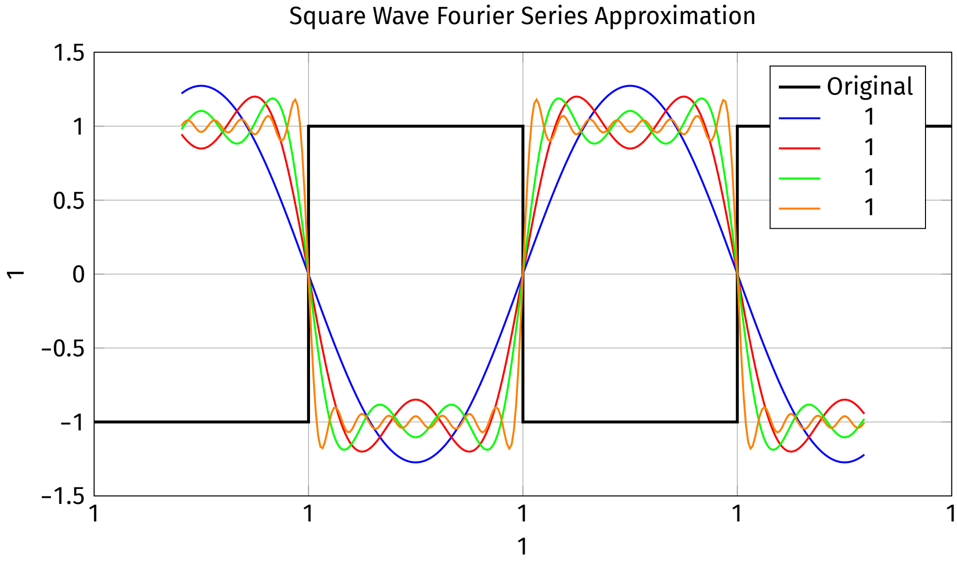


Figure 4.3: Fourier series approximation of a square wave showing convergence with increasing number of terms

Example 18. Sawtooth Wave

Consider the sawtooth wave defined on $(-\pi, \pi)$:

$$f(x) = x \quad \text{for } x \in (-\pi, \pi)$$

Since this function is odd, we have $a_n = 0$ for all n, and:

$$\begin{aligned} b_n &= \frac{1}{\pi}\int_{-\pi}^{\pi} x \sin(nx)\, dx \\ &= \frac{2}{\pi}\int_{0}^{\pi} x \sin(nx)\, dx \\ &= \frac{2(-1)^{n+1}}{n} \end{aligned}$$

The Fourier series is:

$$f(x) = 2\sum_{n=1}^{\infty} \frac{(-1)^{n+1}}{n}\sin(nx) = 2\left(\sin x - \frac{\sin 2x}{2} + \frac{\sin 3x}{3} - \cdots\right)$$

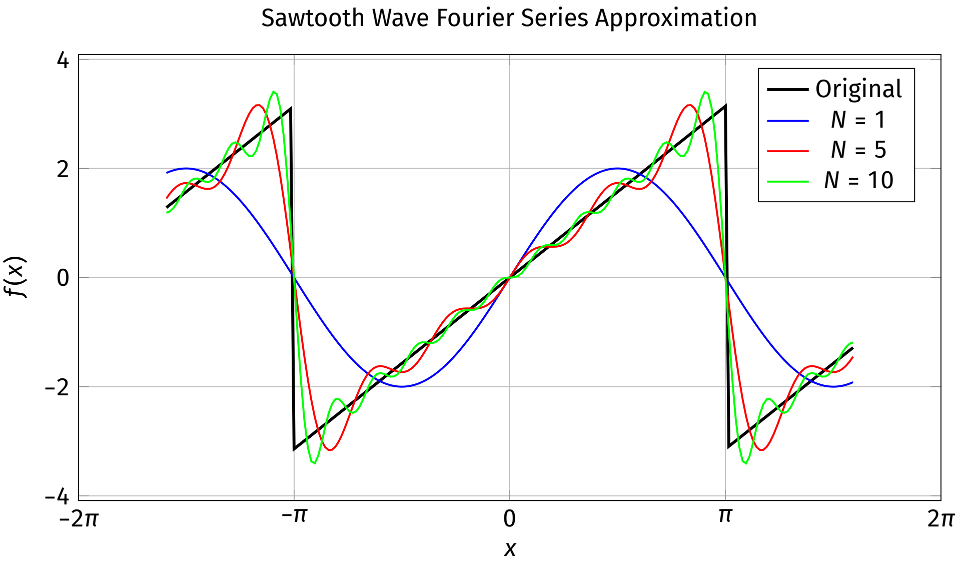


Figure 4.4: Fourier series approximation of a sawtooth wave

Chapter 5

Interpolation and Splines

Introduction and Motivation

Interpolation is a fundamental concept in numerical mathematics, where the goal is to construct a function that exactly matches a given set of data points. This is crucial in applications such as data fitting, numerical integration, computer graphics, and solving differential equations. The two main approaches are polynomial interpolation and spline interpolation, each with their own advantages and limitations.

5.1 Polynomial Interpolation

Problem Statement and Existence Given $n+1$ distinct nodes $x_0 < x_1 < \cdots < x_n$ and data values $y_0, \ldots, y_n$, the interpolation problem seeks a function $p \in \mathbb{P}_n$ (the space of polynomials of degree at most n) such that

$$p(x_i) = y_i, \quad i = 0, \ldots, n.$$

Theorem 5.1: Existence and Uniqueness of the Interpolating Polynomial

For any $n + 1$ distinct nodes $x_0, \ldots, x_n$ and data $y_0, \ldots, y_n$, there exists a unique polynomial $p \in \mathbb{P}_n$ such that $p(x_i) = y_i$ for all i.

Proof. The Vandermonde matrix $V = (x_i^j)_{i,j=0}^n$ is invertible when the x_i are distinct, so the linear system for the coefficients of p has a unique solution. □

5.1.1 Lagrange Interpolation

Lagrange interpolation provides a direct formula for the interpolating polynomial, useful when all data points are known in advance and the polynomial needs to be written explicitly.

Lagrange interpolation constructs the interpolating polynomial $p_n \in \mathbb{P}_n$ as a linear combination of specially designed basis polynomials $\ell_i(x)$, known as Lagrange basis polynomials, which are defined for each node x_i and the corresponding data value y_i.

Definition 5.2: Lagrange Interpolating Polynomial

Given $n+1$ distinct nodes $x_0, \dots, x_n$ and data values $y_0, \dots, y_n$, the *Lagrange interpolating polynomial* is the unique polynomial $p \in \mathbb{P}_n$ such that $p(x_i) = y_i$ for all i. It is given explicitly by

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

where the *Lagrange basis polynomial* $\ell_i(x)$ is defined as:

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

The Lagrange basis polynomial $\ell_i(x)$ is constructed to be 1 at x_i and 0 at all other nodes x_j ($j \neq i$), ensuring that the interpolant matches the data exactly at each node.

Properties of Lagrange Interpolation

This construction guarantees that $p(x_j) = y_j$ for each j, since $\ell_i(x_j) = \delta_{ij}$. The Lagrange form is symmetric in the data points and provides a direct formula for the interpolating polynomial, although it is not the most efficient for updating when new nodes are added.

Property 10: Properties of Lagrange Interpolation

- $\ell_i(x_j) = \delta_{ij}$ (the Kronecker delta), so $p(x_i) = y_i$.
- The interpolation is exact for all polynomials of degree at most n.
- The form is symmetric in the data points.
- Adding a new node requires recomputing all basis polynomials.

Algorithm

Algorithm 1 Lagrange Interpolation

```
Require: Nodes $x_0, \dots, x_n$; values $y_0, \dots, y_n$; evaluation point $x$
Ensure: Interpolated value $p(x)$
$p \leftarrow 0$
for $i = 0$ to $n$ do
    $\ell \leftarrow 1$
    for $j = 0$ to $n$ do
        if $j \neq i$ then
            $\ell \leftarrow \ell \cdot \frac{x - x_j}{x_i - x_j}$
    $p \leftarrow p + y_i \cdot \ell$
return $p$
```

Example 19. Lagrange Interpolation for Three Points

Given $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, the interpolating polynomial is

$$p(x) = y_0 \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} + y_1 \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} + y_2 \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}.$$

5.1.2 Newton Interpolation

Newton interpolation provides an efficient and flexible way to construct the interpolating polynomial, especially when new data points are added incrementally. The Newton form expresses the interpolant using divided differences, which can be computed recursively and updated easily.

Definition 5.3: Newton Interpolating Polynomial

Given $n+1$ distinct nodes $x_0, \dots, x_n$ and data values $y_0, \dots, y_n$, the *Newton interpolating polynomial* is the unique polynomial $p \in \mathbb{P}_n$ such that $p(x_i) = y_i$ for all i. It is given by

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

where the *divided differences* are defined recursively as

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

The Newton basis functions are nested products of the form $(x - x_0)\cdots(x - x_{k-1})$, and the coefficients are the divided differences computed from the data.

Property 11: Properties of Newton Interpolation

- **Uniqueness:** The Newton interpolating polynomial $p(x)$ is unique and belongs to $\mathbb{P}_n$, the space of polynomials of degree at most n.
- **Recursive Construction:** The divided differences $f[x_0], f[x_0, x_1], \dots, f[x_0, \dots, x_n]$ are computed recursively, allowing efficient updates when new nodes are added.
- **Basis Functions:** The polynomial $p(x)$ is expressed as a linear combination of nested basis functions:

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

  where each term involves a product of differences $(x - x_k)$ for $k = 0, \dots, i - 1$.
- **Symmetry:** The Newton form is not symmetric in the nodes, unlike the Lagrange form. The order of nodes affects the divided differences and the resulting polynomial.
- **Numerical Stability:** Newton interpolation is numerically stable for well-chosen nodes, such as Chebyshev nodes, and avoids the instability of high-degree polynomials at equally spaced nodes.
- **Error Formula:** The interpolation error at x is given by

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

  for some $\xi \in [x_0, x_n]$, where $f^{(n+1)}(\xi)$ is the $(n+1)$-th derivative of f.

Algorithm 2 Newton Interpolation (Divided Differences)

```
Require: Nodes $x_0, \dots, x_n$; values $y_0, \dots, y_n$; evaluation point $x$
Ensure: Interpolated value $p(x)$
$f[0], \dots, f[n] \leftarrow y_0, \dots, y_n$                ▷ Initialize divided differences
for $j = 1$ to $n$ do
    for $i = n$ down to $j$ do
        $f[i] \leftarrow \frac{f[i]-f[i-1]}{x_i - x_{i-j}}$
$p \leftarrow f[n]$
for $k = n - 1$ down to $0$ do
    $p \leftarrow p \cdot (x - x_k) + f[k]$
return $p$
```

Example 20. Newton Interpolation for Three Points

Given $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, the interpolating polynomial is

$$p(x) = y_0 + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1).$$

- **Pros:** Efficient for adding points; numerically stable for well-chosen nodes.
- **Cons:** Slightly more complex to write explicitly; still suffers from instability for large n and poor node choice.

5.1.3 Neville's Algorithm

Neville's algorithm is useful for evaluating the interpolating polynomial at a specific point without explicitly constructing the full polynomial. The algorithm recursively computes the interpolating polynomial at a point x using the data $(x_0, y_0), \dots, (x_n, y_n)$:

Definition 5.4: Neville's Algorithm

Given $n + 1$ distinct nodes $x_0, \dots, x_n$ and data values $y_0, \dots, y_n$, Neville's algorithm computes the value of the interpolating polynomial at a point x recursively. Define $P_{i,j}(x)$ as the value at x of the polynomial interpolating the data points $(x_{i-j}, y_{i-j}), \dots, (x_i, y_i)$. Then:

$$\begin{aligned} P_{i,0}(x) &= y_i, \quad i = 0, \dots, n, \\ P_{i,j}(x) &= \frac{(x - x_{i-j})P_{i,j-1}(x) - (x - x_i)P_{i-1,j-1}(x)}{x_i - x_{i-j}}, \quad i = j, \dots, n,\ j = 1, \dots, i. \end{aligned}$$

The final value $P_{n,n}(x)$ gives the interpolated value at x.

Property 12: Properties of Neville's Algorithm

- **Recursion:** The algorithm computes the value $P_{n,n}(x)$ recursively using the relation
$$P_{i,j}(x) = \frac{(x - x_{i-j})P_{i,j-1}(x) - (x - x_i)P_{i-1,j-1}(x)}{x_i - x_{i-j}},$$
for $i = 0, \dots, n$, $j = 1, \dots, i$, with $P_{i,0}(x) = y_i$.
- **Numerical Stability:** For a fixed evaluation point x, the recursive computation is numerically stable, as it avoids constructing the full polynomial.
- **No Explicit Polynomial:** The algorithm yields only the value $P_{n,n}(x)$ at the chosen x; it does not construct or return the explicit polynomial $p(x)$.

- **Single-Point Evaluation:** Efficient for evaluating the interpolant at a single point, but for multiple points, the computation must be repeated for each x.

Algorithm 3 Neville's Algorithm

```
Require: Nodes $x_0, \dots, x_n$; values $y_0, \dots, y_n$; evaluation point $x$
Ensure: Interpolated value $P_{n,n}(x)$
for $i = 0$ to $n$ do
    $P[i][0] \leftarrow y_i$
for $j = 1$ to $n$ do
    for $i = j$ to $n$ do
        $P[i][j] \leftarrow \frac{(x-x_{i-j})P[i][j-1]-(x-x_i)P[i-1][j-1]}{x_i-x_{i-j}}$
return $P[n][n]$
```

Example 21. Neville's Algorithm for Three Points

Given $(x_0, y_0), (x_1, y_1), (x_2, y_2)$, compute $P_{2,2}(x)$:

$$\begin{aligned}
P_{0,0}(x) &= y_0, \\
P_{1,0}(x) &= y_1, \\
P_{2,0}(x) &= y_2, \\
P_{1,1}(x) &= \frac{(x - x_0)y_1 - (x - x_1)y_0}{x_1 - x_0}, \\
P_{2,1}(x) &= \frac{(x - x_1)y_2 - (x - x_2)y_1}{x_2 - x_1}, \\
P_{2,2}(x) &= \frac{(x - x_0)P_{2,1}(x) - (x - x_2)P_{1,1}(x)}{x_2 - x_0}.
\end{aligned}$$

- **Pros:** Simple, recursive, and numerically stable for evaluating at a single point; no need to construct the full polynomial.
- **Cons:** Not efficient for evaluating at many points; does not provide an explicit polynomial form.

5.1.4 Comparison and Practical Considerations

- **Lagrange:** Simple formula, but expensive to update for new points.
- **Newton:** Efficient for adding points (nested form), numerically stable for well-chosen nodes.
- **Node choice:** Equally spaced nodes can cause large errors (Runge phenomenon); Chebyshev nodes minimize the maximum error.

5.1.5 Error Analysis of Polynomial Interpolation

Error Bound

If $f \in C^{n+1}([a, b])$, the error at x is

$$f(x) - p(x) = \frac{f^{(n+1)}(\xi)}{(n + 1)!} \prod_{i=0}^{n} (x - x_i), \quad \text{for some } \xi \in [a, b].$$

The Runge Phenomenon

A well-known issue in polynomial interpolation is the *Runge phenomenon*, which refers to the large oscillations that can occur near the endpoints of an interval when interpolating with high-degree polynomials at equally spaced nodes. This effect was first observed by Carl Runge in 1901, who considered the function

$$f(x) = \frac{1}{1 + 25x^2}, \quad x \in [-1, 1].$$

When interpolating $f(x)$ at equally spaced nodes with increasing degree n, the resulting polynomial oscillates more and more near the endpoints, and the interpolation error grows rapidly, even though the function itself is smooth. This demonstrates that simply increasing the number of nodes does not always improve the quality of polynomial interpolation.

The Runge phenomenon is particularly severe for functions with high curvature near the endpoints or when using a large number of equally spaced nodes. The error bound for polynomial interpolation,

$$|f(x) - p(x)| \leq \frac{\max_{\xi \in [a,b]} |f^{(n+1)}(\xi)|}{(n+1)!} \prod_{i=0}^{n} |x - x_i|,$$

shows that the product $\prod_{i=0}^{n} |x - x_i|$ can become very large near the endpoints for equally spaced nodes.

> **Remark 19. Chebyshev Nodes Mitigate Runge Phenomenon**
>
> Using *Chebyshev nodes*, which are clustered more densely near the endpoints, greatly reduces the oscillations and minimizes the maximum interpolation error. Chebyshev nodes on $[-1, 1]$ are given by
>
> $$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \quad i = 0, \dots, n.$$
>
> Interpolating at these nodes leads to much more stable and accurate results, even for large n.

5.2 Spline Interpolation

Spline interpolation is a method for constructing smooth curves that pass through a set of data points. Instead of using a single high-degree polynomial, which can lead to oscillations and instability, splines use low-degree polynomials on each interval between data points. These polynomial segments are joined at the data points (called knots) in a way that ensures smoothness and continuity.

Definition 5.5: Spline Interpolation

Let $a = x_0 < x_1 < \cdots < x_n = b$ be a sequence of distinct points in $\mathbb{R}$, called *knots*, and let $y_0, \dots, y_n$ be given data values. The intervals $[x_{i-1}, x_i]$, for $i = 1, \dots, n$, are called *subintervals*.
Spline interpolation of degree k seeks a function $s(x)$ such that:

- **Piecewise Polynomial:** On each subinterval $[x_{i-1}, x_i]$, $S(x)$ belongs to the space of polynomials of degree at most k, i.e.,
$$s|_{[x_{i-1},x_i]} \in \mathbb{P}_k, \quad i = 1, \dots, n.$$
- **Smoothness:** $s(x)$ and its derivatives up to order $k-1$ are continuous on $[x_0, x_n]$, that is,
$$s \in C^{k-1}([x_0, x_n]).$$
- **Interpolation:** $S(x_i) = y_i$ for all $i = 0, \dots, n$, or
$$s(x_i) = y_i, \quad i = 0, \dots, n.$$

The set of all such splines forms a vector space, often denoted by

$$S_k(\{x_i\}) = \left\{ s \in C^{k-1}([x_0, x_n]) \;\middle|\; s|_{[x_{i-1},x_i]} \in \mathbb{P}_k \text{ for } i = 1, \dots, n \right\}.$$

We also define the notion of *Spline space* $\mathcal{S}_n \subset C^{k-1}([x_0, x_n])$ as the space of all splines of degree k with knots $x_0, \dots, x_n$.

$$\mathcal{S}_n = \left\{ s \in C^{k-1}([x_0, x_n]) \;\middle|\; s|_{[x_{i-1},x_i]} \in \mathbb{P}_k,\ s(x_i) = y_i \text{ for } i = 0, \dots, n \right\}.$$

Example 22. Exam S2019: Problem 4 a)

Q: Which two properties define the linear space of splines of degree k relative to the nodes (knots) $x_0, \dots, x_n$?
A: The linear space of splines of degree k relative to the nodes (knots) $x_0, \dots, x_n$ is defined by the following two properties:

(i) **Piecewise Polynomial:** On each interval $[x_{i-1}, x_i]$, the function is a polynomial of degree at most k, i.e., $S|_{[x_{i-1},x_i]} \in \mathbb{P}_k$ for $i = 1, \dots, n$.
(ii) **Smoothness:** The function and its derivatives up to order $k-1$ are continuous on the whole interval $[x_0, x_n]$, i.e., $S \in C^{k-1}([x_0, x_n])$.

5.2.1 Piecewise Polynomial Splines

A piecewise polynomial spline is a function $S(x)$ defined on an interval $[a, b]$ that is a polynomial of degree k on each subinterval $[x_{i-1}, x_i]$ for a sequence of knots $a = x_0 < x_1 < \cdots < x_n = b$. The key requirement is that $S(x)$ and its derivatives up to order $k-1$ are continuous everywhere, ensuring a smooth overall curve. Piecewise linear splines ($k = 1$) are continuous but not smooth at the knots, while cubic splines ($k = 3$) are twice continuously differentiable and much smoother.

Definition 5.6: Piecewise Polynomial Spline

A spline of degree k with knots $x_0 < x_1 < \cdots < x_n$ is a function $S(x)$ such that:

- On each interval $[x_{i-1}, x_i]$, $S(x)$ is a polynomial of degree at most k.
- $S(x)$ and its derivatives up to order $k-1$ are continuous on $[x_0, x_n]$.

This construction allows for local control and avoids the oscillations (Runge phenomenon) seen in high-degree global polynomial interpolation. For example, piecewise linear splines ($k = 1$) are continuous but not smooth at the knots, while cubic splines ($k = 3$) are twice continuously differentiable and much smoother.

5.2.2 B-spline Basis

B-splines provide a set of basis functions for the space of splines of a given degree and knot sequence. Each B-spline basis function $B_{i,k}(x)$ is nonzero only on a small interval, giving local control over the shape of the spline. The recursive Cox–de Boor formula defines these basis functions, and their properties (local support, partition of unity, non-negativity, and smoothness) make them ideal for both theory and applications. The figure above illustrates how quadratic B-spline basis functions overlap and sum to one at every x.

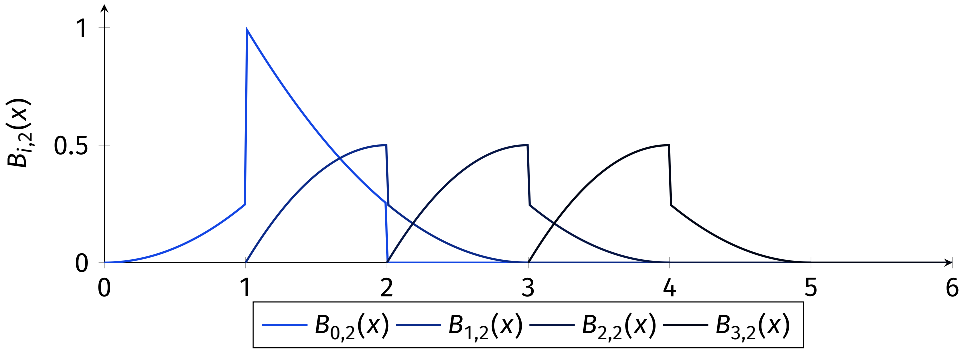


Figure 5.1: Quadratic ($k = 2$) B-spline basis functions for uniform knots $x_0 = 0, \ldots, x_6 = 6$. Each basis function is nonzero only on a small interval.

B-splines are widely used for interpolation, approximation, and geometric modeling. Their flexibility and stability make them a standard choice in computer graphics, CAD, and numerical solutions of differential equations.

5.2.3 Cubic Splines

Cubic splines are a popular method for numerical interpolation due to their smoothness and computational efficiency. They are piecewise cubic polynomials that ensure continuity of the function, its first derivative, and its second derivative across the interpolation interval.

Definition 5.7: Cubic Spline

A *cubic spline* $S(x)$ interpolating data points $(x_0, y_0), \ldots, (x_n, y_n)$ satisfies:

- $S(x_i) = y_i$ for all i (interpolates the data points).
- $S(x)$ is a cubic polynomial on each interval $[x_{i-1}, x_i]$.
- $S(x)$, $S'(x)$, and $S''(x)$ are continuous on $[x_0, x_n]$.

Additional boundary conditions are required to uniquely determine $S(x)$:

- **Natural spline:** $S''(x_0) = S''(x_n) = 0$.
- **Clamped spline:** $S'(x_0)$ and $S'(x_n)$ are specified.
- **Not-a-knot:** The third derivative is continuous at x_1 and x_{n-1}.

General Form

On each interval $[x_{i-1}, x_i]$, the cubic spline $S(x)$ is expressed as:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

where the coefficients a_i, b_i, c_i, d_i are determined by interpolation conditions, smoothness requirements, and boundary conditions.

Properties

Property 13: Properties of Cubic Splines

A cubic spline $S(x)$ interpolating $(x_0, y_0), \dots, (x_n, y_n)$ satisfies:

- **Interpolation:** $S(x_i) = y_i$ for all i.
- **Piecewise cubic:** $S(x)$ is a cubic polynomial on each interval $[x_{i-1}, x_i]$.
- **Smoothness:** $S(x)$, $S'(x)$, and $S''(x)$ are continuous on $[x_0, x_n]$.
- **Boundary conditions:** Imposed at x_0 and x_n (e.g., natural, clamped, or not-a-knot).

Construction Steps

The construction of cubic splines involves the following steps:

1. Impose interpolation conditions: $S(x_i) = y_i$ for all i.
2. Enforce continuity of $S'(x)$ and $S''(x)$ at interior knots.
3. Apply boundary conditions at x_0 and x_n (e.g., $S''(x_0) = S''(x_n) = 0$ for a natural spline).
4. Solve the resulting tridiagonal linear system for the second derivatives at the knots.
5. Use these second derivatives to compute the coefficients for each cubic polynomial piece.

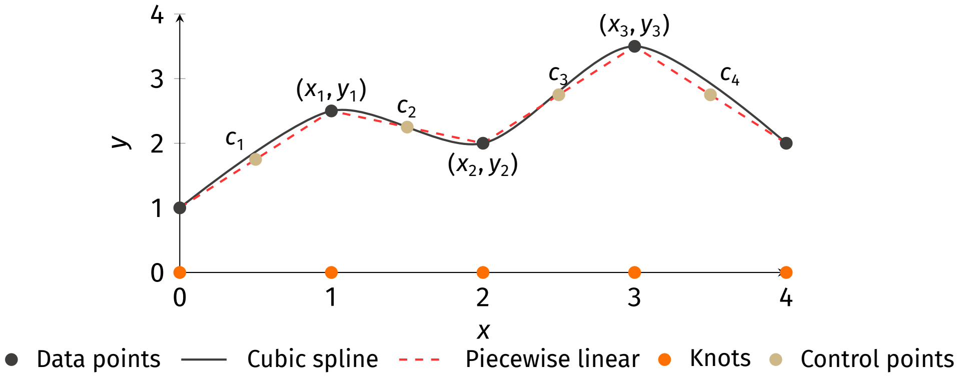


Figure 5.2: Comparison of cubic spline interpolation (solid) and piecewise linear interpolation (dashed) through five data points. The cubic spline yields a smooth curve, while the piecewise linear interpolant is only continuous.

Remark 20. Computation of Cubic Splines

The coefficients of the cubic polynomials are found by solving a tridiagonal linear system arising from the continuity and interpolation conditions.

Explicit Construction of the Cubic Spline System

Let $S(x)$ be a cubic spline interpolant for data $(x_0, y_0), \dots, (x_n, y_n)$, with $x_0 < x_1 < \cdots < x_n$. On each interval $[x_{i-1}, x_i]$, $S(x)$ can be written as:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, \quad x \in [x_{i-1}, x_i].$$

The coefficients a_i, b_i, c_i, d_i are determined by:

1. **Interpolation:** $S_i(x_{i-1}) = y_{i-1}$, $S_i(x_i) = y_i$.
2. **Smoothness:** $S_i'(x_i) = S_{i+1}'(x_i)$, $S_i''(x_i) = S_{i+1}''(x_i)$ for $i = 1, \dots, n-1$.
3. **Boundary:** Two additional conditions, e.g., $S''(x_0) = 0$ and $S''(x_n) = 0$ (natural spline).

Let $h_i = x_i - x_{i-1}$ and $m_i = S''(x_i)$. By differentiating $S_i(x)$ and matching values and derivatives at the knots, one obtains the following tridiagonal system for the second derivatives:

$$h_{i-1}m_{i-1} + 2(h_{i-1} + h_i)m_i + h_i m_{i+1} = 6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right), \quad i = 1, \dots, n-1.$$

This system, together with the boundary conditions, uniquely determines $m_0, \dots, m_n$.

Variational Characterization of Cubic Splines

Cubic splines can also be characterized as the unique minimizer of the following variational problem:

$$S = \underset{g \in C^2([x_0,x_n]),\, g(x_i)=y_i}{\arg\min} \int_{x_0}^{x_n} [g''(x)]^2 \, dx.$$

That is, among all twice continuously differentiable functions interpolating the data, the cubic spline minimizes the L^2 norm of the second derivative, making it the ”smoothest” interpolant.

Existence and Uniqueness of Cubic Splines

A fundamental result is that, given $n+1$ data points $(x_0, y_0), \dots, (x_n, y_n)$ with $x_0 < x_1 < \cdots < x_n$, and suitable boundary conditions, there exists a unique cubic spline $S \in C^2([x_0, x_n])$ such that $S(x_i) = y_i$ for all i.

Theorem 5.8: Existence and Uniqueness of Cubic Spline Interpolant

Let $x_0 < x_1 < \cdots < x_n$ and $y_0, \dots, y_n \in \mathbb{R}$. Given boundary conditions (e.g., natural, clamped, or not-a-knot), there exists a unique function $S \in C^2([x_0, x_n])$ such that:

$$\begin{aligned} &S(x_i) = y_i, \quad i = 0, \dots, n, \\ &S|_{[x_{i-1},x_i]} \text{ is a cubic polynomial for } i = 1, \dots, n, \\ &S', S'' \text{ are continuous on } [x_0, x_n]. \end{aligned}$$

Error Formula for Cubic Spline Interpolation

The error of cubic spline interpolation can be bounded in terms of the fourth derivative of the interpolated function.

Theorem 5.9: Error Bound for Cubic Spline Interpolation

Let $f \in C^4([a, b])$ and let S be the cubic spline interpolant to f at the knots $a = x_0 < x_1 < \cdots < x_n = b$. Then for $x \in [x_{i-1}, x_i]$,

$$f(x) - S(x) = -\frac{1}{384} h_i^4 f^{(4)}(\xi_x), \quad \text{for some } \xi_x \in [x_{i-1}, x_i],$$

where $h_i = x_i - x_{i-1}$.

Part III

Linear Systems

Chapter 6

Introduction to Linear Systems

6.1 Linear Systems in Numerical Mathematics

Linear systems of equations are fundamental in numerical mathematics, arising in applications from finite-difference discretisation of differential equations to least-squares data fitting.

A system is written as

$$A\mathbf{x} = \mathbf{b}, \qquad A \in \mathbb{R}^{m\times n},\ \mathbf{x} \in \mathbb{R}^{n},\ \mathbf{b} \in \mathbb{R}^{m}.$$

- *Unknown vector:* $\mathbf{x} \in \mathbb{R}^n$ collects the n variables to determine.
- *Coefficient matrix:* $A \in \mathbb{R}^{m\times n}$ encodes how the variables couple.
- *Right-hand side:* $\mathbf{b} \in \mathbb{R}^m$ represents the forcing, data, or boundary conditions.

Why linear systems matter: They appear in engineering, physics, data science, and more. Efficient and stable solution methods are essential for practical computation.

Types of linear systems:

- **Square systems** ($m = n$): Unique solution when A is invertible.
- **Over-determined** ($m > n$): More equations than unknowns; solved via least-squares.
- **Under-determined** ($m < n$): Infinitely many solutions; requires additional constraints.
- **Consistent/Inconsistent**: Solution exists iff $\mathbf{b} \in \mathrm{col}(A)$.
- **Ill-conditioned**: Large $\kappa(A) \gg 1$; sensitive to perturbations, i.e. $\delta\mathbf{b} \neq 0$ can lead to $\delta\mathbf{x} \gg 0$.
- **Well-posed**: Satisfies existence, uniqueness, and stability; otherwise ill-posed.

6.2 Solution Strategies

6.2.1 Direct Methods: Factorization Approaches

Direct methods compute the exact solution $\mathbf{x}$ (up to round-off) in a finite sequence of operations, typically via matrix factorizations such as LU, Cholesky, or QR. These are robust and accurate for small to medium-sized problems but become computationally expensive and memory-intensive for large systems.

- *Gaussian Elimination*: For dense systems ($\mathcal{O}(n^3)$)
- *LU, Cholesky*: For sparse or symmetric positive-definite (SPD) matrices
- *QR*: For least-squares and numerically stable eigenproblems

6.2.2 Iterative Methods: Stationary and Krylov Subspace Methods

Iterative methods start from an initial guess and refine the solution through repeated updates. They are especially effective for large, sparse systems where direct methods are impractical. Convergence depends on properties of A (such as symmetry and positive-definiteness) and the choice of preconditioner.

- *Stationary schemes*: Jacobi, Gauss–Seidel, Successive Over-Relaxation (SOR)
- *Krylov methods*: Conjugate Gradient (CG), GMRES, BiCGSTAB

6.2.3 Choosing a Method

- For dense $n \lesssim 10^3$ or moderately sparse *one-off* solves, direct factorizations dominate.
- When $n \gg 10^4$, memory and $\mathcal{O}(n^3)$ work become prohibitive, and well-preconditioned iterative solvers excel.

6.3 Numerical Issues in Solving Linear Systems

When solving linear systems numerically, it is crucial to understand:

- **Conditioning:** Sensitivity of the problem to perturbations.
- **Stability:** Robustness of the algorithm to errors during execution.
- **Preconditioning:** Transforming the problem to accelerate convergence.

6.3.1 Conditioning: Sensitivity to Perturbations

The *condition number* of a matrix A is

$$\kappa(A) = \|A\| \, \|A^{-1}\| \qquad (\kappa \geq 1).$$

It measures how much errors in $\mathbf{b}$ (or round-off) can be amplified in the solution $\mathbf{x}$. Small κ (well-conditioned) means the mapping $\mathbf{b} \mapsto \mathbf{x}$ is robust; large κ indicates potential magnification of errors.

Error bound: For a perturbed system $A\mathbf{x} = \mathbf{b} + \delta\mathbf{b}$, the relative error in the solution satisfies

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} \leq \kappa(A)\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}.$$

This shows that $\kappa(A)$ bounds the amplification factor from data errors to solution errors.

Definitions:

- **Ill-conditioned:** $\kappa(A)$ is large; small changes in $\mathbf{b}$ or A can cause large changes in $\mathbf{x}$.
- **Rank-deficient:** A does not have full rank; the system may have no or infinitely many solutions.
- **Ill-posed:** The problem is not well-defined or is extremely sensitive to data/errors.

6.3.2 Stability: Algorithmic Robustness

A stable method does not amplify errors beyond what is unavoidable given the problem's conditioning. Stability is a property of the algorithm, not the problem.

Forward vs. backward stability:

- *Forward stable:* The computed solution $\hat{\mathbf{x}}$ satisfies $\|\mathbf{x} - \hat{\mathbf{x}}\| \leq C\epsilon_{\text{machine}}\kappa(A)\|\mathbf{x}\|$ for some modest constant C.

- *Backward stable:* The computed solution exactly solves a nearby problem: $A\hat{\mathbf{x}} = \mathbf{b} + \delta\mathbf{b}$ with $\|\delta\mathbf{b}\| \leq C\epsilon_{\text{machine}}\|A\|\,\|\hat{\mathbf{x}}\|$.

Examples:

- *Gaussian elimination with partial pivoting:* Backward stable in practice
- *Householder QR:* Backward stable for least squares
- *Iterative refinement:* Can improve accuracy of direct solvers

6.3.3 Preconditioning: Accelerating Iterative Solvers

Preconditioning transforms $A\mathbf{x} = \mathbf{b}$ into an equivalent system with better conditioning. The most common approach is *left preconditioning*, where we solve

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b},$$

where $M \approx A$ is chosen to be easily invertible and to improve the condition number $\kappa(M^{-1}A) \ll \kappa(A)$.

Purpose: Reduce the condition number $\kappa(M^{-1}A) \ll \kappa(A)$ to accelerate convergence and improve numerical stability of iterative methods.

Implementation: The preconditioner is applied *within each iteration*, not as a preprocessing step. Iterative algorithms typically involve matrix-vector products that are preconditioned at every step.

Common preconditioners:

- *Diagonal (Jacobi):* $M = \text{diag}(A)$ — simple but limited effectiveness
- *Incomplete LU (ILU):* $M \approx LU$ with sparsity constraints
- *Multigrid:* Hierarchical approach for PDEs on grids
- *Domain decomposition:* Parallel-friendly for large problems

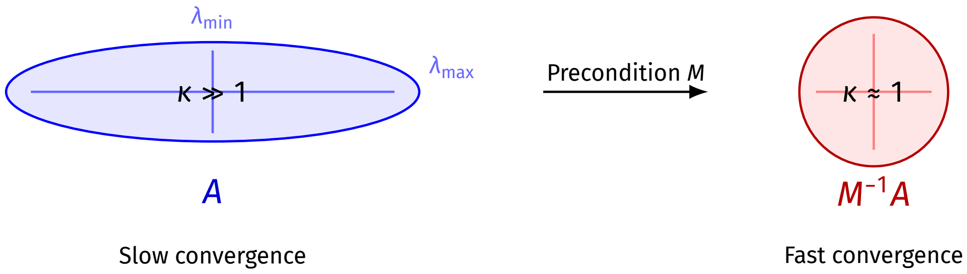


Figure 6.1: Preconditioning reshapes the transformation from elongated (high κ) to circular (low κ), dramatically improving iterative convergence rates.

Example 23. Effect of Preconditioning on Condition Number

Consider the ill-conditioned matrix

$$A = \begin{bmatrix} 4 & 3 \\ 3 & 4 \end{bmatrix}, \quad \kappa(A) = \frac{\lambda_{\max}}{\lambda_{\min}} = \frac{7}{1} = 7.$$

Using diagonal preconditioning $M = \text{diag}(A) = \text{diag}(4, 4)$, we get

$$M^{-1}A = \frac{1}{4}\begin{bmatrix} 4 & 0 \\ 0 & 4 \end{bmatrix}\begin{bmatrix} 4 & 3 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 1 & 0.75 \\ 0.75 & 1 \end{bmatrix}.$$

The eigenvalues of $M^{-1}A$ are $\{1.75, 0.25\}$, giving $\kappa(M^{-1}A) = 1.75/0.25 = 7$. While this simple example shows no improvement, more sophisticated preconditioners like ILU can dramatically reduce κ for larger, sparser systems.

Rule of thumb. Iterative solvers converge in roughly $\mathcal{O}(\sqrt{\kappa})$ iterations (CG) when a suitable preconditioner brings κ close to 1.

6.4 Summary and Practical Recommendations

- **Method selection:** Choose direct methods for small ($n \lesssim 10^3$) or moderately sized dense problems. Use iterative methods for large, sparse systems where $n \gg 10^4$.
- **Conditioning matters:** Always estimate $\kappa(A)$ before solving. If $\kappa(A) \approx 1/\epsilon_{\text{machine}}$, expect difficulties.
- **Stability is crucial:** Prefer backward stable algorithms (e.g., Householder QR, Gaussian elimination with partial pivoting).
- **Preconditioning is often essential:** For iterative methods, a good preconditioner can reduce iteration counts from thousands to dozens.
- **Sparsity structure:** Exploit problem structure (symmetry, positive-definiteness, bandedness) to choose specialized algorithms.
- **Error analysis:** Always verify your solution, especially for ill-conditioned problems. Compute residuals $\|\mathbf{b} - A\hat{\mathbf{x}}\|$ and relative errors.

Chapter 7

Direct Methods for Solving Linear Systems

7.1 Gaussian Elimination

Theorem 7.1: Gaussian Elimination

A systematic method for solving $A\mathbf{x} = \mathbf{b}$) by reducing A) to upper triangular form, then using back substitution.

7.1.1 LU Decomposition

Definition and Structure

LU decomposition factors a nonsingular matrix $A \in \mathbb{R}^{n\times n}$) into the product of a unit lower-triangular matrix L) and an upper-triangular matrix U):

$$A = LU, \quad L = \begin{bmatrix} 1 & & & 0 \\ \ell_{21} & 1 & & \\ \vdots & \ddots & \ddots & \\ \ell_{n1} & \cdots & \ell_{n,n-1} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \cdots & 0 & u_{nn} \end{bmatrix}.$$

Solving Linear Systems with LU

Definition 7.2: Solving Linear Systems with LU Decomposition

Given $A = LU$), we can solve the linear system $A\mathbf{x} = \mathbf{b}$) in two steps:

$$\begin{aligned} L\mathbf{y} = \mathbf{b} \quad &\Rightarrow \quad \mathbf{y} = L^{-1}\mathbf{b} \\ U\mathbf{x} = \mathbf{y} \quad &\Rightarrow \quad \mathbf{x} = U^{-1}\mathbf{y} \end{aligned}$$

Pivoting for Stability

To improve numerical stability, partial pivoting is often applied. One introduces a permutation matrix P) such that

$$PA = LU,$$

where at each step k) one swaps row k) with the row $p \geq k$) maximizing $|a_{pk}|$).

7.2 Cholesky Decomposition

7.2.1 Definition and Structure

Cholesky decomposition factors a symmetric positive-definite matrix $A \in \mathbb{R}^{n\times n}$) into the product of a lower-triangular matrix and its transpose:

$$A = LL^T, \quad L = \begin{bmatrix} \ell_{11} & & & 0 \\ \ell_{21} & \ell_{22} & & \\ \vdots & \ddots & \ddots & \\ \ell_{n1} & \cdots & \ell_{n,n-1} & \ell_{nn} \end{bmatrix},$$

where each diagonal entry $\ell_{kk} > 0$).

Example 24. LU factorization and solution of $A\mathbf{x} = \mathbf{b}$

Given

$$A = \begin{bmatrix} -2 & 2 & 1 & 0 \\ -2 & 0 & 2 & 1 \\ 0 & 2 & -2 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

We seek $A = LU$ with L unit lower-triangular and U upper-triangular, and then solve $A\mathbf{x} = \mathbf{b}$.

Step 1: LU factorization (no pivoting)

Let

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \ell_{21} & 1 & 0 & 0 \\ \ell_{31} & \ell_{32} & 1 & 0 \\ \ell_{41} & \ell_{42} & \ell_{43} & 1 \end{bmatrix}, \quad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}$$

First row:

$$\begin{aligned} u_{11} &= -2, \qquad u_{12} = 2, \qquad u_{13} = 1, \qquad u_{14} = 0 \\ \ell_{21} &= \frac{a_{21}}{u_{11}} = \frac{-2}{-2} = 1 \\ \ell_{31} &= \frac{a_{31}}{u_{11}} = \frac{0}{-2} = 0 \\ \ell_{41} &= \frac{a_{41}}{u_{11}} = \frac{0}{-2} = 0 \end{aligned}$$

Second row:

$$\begin{aligned} u_{22} &= a_{22} - \ell_{21}u_{12} = 0 - (1)(2) = -2 \\ u_{23} &= a_{23} - \ell_{21}u_{13} = 2 - (1)(1) = 1 \\ u_{24} &= a_{24} - \ell_{21}u_{14} = 1 - (1)(0) = 1 \\ \ell_{32} &= \frac{a_{32} - \ell_{31}u_{12}}{u_{22}} = \frac{2 - (0)(2)}{-2} = -1 \\ \ell_{42} &= \frac{a_{42} - \ell_{41}u_{12}}{u_{22}} = \frac{0 - (0)(2)}{-2} = 0 \end{aligned}$$

Third row:

$$\begin{aligned} u_{33} &= a_{33} - \ell_{31}u_{13} - \ell_{32}u_{23} = -2 - (0)(1) - (-1)(1) = -2 + 1 = -1 \\ u_{34} &= a_{34} - \ell_{31}u_{14} - \ell_{32}u_{24} = 0 - (0)(0) - (-1)(1) = 0 + 1 = 1 \\ \ell_{43} &= \frac{a_{43} - \ell_{41}u_{13} - \ell_{42}u_{23}}{u_{33}} = \frac{1 - (0)(1) - (0)(1)}{-1} = \frac{1}{-1} = -1 \end{aligned}$$

Fourth row:

$$\begin{aligned} u_{44} &= a_{44} - \ell_{41}u_{14} - \ell_{42}u_{24} - \ell_{43}u_{34} \\ &= 0 - (0)(0) - (0)(1) - (-1)(1) = 0 + 1 = 1 \end{aligned}$$

So,

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \end{bmatrix}, \quad U = \begin{bmatrix} -2 & 2 & 1 & 0 \\ 0 & -2 & 1 & 1 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 2: Solve $L\mathbf{y} = \mathbf{b}$ (forward substitution)

$$\begin{cases} y_1 = -1 \\ y_2 = 1 - 1 \cdot y_1 = 1 - (-1) = 2 \\ y_3 = 1 - (-1) \cdot y_2 = 1 + 2 = 3 \\ y_4 = 1 - (-1) \cdot y_3 = 1 + 3 = 4 \end{cases} \quad \Rightarrow \quad \mathbf{y} = \begin{bmatrix} -1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Step 3: Solve $U\mathbf{x} = \mathbf{y}$ (back substitution)

$$\begin{cases} x_4 = \frac{y_4}{u_{44}} = \frac{4}{1} = 4 \\ x_3 = \frac{y_3 - u_{34}x_4}{u_{33}} = \frac{3-1\cdot 4}{-1} = \frac{-1}{-1} = 1 \\ x_2 = \frac{y_2 - u_{23}x_3 - u_{24}x_4}{u_{22}} = \frac{2-1\cdot 1-1\cdot 4}{-2} = \frac{2-1-4}{-2} = \frac{-3}{-2} = \frac{3}{2} \\ x_1 = \frac{y_1 - u_{12}x_2 - u_{13}x_3 - u_{14}x_4}{u_{11}} = \frac{-1-2\cdot\frac{3}{2}-1\cdot 1-0\cdot 4}{-2} = \frac{-1-3-1}{-2} = \frac{-5}{-2} = \frac{5}{2} \end{cases}$$

So the solution is $\mathbf{x} = \begin{bmatrix} 5/2 & 3/2 & 1 & 4 \end{bmatrix}^\mathsf{T}$ **Step 4: Compute the condition number $\kappa_\infty(A)$**
Recall that the condition number in the infinity norm is defined as

$$\kappa_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty.$$

From the given information, $\|A^{-1}\mathbf{b}\|_\infty = \|A^{-1}\|_\infty \|\mathbf{b}\|_\infty$.
Step 4: Compute the condition number $\kappa_\infty(A)$
First, compute the infinity norm of A:

$$\begin{aligned} \|A\|_\infty &= \max_{1\le i\le 4} \sum_{j=1}^{4} |a_{ij}| \\ &= \max\{|-2| + |2| + |1| + |0|,\ |-2| + |0| + |2| + |1|,\ |0| + |2| + |-2| + |0|,\ |0| + |0| + |1| + |0|\} \\ &= \max\{5, 5, 4, 1\} = 5 \end{aligned}$$

Next, compute $\|\mathbf{b}\|_\infty = \max\{| - 1|, 1, 1, 1\} = 1$.
From the solution $\mathbf{x} = (5/2,\ 3/2,\ 1,\ 4)^\mathsf{T}$, we have

$$\|A^{-1}\mathbf{b}\|_\infty = \|\mathbf{x}\|_\infty = 4$$

Therefore,

$$\|A^{-1}\|_\infty = \frac{\|A^{-1}\mathbf{b}\|_\infty}{\|\mathbf{b}\|_\infty} = \frac{4}{1} = 4$$

and

$$\kappa_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty = 5 \cdot 4 = 20$$

So, the condition number is $\boxed{20}$.

7.2.2 Cholesky Algorithm

An in-place Cholesky factorization proceeds as:

Algorithm 4 Cholesky Decomposition

```
Require: $A \in \mathbb{R}^{n\times n}$) symmetric positive-definite
Ensure: $L$) lower-triangular with $\ell_{kk} > 0$), such that $A = LL^T$)
for $k = 1$) to $n$) do
    $\ell_{kk} \leftarrow \sqrt{a_{kk} - \sum_{p=1}^{k-1} \ell_{kp}^2}$)
    for $i = k + 1$) to $n$) do
        $\ell_{ik} \leftarrow \left(a_{ik} - \sum_{p=1}^{k-1} \ell_{ip}\,\ell_{kp}\right)/\ell_{kk}$)
```

7.2.3 Complexity and Stability

The Cholesky algorithm requires $\frac{1}{3}n^3 + O(n^2)$) operations. For symmetric positive-definite matrices, it is numerically stable without the need for pivoting.

Example 25. Cholesky factorization $A = LL^T$) of a 3×3) matrix

$$\underbrace{\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} \ell_{11} & 0 & 0 \\ \ell_{21} & \ell_{22} & 0 \\ \ell_{31} & \ell_{32} & \ell_{33} \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} \ell_{11} & \ell_{21} & \ell_{31} \\ 0 & \ell_{22} & \ell_{32} \\ 0 & 0 & \ell_{33} \end{bmatrix}}_{L^T}$$

7.3 Givens Rotation

7.3.1 Motivation and Overview

Givens rotations are used to introduce zeros into vectors or matrices, particularly in the context of QR decomposition and solving linear systems. Unlike Gaussian elimination, which uses row operations, Givens rotations use orthogonal transformations to zero out specific elements, preserving numerical stability and orthogonality.

7.3.2 Definition of a Givens Rotation

A Givens rotation is a plane rotation that zeroes out a specific entry in a vector or matrix while preserving orthogonality. The rotation matrix G is an $n \times n$ identity matrix, modified in the (i, j)-plane by the 2×2 block:

$$G_{[i,j]} = \begin{bmatrix} c & s \\ -s & c \end{bmatrix},$$

where $c = \cos\theta$ and $s = \sin\theta$.

7.3.3 Constructing a Givens Rotation

Given a vector $\mathbf{x} = (a, b)^T$, we wish to find c and s such that

$$\begin{bmatrix} c & s \\ -s & c \end{bmatrix}\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix},$$

where $r = \sqrt{a^2 + b^2}$. The parameters are chosen as

$$c = \frac{a}{\sqrt{a^2 + b^2}}, \qquad s = \frac{b}{\sqrt{a^2 + b^2}}.$$

7.3.4 Application to Matrices

To zero out the j-th entry of a column vector (or matrix), apply the Givens rotation to rows i and j:

- For QR decomposition, Givens rotations are applied successively to introduce zeros below the diagonal, transforming a matrix A into an upper triangular matrix R.
- The product of all Givens rotations forms an orthogonal matrix Q such that $A = QR$.

7.3.5 Recipe: Applying Givens Rotations for QR Decomposition

To perform QR decomposition of a matrix $A \in \mathbb{R}^{m \times n}$ using Givens rotations:

1. For each column j from 1 to n:
   (a) For each row i from m down to $j + 1$:
      i. If $A_{i,j} \neq 0$, construct a Givens rotation $G(i, j, \theta)$ to zero $A_{i,j}$:
         - Let $a = A_{j,j}$, $b = A_{i,j}$.
         - Compute $r = \sqrt{a^2 + b^2}$.
         - Set $c = a/r$, $s = -b/r$.

      ii. Apply the rotation to rows j and i for columns j to n:

$$\begin{bmatrix} A_{j,k} \\ A_{i,k} \end{bmatrix} \leftarrow \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} A_{j,k} \\ A_{i,k} \end{bmatrix}$$

2. After all rotations, A is upper triangular (R factor).
3. (Optionally) Accumulate the product of all Givens rotations to form the orthogonal matrix Q.

Example 26. QR decomposition of a 4×3 matrix using Givens rotations

Let

$$A = \begin{bmatrix} 6 & 5 & 0 \\ 5 & 1 & 4 \\ 3 & 4 & 2 \\ 2 & 3 & 1 \end{bmatrix}$$

We will zero entries below the diagonal in each column using Givens rotations.

Step 1: Column 1

- Zero $A_{4,1}$ ($a = 6$, $b = 2$):

$$r_1 = \sqrt{6^2 + 2^2} = 2\sqrt{10}, \quad c_1 = 6/r_1 = 3/\sqrt{10}, \quad s_1 = -2/r_1 = -1/\sqrt{10}$$

Apply to rows 1 and 4 for columns 1–3:

$$\begin{bmatrix} A_{1,k} \\ A_{4,k} \end{bmatrix} \leftarrow \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} A_{1,k} \\ A_{4,k} \end{bmatrix}$$

For $k = 1$:

$$\begin{bmatrix} 6 \\ 2 \end{bmatrix} \to \begin{bmatrix} 2\sqrt{10} \\ 0 \end{bmatrix}$$

For $k = 2$:

$$\begin{bmatrix}5\\3\end{bmatrix} \to \begin{bmatrix}c_1 \cdot 5 - s_1 \cdot 3\\ s_1 \cdot 5 + c_1 \cdot 3\end{bmatrix} = \begin{bmatrix}\frac{3}{\sqrt{10}} \cdot 5 - \left(-\frac{1}{\sqrt{10}}\right) \cdot 3\\ -\frac{1}{\sqrt{10}} \cdot 5 + \frac{3}{\sqrt{10}} \cdot 3\end{bmatrix} = \begin{bmatrix}\frac{15+3}{\sqrt{10}}\\ \frac{-5+9}{\sqrt{10}}\end{bmatrix} = \begin{bmatrix}\frac{18}{\sqrt{10}}\\ \frac{4}{\sqrt{10}}\end{bmatrix}$$

For $k = 3$:

$$\begin{bmatrix}0\\1\end{bmatrix} \to \begin{bmatrix}c_1 \cdot 0 - s_1 \cdot 1\\ s_1 \cdot 0 + c_1 \cdot 1\end{bmatrix} = \begin{bmatrix}-\frac{1}{\sqrt{10}}\\ \frac{3}{\sqrt{10}}\end{bmatrix}$$

The updated matrix after this rotation:

$$A^{(1)} = \begin{bmatrix}2\sqrt{10} & \frac{18}{\sqrt{10}} & -\frac{1}{\sqrt{10}}\\ 5 & 1 & 4\\ 3 & 4 & 2\\ 0 & \frac{4}{\sqrt{10}} & \frac{3}{\sqrt{10}}\end{bmatrix}$$

- Zero $A_{3,1}$ ($a = 2\sqrt{10}$, $b = 3$):

$$r_2 = \sqrt{(2\sqrt{10})^2 + 3^2} = \sqrt{40 + 9} = \sqrt{49} = 7$$

$$c_2 = \frac{2\sqrt{10}}{7}, \quad s_2 = -\frac{3}{7}$$

Apply to rows 1 and 3 for columns 1–3:

$$\begin{bmatrix}2\sqrt{10}\\3\end{bmatrix} \to \begin{bmatrix}7\\0\end{bmatrix}$$

For $k = 2$:

$$\begin{bmatrix}\frac{18}{\sqrt{10}}\\ 4\end{bmatrix} \to \begin{bmatrix}c_2 \cdot \frac{18}{\sqrt{10}} - s_2 \cdot 4\\ s_2 \cdot \frac{18}{\sqrt{10}} + c_2 \cdot 4\end{bmatrix}$$

For $k = 3$:

$$\begin{bmatrix}-\frac{1}{\sqrt{10}}\\ 2\end{bmatrix} \to \begin{bmatrix}c_2 \cdot \left(-\frac{1}{\sqrt{10}}\right) - s_2 \cdot 2\\ s_2 \cdot \left(-\frac{1}{\sqrt{10}}\right) + c_2 \cdot 2\end{bmatrix}$$

The updated matrix after this rotation:

$$A^{(2)} = \begin{bmatrix}7 & a_{12}^{(2)} & a_{13}^{(2)}\\ 5 & 1 & 4\\ 0 & a_{32}^{(2)} & a_{33}^{(2)}\\ 0 & \frac{4}{\sqrt{10}} & \frac{3}{\sqrt{10}}\end{bmatrix}$$

(where $a_{12}^{(2)}, a_{13}^{(2)}, a_{32}^{(2)}, a_{33}^{(2)}$ are computed as above.)

- Zero $A_{2,1}$ ($a = 7$, $b = 5$):

$$r_3 = \sqrt{7^2 + 5^2} = \sqrt{49 + 25} = \sqrt{74}$$

$$c_3 = 7/\sqrt{74}, \quad s_3 = -5/\sqrt{74}$$

Apply to rows 1 and 2 for columns 1–3:

$$\begin{bmatrix} 7 \\ 5 \end{bmatrix} \to \begin{bmatrix} \sqrt{74} \\ 0 \end{bmatrix}$$

For $k = 2, 3$ update similarly.
The matrix after column 1 is upper triangular in column 1:

$$A^{(3)} = \begin{bmatrix} \sqrt{74} & a_{12}^{(3)} & a_{13}^{(3)} \\ 0 & a_{22}^{(3)} & a_{23}^{(3)} \\ 0 & a_{32}^{(3)} & a_{33}^{(3)} \\ 0 & a_{42}^{(3)} & a_{43}^{(3)} \end{bmatrix}$$

Step 2: Column 2

- Zero $A_{4,2}$ ($a = a_{22}^{(3)}$, $b = a_{42}^{(3)}$):

$$r_4 = \sqrt{(a_{22}^{(3)})^2 + (a_{42}^{(3)})^2}, \quad c_4 = a_{22}^{(3)}/r_4, \quad s_4 = -a_{42}^{(3)}/r_4$$

Apply to rows 2 and 4 for columns 2–3.

- Zero $A_{3,2}$ ($a = a_{22}^{(3)}$, $b = a_{32}^{(3)}$):

$$r_5 = \sqrt{(a_{22}^{(3)})^2 + (a_{32}^{(3)})^2}, \quad c_5 = a_{22}^{(3)}/r_5, \quad s_5 = -a_{32}^{(3)}/r_5$$

Apply to rows 2 and 3 for columns 2–3.
The matrix after column 2:

$$A^{(5)} = \begin{bmatrix} \sqrt{74} & a_{12}^{(3)} & a_{13}^{(3)} \\ 0 & r_5 & a_{23}^{(5)} \\ 0 & 0 & a_{33}^{(5)} \\ 0 & 0 & a_{43}^{(5)} \end{bmatrix}$$

Step 3: Column 3

- Zero $A_{4,3}$ ($a = a_{33}^{(5)}$, $b = a_{43}^{(5)}$):

$$r_6 = \sqrt{(a_{33}^{(5)})^2 + (a_{43}^{(5)})^2}, \quad c_6 = a_{33}^{(5)}/r_6, \quad s_6 = -a_{43}^{(5)}/r_6$$

Apply to rows 3 and 4 for column 3.
The final matrix is upper triangular:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ 0 & r_{22} & r_{23} \\ 0 & 0 & r_{33} \\ 0 & 0 & 0 \end{bmatrix}$$

where each r_{ij} is computed during the process.

Summary: Each Givens rotation zeros a subdiagonal entry, and the explicit formulas for c and s are given at each step. The matrix is updated after each rotation, and all intermediate matrices can be written out as above. The process continues until all entries below the diagonal are zero, yielding the R factor.

7.3.6 Algorithm: QR Decomposition via Givens Rotations

Algorithm 5 Givens-Rotation QR Decomposition

```
Require: $A \in \mathbb{R}^{m\times n}$
Ensure: Upper triangular $R$ in place of $A$; optionally, orthogonal $Q$ accumulates
function GivensQR($A$)
    $(m, n) \leftarrow$ size($A$)
    for $j \leftarrow 1$ to $n$ do
        for $i \leftarrow m$ down to $j+1$ do
            $a \leftarrow A_{j,j}$, $b \leftarrow A_{i,j}$
            if $b \neq 0$ then
                $r \leftarrow \sqrt{a^2 + b^2}$
                $c \leftarrow a/r$, $s \leftarrow b/r$
                ApplyGivens($A$, $j$, $i$, $c$, $s$, $j : n$)
    return $A$                                          ▷ now contains $R$
function ApplyGivens($M, i, k, c, s, \ell : u$)
    for $j \leftarrow \ell$ to $u$ do
        $t_i \leftarrow c\,M_{i,j} + s\,M_{k,j}$
        $t_k \leftarrow -s\,M_{i,j} + c\,M_{k,j}$
        $M_{i,j} \leftarrow t_i$, $M_{k,j} \leftarrow t_k$
```

Algorithm 6 QR by Hand-Friendly Givens Rotations

```
Require: Square or tall matrix $A \in \mathbb{R}^{m\times n}$        ▷ $m \geq n$ is typical
Ensure: $A$ overwritten by upper-triangular $R$
for $j \leftarrow 1$ to $n$ do                                  ▷ work column by column
    for $i \leftarrow m$ down to $j+1$ do                       ▷ zero entries below the pivot
        $a \leftarrow A_{j,j}$,  $b \leftarrow A_{i,j}$                   ▷ the 2-vector to rotate
        if $b \neq 0$ then                                      ▷ nothing to do if already zero
            $r \leftarrow \sqrt{a^2 + b^2}$                               ▷ Euclidean norm
            $c \leftarrow a/r$,  $s \leftarrow -b/r$                      ▷ $G(a,b)\,[a\ b]^T = [r\ 0]^T$
            for $k \leftarrow j$ to $n$ do                      ▷ apply to remaining cols
                $u \leftarrow A_{j,k}$,  $v \leftarrow A_{i,k}$
                $A_{j,k} \leftarrow c\,u - s\,v$
                $A_{i,k} \leftarrow s\,u + c\,v$
return $A$                                              ▷ $A$ now *is* the $R$ factor
```

7.3.7 Advantages and Use Cases

- Givens rotations are especially efficient for sparse matrices, as each rotation affects only two rows.
- They are numerically stable due to the use of orthogonal transformations.
- Widely used in least squares problems, QR decomposition, and iterative methods for eigenvalue problems.

Example 27. Givens rotation zeroing an entry

Let $\mathbf{x} = (3, 4)^T$. To zero the second entry:

$$r = \sqrt{3^2 + 4^2} = 5, \quad c = 3/5, \quad s = 4/5.$$

> The Givens rotation matrix is
>
> $$G = \begin{bmatrix} 3/5 & 4/5 \\ -4/5 & 3/5 \end{bmatrix},$$
>
> and
>
> $$G \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \end{bmatrix}.$$

Chapter 8

Iterative Methods for Solving Linear Systems

Iterative methods provide an alternative approach to direct methods for solving linear systems

$$A\mathbf{x} = \mathbf{b},$$

where $A \in \mathbb{R}^{n \times n}$ is nonsingular and $\mathbf{b} \in \mathbb{R}^n$. Instead of computing the exact solution in a finite number of steps, iterative methods generate a sequence of approximations

$$\mathbf{x}^{(0)}, \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots \to \mathbf{x}^* = A^{-1}\mathbf{b}$$

starting from an initial guess $\mathbf{x}^{(0)}$.

8.1 When to Use Iterative Methods

Iterative methods are particularly advantageous in the following scenarios:

Large Sparse Systems For sparse matrices with $\mathcal{O}(n)$ nonzero entries, direct methods may suffer from *fill-in* during factorization, requiring $\mathcal{O}(n^2)$ or more storage and computation. Iterative methods can exploit sparsity by only requiring matrix-vector products, maintaining the sparse structure.

Computational Complexity Each iteration typically costs $\mathcal{O}(n^2)$ operations for dense matrices (or $\mathcal{O}(\text{nnz}(A))$ for sparse matrices), compared to $\mathcal{O}(n^3)$ for direct methods. If convergence occurs in $k \ll n$ iterations, iterative methods can be significantly faster.

Memory Requirements Iterative methods generally require only $\mathcal{O}(n)$ storage beyond the matrix itself, while direct methods may need $\mathcal{O}(n^2)$ for factorization storage.

Preconditioning Iterative methods can be enhanced with *preconditioning* to accelerate convergence, especially for ill-conditioned systems.

8.2 Error and Residual

Since the exact solution $\mathbf{x}^*$ is unknown, we cannot directly compute the error

$$\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*.$$

Instead, we monitor the *residual* as a practical convergence indicator:

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}.$$

The residual measures how well the current iterate satisfies the original system. We have $\mathbf{r}^{(k)} = \mathbf{0}$ if and only if $\mathbf{x}^{(k)}$ is the exact solution. Common stopping criteria include:

$$\begin{aligned} \|\mathbf{r}^{(k)}\| &< \varepsilon \quad \text{(absolute residual)} \\ \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{r}^{(0)}\|} &< \varepsilon \quad \text{(relative residual)} \\ \frac{\|\mathbf{r}^{(k)}\|}{\|\mathbf{b}\|} &< \varepsilon \quad \text{(normalized residual)} \end{aligned}$$

for some tolerance $\varepsilon > 0$.

8.3 General Framework for Iterative Methods

Most iterative methods can be written in the general form

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f}, \tag{8.1}$$

where $B \in \mathbb{R}^{n\times n}$ is the *iteration matrix* and $\mathbf{f} \in \mathbb{R}^n$ is derived from $\mathbf{b}$.

Definition 8.1: Consistent Iteration Method

An iterative method of the form (8.1) is *consistent* with $A\mathbf{x} = \mathbf{b}$ if the exact solution $\mathbf{x}^*$ satisfies

$$\mathbf{x}^* = B\mathbf{x}^* + \mathbf{f}.$$

This is equivalent to requiring $\mathbf{f} = (I - B)A^{-1}\mathbf{b}$.

8.3.1 Convergence Theory

Theorem 8.2: Convergence Criterion for Linear Iterative Methods

Consider an iterative method of the form (8.1). The method converges to the exact solution for any initial guess $\mathbf{x}^{(0)}$ if and only if

$$\rho(B) < 1,$$

where $\rho(B)$ is the spectral radius of the iteration matrix B.

Proof. The error at iteration k satisfies $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}^*$. From consistency and the iteration formula:

$$\begin{aligned} \mathbf{e}^{(k+1)} &= \mathbf{x}^{(k+1)} - \mathbf{x}^* \\ &= B\mathbf{x}^{(k)} + \mathbf{f} - (B\mathbf{x}^* + \mathbf{f}) \\ &= B(\mathbf{x}^{(k)} - \mathbf{x}^*) = B\mathbf{e}^{(k)} \end{aligned}$$

> By induction, $\mathbf{e}^{(k)} = B^k \mathbf{e}^{(0)}$. The sequence $\{\mathbf{e}^{(k)}\}$ converges to zero for all initial errors if and only if $\lim_{k\to\infty} B^k = 0$, which holds if and only if $\rho(B) < 1$. □

This fundamental result shows that convergence depends entirely on the spectral radius of the iteration matrix, regardless of the specific iterative method used.

8.4 Matrix Splittings and Linear Iterative Methods

A systematic way to construct iterative methods is through *matrix splittings*. We write

$$A = P - N,$$

where P is nonsingular and easy to invert, and $N = P - A$. The matrix P is called the *preconditioner* or *preconditioning matrix*.

Starting with $A\mathbf{x} = \mathbf{b}$, we rearrange as:

$$P\mathbf{x} = N\mathbf{x} + \mathbf{b}$$

This suggests the iterative scheme:

$$P\mathbf{x}^{(k+1)} = N\mathbf{x}^{(k)} + \mathbf{b}, \quad k \geq 0. \tag{8.2}$$

Equivalently, this can be written in residual form as:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + P^{-1}\mathbf{r}^{(k)},$$

where $\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}$ is the residual.

From (8.2), the iteration matrix and fixed vector are:

$$\begin{aligned} B &= P^{-1}N = P^{-1}(P - A) = I - P^{-1}A \\ \mathbf{f} &= P^{-1}\mathbf{b} \end{aligned}$$

Key Insight The choice of P determines the iterative method:

- P should be easy to invert (to solve $P\mathbf{x}^{(k+1)} = \cdots$ efficiently)
- P should approximate A well (to ensure rapid convergence)
- These goals are often conflicting, requiring careful balance

8.4.1 Matrix Decomposition Notation

For convenience, we decompose A into its diagonal, lower triangular, and upper triangular parts:

$$A = D + L + U,$$

where

$$D_{ij} = \begin{cases} a_{ij} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

$$L_{ij} = \begin{cases} a_{ij} & \text{if } i > j \\ 0 & \text{otherwise} \end{cases}$$

$$U_{ij} = \begin{cases} a_{ij} & \text{if } i < j \\ 0 & \text{otherwise} \end{cases}$$

Classical stationary methods correspond to specific choices of P in terms of D, L, and U.

8.5 Stationary Iterative Methods

Stationary iterative methods are those where the iteration matrix B and vector $\mathbf{f}$ remain constant throughout the iteration process. These form the foundation for understanding more advanced methods.

Definition 8.3: Stationary Iterative Method

A stationary iterative method for solving $A\mathbf{x} = \mathbf{b}$ has the form

$$\mathbf{x}^{(k+1)} = B\mathbf{x}^{(k)} + \mathbf{f},$$

where B and $\mathbf{f}$ are fixed matrices/vectors independent of the iteration index k.

We now examine several important stationary methods, each corresponding to different choices of the preconditioner P in the splitting $A = P - N$.

8.5.1 Richardson Iteration

The Richardson iteration is the simplest stationary method and provides excellent intuition for the general approach.

Motivation and Derivation

The basic idea is straightforward: if $A\mathbf{x} = \mathbf{b}$, then for any parameter $\omega \neq 0$:

$$\omega(A\mathbf{x} - \mathbf{b}) = \mathbf{0}$$

This suggests an iterative correction scheme:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \omega(A\mathbf{x}^{(k)} - \mathbf{b})$$

We can view this as a fixed-point iteration. Rearranging the exact equation:

$$\mathbf{x}^* = \mathbf{x}^* - \omega(A\mathbf{x}^* - \mathbf{b}) \qquad = (I - \omega A)\mathbf{x}^* + \omega\mathbf{b}$$

Definition 8.4: Richardson Iteration

The Richardson iteration for solving $A\mathbf{x} = \mathbf{b}$ is:

$$\mathbf{x}^{(k+1)} = (I - \omega A)\mathbf{x}^{(k)} + \omega\mathbf{b}$$

where $\omega > 0$ is the *relaxation parameter*. In terms of matrix splittings:

$$\begin{aligned} P &= \frac{1}{\omega} I \\ N &= P - A = \frac{1}{\omega} I - A \\ B &= I - \omega A \\ \mathbf{f} &= \omega\mathbf{b} \end{aligned}$$

Convergence Analysis

For Richardson iteration, convergence depends on $\rho(I - \omega A) < 1$.

Theorem 8.5: Convergence of Richardson Iteration

Let A be symmetric positive definite with eigenvalues $0 < \lambda_{\min} \le \cdots \le \lambda_{\max}$. The Richardson iteration converges if and only if

$$0 < \omega < \frac{2}{\lambda_{\max}}$$

The optimal parameter is $\omega_{\text{opt}} = \frac{2}{\lambda_{\min}+\lambda_{\max}}$, giving convergence rate

$$\rho(I - \omega_{\text{opt}}A) = \frac{\kappa(A) - 1}{\kappa(A) + 1}$$

where $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ is the condition number.

Proof. Since A is symmetric positive definite, the iteration matrix $B = I - \omega A$ has eigenvalues $\mu_i = 1 - \omega\lambda_i$ where λ_i are the eigenvalues of A. For convergence, we need $|\mu_i| < 1$ for all i.
This requires $|1 - \omega\lambda_i| < 1$, or equivalently $-1 < 1 - \omega\lambda_i < 1$. Since $\lambda_i > 0$, the right inequality gives $\omega < 2/\lambda_i$. The most restrictive constraint comes from the largest eigenvalue: $\omega < 2/\lambda_{\max}$.
To minimize $\rho(B) = \max_i |1 - \omega\lambda_i|$, we balance the extreme eigenvalues by choosing ω such that $|1 - \omega\lambda_{\min}| = |1 - \omega\lambda_{\max}|$. This gives $\omega_{\text{opt}} = 2/(\lambda_{\min} + \lambda_{\max})$, yielding the stated convergence rate. □

When is Richardson Efficient?

Richardson iteration is most effective when:

- Computing $A\mathbf{x}$ is very cheap (sparse matrices, structured matrices)
- A is well-conditioned ($\kappa(A)$ is small)
- As a smoother in multigrid methods
- As a building block for more sophisticated methods

Example 28. Richardson Iteration Example

Consider $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

The exact solution is $\mathbf{x}^* = A^{-1}\mathbf{b} = \frac{1}{3}\begin{bmatrix} 1 \\ 1 \end{bmatrix}$.

The eigenvalues of A are $\lambda_1 = 1, \lambda_2 = 3$, so the optimal parameter is $\omega_{\text{opt}} = \frac{2}{1+3} = \frac{1}{2}$.

Starting with $\mathbf{x}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$:

For $\omega = 1/2$ (optimal):

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - \frac{1}{2}(A\mathbf{x}^{(0)} - \mathbf{b}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\mathbf{x}^{(2)} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \frac{1}{2}\left(\begin{bmatrix} 1.5 \\ 1.5 \end{bmatrix} - \begin{bmatrix} 1 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix}$$

For $\omega = 1$ (suboptimal):

$$\mathbf{x}^{(1)} = \begin{bmatrix}0\\0\end{bmatrix} + \begin{bmatrix}1\\1\end{bmatrix} = \begin{bmatrix}1\\1\end{bmatrix}$$
$$\mathbf{x}^{(2)} = \begin{bmatrix}1\\1\end{bmatrix} - \left(\begin{bmatrix}3\\3\end{bmatrix} - \begin{bmatrix}1\\1\end{bmatrix}\right) = \begin{bmatrix}-1\\-1\end{bmatrix}$$

The optimal choice converges smoothly, while $\omega = 1$ oscillates.

8.5.2 General Convergence Results for Matrix Splittings

Theorem 8.6: Convergence for Matrix Splittings

Let $A = P - N$ with A and P positive definite. Then:

1. If $2P - A$ is positive definite, the iteration $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + P^{-1}\mathbf{r}^{(k)}$ converges.
2. If A is symmetric positive definite and $P + P^T - A$ is positive definite, the iteration converges.

This provides general conditions for convergence that apply to Richardson iteration (where $P = \frac{1}{\omega}I$) and the classical methods we examine next.

8.5.3 Jacobi Method

The Jacobi method represents one of the most intuitive approaches to solving linear systems iteratively. It updates all components of $\mathbf{x}$ simultaneously, using only values from the previous iteration.

Definition 8.7: Jacobi Method

The Jacobi method uses the matrix splitting $P = D$ and $N = L + U$, giving the iteration:

$$\mathbf{x}^{(k+1)} = B_J\mathbf{x}^{(k)} + \mathbf{f}$$
$$\text{where} \quad B_J = -D^{-1}(L + U) = I - D^{-1}A$$
$$\mathbf{f} = D^{-1}\mathbf{b}$$

Corollary 1: (Jacobi Component-wise Formula)**.** The Jacobi method can be expressed component-wise as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \dots, n \tag{8.3}$$

Each component $x_i^{(k+1)}$ depends only on the *previous* iteration values $x_j^{(k)}$.

Interpretation The i-th equation of $A\mathbf{x} = \mathbf{b}$ can be written as:

$$a_{ii}x_i + \sum_{j\neq i} a_{ij}x_j = b_i$$

Solving for x_i gives (8.3). The Jacobi method simply uses the most recent available approximations.

Properties of the Jacobi Method

Advantages:

- Simple to implement and understand
- Naturally parallelizable (each component can be computed independently)
- Memory efficient (only need to store two vectors)

Disadvantages:

- May converge slowly compared to other methods
- Does not immediately use updated information

Theorem 8.8: Convergence of the Jacobi Method

The Jacobi method converges if any of the following conditions hold:

1. A is strictly diagonally dominant by rows
2. A is symmetric positive definite and $2D - A$ is positive definite
3. $\rho(D^{-1}(L+U)) < 1$

8.5.4 Gauss-Seidel Method

The Gauss-Seidel method improves upon Jacobi by immediately using updated values as they become available.

Definition 8.9: Gauss-Seidel Method

The Gauss-Seidel method uses the splitting $P = D + L$ and $N = -U$, leading to:

$$\begin{aligned} \mathbf{x}^{(k+1)} &= B_{GS}\mathbf{x}^{(k)} + \mathbf{f} \\ \text{where} \quad B_{GS} &= -(D+L)^{-1}U \\ \mathbf{f} &= (D+L)^{-1}\mathbf{b} \end{aligned}$$

Corollary 2: (Gauss-Seidel Component-wise Formula)**.** The Gauss-Seidel method updates components sequentially:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right), \quad i = 1, \dots, n$$

Note that $x_i^{(k+1)}$ uses already computed values $x_j^{(k+1)}$ for $j < i$.

Key Difference from Jacobi Gauss-Seidel immediately incorporates new information: when computing $x_i^{(k+1)}$, it uses the updated values $x_1^{(k+1)}, \dots, x_{i-1}^{(k+1)}$ that were just computed.

Theorem 8.10: Convergence Properties of Gauss-Seidel

1. If A is strictly diagonally dominant, Gauss-Seidel converges.
2. If A is symmetric positive definite, Gauss-Seidel converges and satisfies:

$$\|\mathbf{e}^{(k+1)}\|_A \le \|\mathbf{e}^{(k)}\|_A$$

(monotonic convergence in the A-norm).

Example 29. Comparing Jacobi and Gauss-Seidel

Consider the 3 × 3 system:

$$\begin{bmatrix} 4 & 1 & 1 \\ 1 & 4 & 1 \\ 1 & 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 6 \\ 6 \end{bmatrix}$$

The exact solution is $\mathbf{x}^* = [1, 1, 1]^T$.

Jacobi iteration:

$$x_1^{(k+1)} = \frac{1}{4}(6 - x_2^{(k)} - x_3^{(k)})$$
$$x_2^{(k+1)} = \frac{1}{4}(6 - x_1^{(k)} - x_3^{(k)})$$
$$x_3^{(k+1)} = \frac{1}{4}(6 - x_1^{(k)} - x_2^{(k)})$$

Gauss-Seidel iteration:

$$x_1^{(k+1)} = \frac{1}{4}(6 - x_2^{(k)} - x_3^{(k)})$$
$$x_2^{(k+1)} = \frac{1}{4}(6 - x_1^{(k+1)} - x_3^{(k)})$$
$$x_3^{(k+1)} = \frac{1}{4}(6 - x_1^{(k+1)} - x_2^{(k+1)})$$

Starting with $\mathbf{x}^{(0)} = [0, 0, 0]^T$:

k	Jacobi	Gauss-Seidel
0	[0.00, 0.00, 0.00]	[0.00, 0.00, 0.00]
1	[1.50, 1.50, 1.50]	[1.50, 1.13, 0.91]
2	[0.75, 0.75, 0.75]	[1.01, 0.99, 1.00]
3	[1.13, 1.13, 1.13]	[1.00, 1.00, 1.00]

Gauss-Seidel converges faster by using updated information immediately.

Convergence Analysis and Comparison

Theorem 8.11: Convergence for Strict Diagonal Dominant Matrices

Let A be a strictly diagonally dominant matrix by rows, i.e.,

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \quad \text{for all } i = 1, \dots, n.$$

Then both the Jacobi and Gauss-Seidel methods converge for any initial guess $\mathbf{x}^{(0)}$.

Proof. For Jacobi, the iteration matrix is $B_J = -D^{-1}(L + U)$. The infinity norm is

$$\|B_J\|_\infty = \max_i \sum_{j \neq i} \left| \frac{a_{ij}}{a_{ii}} \right| < 1$$

by the strict diagonal dominance assumption. Thus, $\rho(B_J) \le \|B_J\|_\infty < 1$, so Jacobi converges. For Gauss-Seidel, a similar but more involved analysis shows convergence. □

Theorem 8.12: Monotonic Convergence of Gauss-Seidel

If A is symmetric positive definite, the Gauss-Seidel method is monotonically convergent with respect to the $\|\cdot\|_A$ norm:

$$\|\mathbf{e}^{(k+1)}\|_A \leq \|\mathbf{e}^{(k)}\|_A, \quad \text{for } k \geq 0.$$

This ensures that the error decreases at each iteration, guaranteeing convergence.

Proof. For symmetric positive definite A, the Gauss-Seidel error satisfies $\mathbf{e}^{(k+1)} = -(D+L)^{-1}U\mathbf{e}^{(k)}$ where $U = L^T$. Since A is positive definite, the iteration matrix $B_{GS} = -(D+L)^{-1}L^T$ has the property that $\|\mathbf{e}^{(k+1)}\|_A^2 < \|\mathbf{e}^{(k)}\|_A^2$ unless $\mathbf{e}^{(k)} = \mathbf{0}$. This follows from the fact that $(D+L)^{-T}A(D+L)^{-1}$ is positive definite but not equal to the identity, ensuring strict contraction in the A-norm. □

8.5.5 Jacobi Over-Relaxation (JOR)

The Jacobi Over-Relaxation (JOR) method generalizes the Jacobi method by introducing a relaxation parameter ω to potentially accelerate convergence.

Definition 8.13: Jacobi Over-Relaxation (JOR) Method

For the system $A\mathbf{x} = \mathbf{b}$, the JOR iteration is:

$$\mathbf{x}^{(k+1)} = B_{JOR}\mathbf{x}^{(k)} + \mathbf{f}$$

where

$$\begin{aligned} B_{JOR} &= (1-\omega)I - \omega D^{-1}(L+U) \\ \mathbf{f} &= \omega D^{-1}\mathbf{b} \end{aligned}$$

Componentwise, for $i = 1, \dots, n$:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}x_j^{(k)}\right)$$

Interpretation of the Relaxation Parameter

- For $\omega = 1$: JOR reduces to the standard Jacobi method
- For $0 < \omega < 1$: Under-relaxation (damping)
- For $\omega > 1$: Over-relaxation (acceleration)

Theorem 8.14: Convergence of the JOR Method

Let A be a matrix with nonzero diagonal entries. The JOR method converges for any initial guess $\mathbf{x}^{(0)}$ if either:

1. A is strictly diagonally dominant by rows and $0 < \omega < 2$, or
2. A is symmetric positive definite and $0 < \omega < \frac{2}{\rho(D^{-1}A)}$

where $\rho(D^{-1}A)$ is the spectral radius of $D^{-1}A$.

Proof. The eigenvalues of the JOR iteration matrix are related to those of the Jacobi matrix by $\mu_k = 1 - \omega + \omega\lambda_k$, where λ_k are the eigenvalues of $-D^{-1}(L+U)$. For A strictly diagonally dominant, $|\lambda_k| < 1$, so for $0 < \omega < 2$, we have $|\mu_k| < 1$ and thus $\rho(B_{JOR}) < 1$. For symmetric positive definite A, convergence holds for $0 < \omega < 2/\rho(D^{-1}A)$ by spectral analysis. □

8.5.6 Successive Over-Relaxation (SOR)

The Successive Over-Relaxation (SOR) method generalizes Gauss-Seidel by introducing a relaxation parameter ω.

Definition 8.15: Successive Over-Relaxation (SOR) Method

The Successive Over-Relaxation (SOR) method is a stationary iterative scheme for solving $A\mathbf{x} = \mathbf{b}$, defined by the iteration:

$$\mathbf{x}^{(k+1)} = B_{SOR}\mathbf{x}^{(k)} + \mathbf{f}, \qquad B_{SOR} = (D + \omega L)^{-1}\left[(1-\omega)D - \omega U\right], \qquad \mathbf{f} = (D+\omega L)^{-1}\omega\mathbf{b}.$$

Componentwise, for $i = 1, \dots, n$:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \frac{\omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right).$$

For $\omega = 1$, SOR reduces to Gauss-Seidel. For $0 < \omega < 1$, the method is under-relaxed; for $\omega > 1$, it is over-relaxed. SOR can converge much faster than Gauss-Seidel for an optimal choice of ω (typically $1 < \omega < 2$ for many problems).

Theorem 8.16: Convergence of the SOR Method

The Successive Over-Relaxation (SOR) method converges for any initial guess $\mathbf{x}^{(0)}$ if the matrix A is either strictly diagonally dominant by rows, or symmetric positive definite, and the relaxation parameter satisfies $0 < \omega < 2$. Moreover, for any $\omega \in \mathbb{R}$, the spectral radius of the SOR iteration matrix satisfies $\rho(B_{SOR}) \geq |\omega - 1|$. Therefore, the SOR method fails to converge if $\omega \leq 0$ or $\omega \geq 2$.

Proof. For any $\omega \in \mathbb{R}$, it can be shown that all eigenvalues λ_i of B_{SOR} satisfy $|\lambda_i| \geq |\omega - 1|$, so $\rho(B_{SOR}) \geq |\omega - 1|$. Thus, if $\omega \leq 0$ or $\omega \geq 2$, $\rho(B_{SOR}) \geq 1$ and the method does not converge.
If A is strictly diagonally dominant by rows or symmetric positive definite, and $0 < \omega < 2$, then $\rho(B_{SOR}) < 1$ and the method converges. See standard texts for details and further discussion. □

Optimal Relaxation Parameter

For many problems, particularly those arising from discretizations of partial differential equations, there exists an optimal value of ω that minimizes the spectral radius.

Theorem 8.17: Optimal SOR Parameter for Consistently Ordered Matrices

Let A be a symmetric positive definite matrix that is consistently ordered, and let ρ_J be the spectral radius of the Jacobi iteration matrix. Then the optimal relaxation parameter for SOR is:

$$\omega_{\text{opt}} = \frac{2}{1 + \sqrt{1 - \rho_J^2}}$$

and the corresponding convergence rate is:

$$\rho(B_{SOR}(\omega_{\text{opt}})) = \omega_{\text{opt}} - 1 = \frac{1 - \sqrt{1 - \rho_J^2}}{1 + \sqrt{1 - \rho_J^2}}$$

Example 30. SOR vs Gauss-Seidel Performance

Consider a tridiagonal system arising from finite difference discretization:

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

For this matrix, $\rho_J \approx 0.618$, giving $\omega_{\text{opt}} \approx 1.27$.

Convergence comparison:

Method	Convergence Rate	Iterations to Reduce Error by 10^{-6}
Gauss-Seidel ($\omega = 1$)	0.618	≈ 29
SOR ($\omega = 1.27$)	0.270	≈ 11

SOR with optimal ω converges nearly 3 times faster than Gauss-Seidel.

8.6 Practical Considerations

8.6.1 Implementation Details

Storage Requirements

- **Jacobi**: Requires storage for two vectors ($\mathbf{x}^{(k)}$ and $\mathbf{x}^{(k+1)}$)
- **Gauss-Seidel/SOR**: Can update in-place, requiring only one vector
- **Matrix storage**: Only need to store A (can exploit sparsity)

Stopping Criteria

In practice, we stop the iteration when a convergence criterion is satisfied:

Algorithm 7 General Iterative Method with Stopping Criteria

```
Choose initial guess $\mathbf{x}^{(0)}$
Set tolerance $\varepsilon > 0$ and maximum iterations $k_{\max}$
for $k = 0, 1, 2, \dots, k_{\max}$ do
    Compute $\mathbf{x}^{(k+1)}$ using chosen method
    Compute residual $\mathbf{r}^{(k+1)} = \mathbf{b} - A\mathbf{x}^{(k+1)}$
    if $\|\mathbf{r}^{(k+1)}\| < \varepsilon$ or $\frac{\|\mathbf{r}^{(k+1)}\|}{\|\mathbf{r}^{(0)}\|} < \varepsilon$ then
        return $\mathbf{x}^{(k+1)}$ (converged)
return failure (did not converge)
```

8.6.2 Convergence Rate Analysis

The *convergence rate* is typically measured by the spectral radius $\rho(B)$ of the iteration matrix. The error satisfies:

$$\|\mathbf{e}^{(k)}\| \leq C\rho(B)^k \|\mathbf{e}^{(0)}\|$$

for some constant C.

Definition 8.18: Asymptotic Convergence Rate

The asymptotic convergence rate of an iterative method is:

$$R = -\log(\rho(B))$$

A larger R indicates faster convergence.

8.6.3 Method Comparison Summary

Method	Matrix Splitting	Parallelizable	Memory	Typical Performance
Richardson	$P = \frac{1}{\omega} I$	Yes	Low	Slow, simple
Jacobi	$P = D$	Yes	Medium	Moderate
Gauss-Seidel	$P = D + L$	No	Low	Good
JOR	$P = \frac{1}{\omega} D$	Yes	Medium	Variable
SOR	$P = \frac{1}{\omega}(D + \omega L)$	No	Low	Often best

8.6.4 When Each Method is Preferred

Richardson Iteration

- Matrix-vector multiplication is very cheap
- As a smoother in multigrid methods
- Preconditioning with simple preconditioners

Jacobi Method

- Parallel computing environments
- Matrices with strong diagonal dominance
- When memory bandwidth is limited

Gauss-Seidel Method

- Sequential computing environments
- When fast convergence is more important than parallelizability
- Symmetric positive definite systems

SOR Method

- Optimal ω can be estimated or computed
- Consistently ordered matrices (e.g., from PDE discretizations)
- When maximum convergence speed is essential

8.7 Advanced Topics

8.7.1 Block Methods

For systems with natural block structure, we can generalize the classical methods to operate on blocks rather than individual components.

Definition 8.19: Block Jacobi Method

Partition A and $\mathbf{x}$ into blocks:

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m1} & A_{m2} & \cdots & A_{mm} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_m \end{bmatrix}$$

The block Jacobi iteration is:

$$A_{ii}\mathbf{x}_i^{(k+1)} = \mathbf{b}_i - \sum_{j\neq i} A_{ij}\mathbf{x}_j^{(k)}, \quad i = 1, \ldots, m$$

Block methods can offer better convergence properties and are natural for parallel implementation.

8.7.2 Preconditioning

The convergence of iterative methods can be dramatically improved by *preconditioning*. Instead of solving $A\mathbf{x} = \mathbf{b}$, we solve the equivalent system:

$$M^{-1}A\mathbf{x} = M^{-1}\mathbf{b}$$

where $M \approx A$ is easy to invert.

> **Example 31. Diagonal Preconditioning**
>
> Choose $M = D$ (the diagonal of A). This is equivalent to scaling each equation by $1/a_{ii}$, which can significantly improve convergence for poorly scaled systems.

8.8 Summary

This chapter introduced the fundamental stationary iterative methods for solving linear systems:

- **Theoretical foundation**: Matrix splittings and convergence criteria based on spectral radius
- **Classical methods**: Richardson, Jacobi, Gauss-Seidel, and their relaxed variants (JOR, SOR)
- **Convergence analysis**: Sufficient conditions for convergence and convergence rates
- **Practical considerations**: Implementation details, stopping criteria, and method selection

The key insight is that convergence depends on the spectral radius $\rho(B) < 1$ of the iteration matrix. While these stationary methods are foundational, modern practice often employs more sophisticated Krylov subspace methods (conjugate gradients, GMRES) that achieve faster convergence for many problems.

However, stationary methods remain important as:

- Building blocks for more advanced methods
- Smoothers in multigrid algorithms
- Preconditioners for Krylov methods
- Simple, robust solvers for well-conditioned problems

Part IV

Non-linear Systems & Optimization

Introduction

Non-linear systems arise frequently in scientific computing, engineering, and applied mathematics. Unlike linear systems, nonlinear systems can exhibit complex behavior such as **multiple solutions**, **bifurcations**, and **sensitivity to initial conditions**. Solving nonlinear equations is therefore a central challenge in numerical mathematics.

Optimization is closely related to the study of nonlinear systems. Many optimization problems, especially those involving nonlinear objective functions or constraints, require solving nonlinear equations as part of their solution process. For example, finding the *stationary points* of a function—where its gradient vanishes—leads directly to solving a system of nonlinear equations. Conversely, techniques developed for optimization, such as *Newton's method* and its variants, are often employed to solve general nonlinear systems.

This part introduces **fundamental concepts** and **numerical methods** for analyzing and solving nonlinear systems and optimization problems, highlighting their deep interconnections and practical applications.

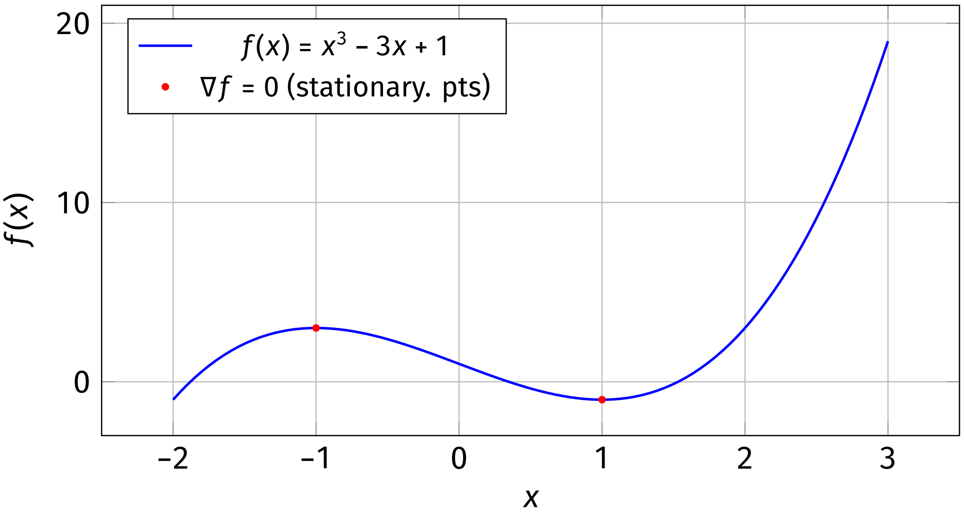


Figure 8.1: A nonlinear function with multiple stationary points illustrates the complexity of nonlinear systems and optimization.

Chapter 9

Linear Optimization

Examples

Exam 2022

Problem 2 Let $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ be given. We want to solve the optimization problem

$$\operatorname*{arg\,min}_{\mathbf{x}\in\mathbb{R}^n} \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|_2^2.$$

```python
import numpy as np
def sol(A, b, x, tau=1, max_iter=200):
    (n, _) = A.shape
    for _ in range(max_iter):
        r = b - (A @ x)
        for j in range(n):
            x[j] = x[j] + tau * r[j]
    return xs
```

(a)

$$\begin{aligned}
F(\mathbf{x}) &= \frac{1}{2}\|A\mathbf{x}-\mathbf{b}\|^2\\
&= \frac{1}{2}(A\mathbf{x}-\mathbf{b})^\top(A\mathbf{x}-\mathbf{b})\\
&= \frac{1}{2}\left(\mathbf{x}^\top A^\top A\mathbf{x} - \mathbf{x}^\top A^\top\mathbf{b} - \mathbf{b}^\top A\mathbf{x} - \mathbf{b}^\top\mathbf{b}\right)\\
&= \frac{1}{2}\left(\mathbf{x}^\top A^\top A\mathbf{x} - 2\mathbf{b}^\top A\mathbf{x} - \mathbf{b}^\top\mathbf{b}\right)
\end{aligned}$$

- Gradient:

$$\begin{aligned}
\nabla F(\mathbf{x}) &= \frac{1}{2}2A^\top A\mathbf{x} - A^\top\mathbf{b}\\
&= A^\top A\mathbf{x} - A^\top\mathbf{b}
\end{aligned}$$

- FONC:

$$\begin{aligned}\nabla F(\mathbf{x}) &= \mathbf{0}\\ A^\top A\mathbf{x} - A^\top \mathbf{b} &= \mathbf{0}\\ A^\top A\mathbf{x} &= A^\top \mathbf{b}\\ \mathbf{x} &= \left(A^\top A\right)^{-1} A^\top \mathbf{b}\end{aligned}$$

(b) The python code shows the Richardson iteration method for solving the linear system $A\mathbf{x} = \mathbf{b}$ iteratively, where x is updated by adding a scaled residual $r = b - Ax$ at each step. It does not converge towards the right solution, since `tau = 1` is not small enough to ensure convergence. To reduce the code to a single line, we can use the following:

```
def sol(A, b, x, tau=1, max_iter=200):
    for _ in range(max_iter):
        x += tau * (b - A @ x)
    return x
```

(c) For this task let $A \in \mathbb{R}^{n \times n}$ be of full rank. Compare the method of using a gradient descent on the problem in a) to the method in b). How are the two methods related?
Both methods aim to solve a linear system $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$. If we apply gradient descent to the problem in (a) we get the iteration

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \tau \nabla F(\mathbf{x}^{(k)}) = \mathbf{x}^{(k)} - \tau(A^\top A\mathbf{x}^{(k)} - A^\top \mathbf{b}).$$

If we set $\tilde{A} = A^\top A$ and $\tilde{\mathbf{b}} = A^\top \mathbf{b}$, we see that the gradient descent method is equivalent to the Richardson iteration method for solving the linear system $\tilde{A}\mathbf{x} = \tilde{\mathbf{b}}$. Thus, both methods are closely related and can be used to solve the same problem, albeit through different iterative approaches.

Problem 3. (Cubic Splines, 20%) We consider the spline space S_3 with knot vector $[-3, -2, 0, 1, 4]$. Which of the following functions is in the space S_3? State a reason for each.

(a) $f_1(x) = \pi x^2$
This is a polynomial of degree 2, which is less than or equal to 3. However, a cubic spline in S_3 must be piecewise cubic with possible breaks at the knots. Since $f_1(x)$ is a global polynomial and does not change form at the knots, it is in S_3.

(b) $f_2(x) = |x + 2|^3 + |x - 1|^4$
The term $|x - 1|^4$ is a quartic function, which is not allowed in S_3 (only degree 3 or less). Therefore, $f_2(x) \notin S_3$.

(c) $f_3(x) = \frac{1}{3}\max\{x + 2, 0\}^3 + \max\{x - 1, 0\}^3$ can be rewritten as:

$$f_3(x) = \begin{cases} 0 & \text{for } x < -2, \\ \frac{1}{3}(x + 2)^3 & \text{for } -2 \le x < 1, \\ (x - 1)^3 & \text{for } x \ge 1. \end{cases}$$

We check the continuity and differentiability at the knots:

- At $x = -2$:

$$f_3(-2) = 0, \quad f_3'(-2) = 0.$$

- At $x = 1$:

$$f_3(1) = 0, \quad f_3'(1) = 0.$$

Since $f_3 \in C^2([-3, 4])$ and is piecewise cubic on the intervals defined by the knots, it is in S_3.
Both terms are shifted cubic functions (truncated power functions), which are standard basis functions for cubic splines with knots at -2 and 1. Thus, $f_3(x) \in S_3$.

(d) For some $a, b \in \mathbb{R}$, consider

$$f_{a,b}(x) = \begin{cases} x^3 & \text{for } x \in [-3, 1), \\ -x^3 + ax^2 + bx + 2 & \text{for } x \in [1, 4]. \end{cases}$$

To be a spline in S_3, the function must be C^2 at $x = 1$ (continuous up to second derivative). We check the conditions at $x = 1$:

- Continuity:

$$1^3 = -1^3 + a(1)^2 + b(1) + 2 \implies 1 = -1 + a + b + 2 \implies a + b = 0.$$

- First derivative continuity:

$$3(1)^2 = -3(1)^2 + 2a(1) + b \implies 3 = -3 + 2a + b \implies 2a + b = 6.$$

- Second derivative continuity:

$$6 = -6 + 2a \implies 2a = 12 \implies a = 6.$$

Substituting $a = 6$ into $a + b = 0$ gives $b = -6$. Thus, $f_{6,-6}(x) \in S_3$.

(e) The dimension of the cubic spline space S_3 with the given knot vector $[-3, -2, 0, 1, 4]$ is $n + k$, where n is the number of intervals (number of knots minus 1) and k is the degree of the spline (here, 3). There are 5 knots, so 4 intervals. Thus,

$$\begin{aligned} \dim(S_k) &= \text{number of intervals} + \text{degree} \\ &= (\text{size}(\Delta) - 1) + \deg s_i(x) \\ &= (n - 1) + k \\ &= 4 + 3 = 7. \end{aligned}$$

Therefore, the space S_3 has dimension 7.

Problem 4. (Numerical Integration, 20%) We want to compute the integral $I(f) = \int_1^2 \frac{2}{x^2}\,dx$ numerically.

(a) **Compute the two quadratures $Q_1(f)$ and $Q_2(f)$:** The quadrature $Q_n(f)$ approximates the integral $I(f)$ using n subintervals. Here, we will use the trapezoidal rule and the composite trapezoidal rule.

$$Q_n(f) = h\left[\frac{1}{2}f(a) + \sum_{i=1}^{n-1} f(a + ih) + \frac{1}{2}f(b)\right], \quad h = \frac{b - a}{n}$$

- **Trapezoidal rule (Q_1):**

$$\begin{aligned} Q_1(f) &= \frac{b - a}{2}\,[f(a) + f(b)] \\ &= \frac{2 - 1}{2}\,[f(1) + f(2)] \\ &= \frac{1}{2}\left(2 + \frac{2}{4}\right) \\ &= \frac{1}{2}\,(2 + 0.5) \\ &= 1.25 \end{aligned}$$

- **Composite trapezoidal rule with $n = 2$ (Q_2):**
The interval $[1, 2]$ is split into two subintervals: $h = \frac{2-1}{2} = 0.5$. The nodes are $x_0 = 1$, $x_1 = 1.5$, $x_2 = 2$.

$$\begin{aligned}
Q_2(f) &= \frac{h}{2}\left[f(x_0) + 2f(x_1) + f(x_2)\right] \\
&= \frac{0.5}{2}\left[2 + 2 \cdot \frac{2}{(1.5)^2} + 0.5\right] \\
&= 0.25\left[2 + 2 \cdot \frac{2}{2.25} + 0.5\right] \\
&= 0.25\left[2 + 2 \cdot 0.8889 + 0.5\right] \\
&\approx 0.25\left[2 + 1.7778 + 0.5\right] \\
&= 0.25 \times 4.2778 \\
&\approx 1.069
\end{aligned}$$

(b) **Best upper bound for the error $|I(f) - Q_2(f)|$:** Given that:

$$f(x) = \frac{2}{x^2}, \quad f''(x) = \frac{12}{x^4}, \quad \max_{x\in[1,2]} |f''(x)| = 12, \quad h = 0.5$$

The error bound for the composite trapezoidal rule is:

$$\begin{aligned}
|I(f) - Q_2(f)| &\le \frac{(b-a)h^2}{12} \max_{x\in[a,b]} |f''(x)| \\
&\le \frac{(b-a)h^2}{12} \max_{x\in[a,b]} |f''(x)| \\
&\le \frac{1 \cdot (0.5)^2}{12} \cdot 12 = 0.25
\end{aligned}$$

(c) **Combine Q_1 and Q_2 using extrapolation to derive a more accurate quadrature Q_3:** Using the Romberg method,

$$A_{m,n} = \frac{4^m A_{m-1,n} - A_{m-1,2n}}{4^m - 1}$$

we can combine $A_{00} = Q_1$ and $A_{10} = Q_2$ to get a more accurate estimate $A_{11} = Q_3$:

$$\begin{aligned}
A_{11} = Q_3(f) &= \frac{4^1 Q_2(f) - Q_1(f)}{4^1 - 1} \\
&= \frac{4 \cdot \frac{77}{72} - \frac{5}{4}}{3} \\
&= \frac{109}{108} \approx 1.00926
\end{aligned}$$

(d) **Which quadrature $\tilde{Q}$ with minimal additional point evaluations do you have to compute to obtain the next step in the Romberg scheme? Which is the highest term in the Romberg scheme you can compute then?**
To continue the Romberg scheme, we need to compute $Q_4(f)$ using the next level of extrapolation. This requires evaluating the function at one additional point, which is $x = 1.25$. The highest term we can compute then is A_{20}, which corresponds to $Q_4(f)$.

$$\begin{aligned}
A_{20} &= \frac{4^2 Q_4(f) - Q_2(f)}{4^2 - 1} \\
&= \frac{16Q_4(f) - Q_2(f)}{15} \\
&= \frac{16Q_4(f) - \frac{77}{72}}{15}
\end{aligned}$$

(e) **Does a quadrature formula** $Q(f) = \alpha f(x_0) + f(x_1)$ **exist with degree of exactness 2?**
Answer: No, such a formula cannot have degree of exactness 2. With only two points and one free weight, you can match degree 1 (linear functions), but not degree 2 (quadratic functions). Simpson's rule, which is exact for quadratics, requires three points.

Assume $Q(f)$ is exact for $f(x) = 1, x, x^2$ on $[a, b]$:

$$\int_a^b 1\,dx = 2\alpha \implies \alpha = \frac{b-a}{2}$$

$$\int_a^b x\,dx = \alpha(x_0 + x_1) \implies x_0 + x_1 = a + b$$

$$\int_a^b x^2\,dx = \alpha(x_0^2 + x_1^2)$$

But $x_0^2 + x_1^2 = (a+b)^2 - 2x_0x_1$, so

$$x_0x_1 = \frac{1}{2}\left[(a+b)^2 - \frac{2}{b-a}\frac{b^3 - a^3}{3}\right]$$

For general a, b, this does not always yield valid nodes in $[a, b]$. In fact, two-point formulas cannot integrate all quadratics exactly unless using special nodes (Gaussian quadrature), not just endpoints or equally weighted nodes.

Conclusion: No such formula exists for all quadratics; three points are needed (e.g., Simpson's rule).

Why three points? A quadratic $ax^2 + bx + c$ has three coefficients, so three conditions (nodes) are needed to determine it uniquely and integrate exactly.

Part V

Numerical Integration

Chapter 10

Introduction to Numerical Integration

10.1 Overview

Numerical integration seeks to approximate definite integrals when an exact analytical solution is unavailable or impractical. This is achieved by replacing the continuous integral with a weighted sum over discrete points:

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} w_i f(x_i)$$

where x_i are nodes and w_i are weights determined by the chosen quadrature method (from Latin *quadratura*, ”making square”). Common methods include the **midpoint rule**, **trapezoidal rule**, and **Simpson’s rule**, each characterized by specific error terms and convergence rates. For example, the error in the trapezoidal rule is $\mathcal{O}((b-a)^3 f''(\xi)/n^2)$, while Simpson’s rule has error $\mathcal{O}((b-a)^5 f^{(4)}(\xi)/n^4)$ for some $\xi \in [a,b]$.

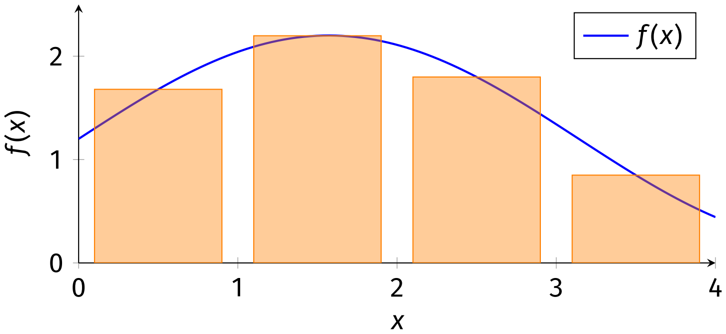


Figure 10.1: Illustration of estimating the area under the curve $f(x)$ using the midpoint rule. The orange rectangles represent the approximation. This figure shows how the area under the curve $f(x)$ can be estimated by summing the areas of the orange rectangles.

In this part, we will systematically develop the theory and practical aspects of numerical integration, with emphasis on:

- **Integration Methods:** Detailed introduction to the midpoint, trapezoidal, and Simpson’s rules for approximating definite integrals, i.e., $I \approx \int_a^b f(x)\,dx$. We will discuss the construction, application, and comparative advantages of each method.

- **Error Analysis:** Examination of error bounds and convergence rates, typically expressed as $\mathcal{O}(h^p)$ for step size h and method order p. We will also address stability criteria for quadrature rules, such as the requirement $\sum_i |w_i| < C$ for some constant C, and clarify the meaning of convergence ($I_h \to I$ as $h \to 0$).

10.2 Fundamental Principles

We consider numerical methods to compute:

$$I(f) = \int_a^b f(x)\,dx$$

Any explicit formula to approximate $I(f)$ is called a **quadrature formula** or **numerical integration formula**.

10.2.1 General Principle

The general approach consists of:

1. Construct an explicitly integrable function f_n, $n \geq 0$, to approximate f on $[a, b]$ (e.g., by interpolation)
2. Define $I_n(f) = I(f_n)$

For $f \in C([a, b])$, the quadrature error $E_n(f) = I(f) - I_n(f)$ satisfies:

$$|E_n(f)| = \left| \int_a^b [f(x) - f_n(x)]\,dx \right| \leq \int_a^b |f(x) - f_n(x)|\,dx \leq \max_{x\in[a,b]} |f(x) - f_n(x)| \cdot (b - a)$$

10.3 Interpolatory Quadrature Formulas

10.3.1 Basic Construction

Take $f_n = \Pi_n f$, i.e., the interpolation polynomial to f with respect to some nodes $\{x_0, \dots, x_n\}$, since this is easily integrable. Then:

$$f_n = \sum_{i=0}^{n} f(x_i)\ell_i(x), \quad \ell_i(x) = \prod_{\substack{k=0 \\ k\neq i}}^{n} \frac{x - x_k}{x_i - x_k} \in \mathbb{P}_n$$

and we get:

$$I_n(f) = I(f_n) = \int_a^b f_n(x)\,dx = \sum_{i=0}^{n} \left(\int_a^b \ell_i(x)\,dx \right) f(x_i)$$

The coefficients $\alpha_i = \int_a^b \ell_i(x)\,dx$ are independent of f. They just depend on the nodes $\{x_0, \dots, x_n\}$!

This is an instance of a **linear formula**:

$$I_n(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$$

10.3.2 Extended Linear Formulas and Exactness

We can extend a linear formula to include derivatives $f^{(q)}(x_i)$, $q = 0, \dots, Q$:

$$I_n(f) = \sum_{i=0}^{n} \sum_{q=0}^{Q} \alpha_{iq} f^{(q)}(x_i)$$

for example using Hermite interpolation ($Q = 1$), where $\alpha_{iq} \in \mathbb{R}$.

Definition 10.1: Degree of Exactness

The **degree of exactness** of a quadrature formula is the maximum integer r such that:

$$I_n(f) = I(f) \text{ for all } f \in \mathbb{P}_r$$

10.4 Classical Quadrature Rules

The midpoint, trapezoidal, and Simpson rules are fundamental examples. We assume these are known, including analysis of their errors:

- **Midpoint rule**: $\int_a^b f(x)\,dx \approx (b-a) f\left(\frac{a+b}{2}\right)$
- **Trapezoidal rule**: $\int_a^b f(x)\,dx \approx \frac{b-a}{2}[f(a) + f(b)]$
- **Simpson rule**: $\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$

Please review these fundamental rules and refresh your memory.

10.5 Newton-Cotes Formulas

Newton-Cotes formulas are based on interpolation with the special case of equispaced points in $[a, b]$:

$$x_k = x_0 + kh, \quad k = 0, \dots, n$$

These include the midpoint, trapezoidal, and Simpson rules.

A formula is called:

- **closed** if $x_0 = a$ and $x_n = b$, and we obtain $h = \frac{b-a}{n}$, $n \geq 1$
- **open** if $x_0 = a + h$ and $x_n = b - h$, and we have $h = \frac{b-a}{n+2}$, $n \geq 0$

Then:

- Midpoint rule is the open formula with $n = 0$
- Trapezoidal and Simpson are the closed formulas for $n = 1, 2$, respectively

10.5.1 Newton-Cotes Properties

The weights α_i in the linear formula $I_n(f) = \sum_{i=0}^{n} \alpha_i f(x_i)$ with equispaced x_i depend only on n and h.

We obtain with $\phi_i(t) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{t-k}{i-k}$:

- **closed**: $I_n(f) = h \sum_{i=0}^{n} w_i f(x_i)$, $w_i = \int_0^n \phi_i(t)\,dt$

- **open**: $I_n(f) = h\sum_{i=0}^{n} w_i f(x_i)$, $w_i = \int_{-1}^{n+1} \phi_i(t)\,dt$

Key properties:

- h only appears as a factor in front
- n determines the number of coefficients/summands and appears once in every weight w_i
- ⇒ precompute and store

10.5.2 Precomputed Newton-Cotes Weights

- In the closed formulas: by symmetry of ϕ_i and ϕ_{n-i} ⇒ $w_i = w_{n-i}$ for $i = 0, \ldots, n-1$ for even n
- For odd cases, similarly: $w_i = w_{n-i}$, $i = 0, \ldots, n$
- We only have to store w_i, $i < \frac{n+1}{2}$
- Higher order open NC formulas: negative weights

Closed Newton-Cotes Weights

n	Formula Name	Weights w_i	Degree of Exactness
1	Trapezoidal	$w_0 = w_1 = \frac{1}{2}$	1
2	Simpson's	$w_0 = w_2 = \frac{1}{6}, w_1 = \frac{4}{6}$	3
3	Simpson's 3/8	$w_0 = w_3 = \frac{1}{8}, w_1 = w_2 = \frac{3}{8}$	3
4	Boole's	$w_0 = w_4 = \frac{7}{90}, w_1 = w_3 = \frac{32}{90}, w_2 = \frac{12}{90}$	5

Open Newton-Cotes Weights

n	Formula Name	Weights w_i	Degree of Exactness
0	Midpoint	$w_0 = 2$	1
1	Two-point	$w_0 = w_1 = \frac{3}{2}$	1
2	Three-point	$w_0 = w_2 = \frac{4}{3}, w_1 = \frac{2}{3}$	3
3	Four-point	$w_0 = w_3 = \frac{5}{4}, w_1 = w_2 = \frac{3}{4}$	3

Note: For $n \geq 8$ in closed formulas and $n \geq 3$ in open formulas, some weights become negative, which can lead to numerical instability.

10.5.3 Error Analysis for Newton-Cotes Formulas

Theorem 10.2: Newton-Cotes Error

Let $f \in C^{(n+2)}([a,b])$ and $n > 1$ be given. The error $E_n(f) = I(f) - I_n(f)$ reads:
For even n:

$$E_n(f) = \frac{M_n}{(n+2)!} h^{n+3} f^{(n+2)}(\xi)$$

For odd n:

$$E_n(f) = \frac{K_n}{(n+1)!} h^{n+2} f^{(n+1)}(\xi)$$

for some $\xi \in (a,b)$, where using $\pi_{n+1}(t) = \prod_{i=0}^{n}(t-i)$ we have:

$$M_n = \begin{cases} \int_0^n t\pi_{n+1}(t)\,dt & \text{for closed} \\ \int_{-1}^{n+1} t\pi_{n+1}(t)\,dt & \text{for open} \end{cases}$$

$$K_n = \begin{cases} \int_0^n \pi_{n+1}(t)\,dt & \text{for closed} \\ \int_{-1}^{n+1} \pi_{n+1}(t)\,dt & \text{for open} \end{cases}$$

10.6 Composite Newton-Cotes Formulas

10.6.1 Motivation

1. Avoid high polynomial degree (i.e., on large intervals $[a,b]$ or ”complicated” f)
2. Known: composite Midpoint | Trapezoidal | Simpson rule

10.6.2 The Idea

Write $[a,b]$ as m subintervals:

$$\int_a^b f(x)\,dx = \sum_{j=0}^{m-1} \int_{T_j} f(x)\,dx$$

with $T_j = [y_j, y_{j+1}]$, $y_j = a + jH$, $j = 0, \dots, m$, i.e., $H = \frac{b-a}{m}$

⇒ Perform Newton-Cotes on each T_j (of length H) with (equispaced) nodes $x_k^{(j)}$ and weights (computed earlier) $\alpha_k^{(j)}$, $0 \le k \le n$, to obtain:

$$I_{n,m}(f) = \sum_{j=0}^{m-1} \sum_{k=0}^{n} \alpha_k^{(j)} f(x_k^{(j)})$$

10.6.3 Composite Newton-Cotes Error

Theorem 10.3: Composite Newton-Cotes Error

Let $f \in C^{(n+2)}(a,b)$, m be given and n be even. Then:

$$E_{n,m}(f) = I(f) - I_{n,m}(f) = \frac{b-a}{(n+2)!} M_n H^{n+2} f^{(n+2)}(\xi)$$

and for odd n:

$$E_{n,m}(f) = I(f) - I_{n,m}(f) = \frac{b-a}{(n+1)!} K_n \frac{\gamma_n H^{n+1}}{\gamma_{n+2}} f^{(n+1)}(\xi)$$

for some $\xi \in (a,b)$, where:

- M_n and K_n are as before
- $\gamma_n = \begin{cases} n+2 & \text{for open} \\ n & \text{for closed} \end{cases}$ formulas

10.7 Richardson Extrapolation for Numerical Integration

10.7.1 Basic Idea

Small step sizes give good results. Extrapolate on h.

Suppose a numerical method depends smoothly on a step size h, e.g., $h = \frac{b-a}{n}$:

$$Q(h) = I_n(f) = I(f) + \alpha_1 h + \alpha_2 h^2 + \alpha_3 h^3 + \dots$$

Idea II: Evaluate Q for different values of h, for example h and δh, $\delta < 1$. Typically: $\delta = \frac{1}{2}$.

$$Q(h) = I(f) + \alpha_1 h + \alpha_2 h^2 + \alpha_3 h^3 + \dots$$
$$Q(\delta h) = I(f) + \alpha_1 \delta h + \alpha_2 \delta^2 h^2 + \alpha_3 \delta^3 h^3 + \dots$$

Goal: Eliminate the first error term $\alpha_1 h$ by computing:

$$Q(\delta h) - \delta Q(h) = (1-\delta)I(f) + 0 + \alpha_2 \delta(\delta - 1)h^2 + (\cdots)h^3 + \dots$$

10.7.2 Richardson Extrapolation Formula

We divide the equation by $1-\delta$ and obtain:

$$\tilde{Q}(h) := \frac{Q(\delta h) - \delta Q(h)}{1-\delta} = I(f) - \alpha_2 \delta h^2 + (\cdots)h^3 + \dots$$
$$= I(f) + \tilde{\alpha}_2 h^2 + \tilde{\alpha}_3 h^3 + \dots, \quad \tilde{\alpha}_k = \frac{\delta^k - \delta}{1-\delta}\alpha_k$$

Observe that:

- We took our "old"/original scheme $Q(h)$ and evaluated it twice (for h and δh)
- We obtain an explicit formula for $\tilde{Q}$
- ⇒ since h is small, the largest error term vanishes ⇒ more accurate!
- We obtain a higher order method (at least accurate up to 2nd order)

10.7.3 Iterative Richardson Extrapolation

But even more: If we have:

- $\tilde{Q}(h)$ (based on $Q(h)$ and $Q(\delta h)$)
- $\tilde{Q}(\delta h)$ (based on $Q(\delta h)$ and $Q(\delta^2 h)$)

⇒ Build $\hat{Q}(h)$ from these two to eliminate the h^2 term!

This iteration process is called **Richardson extrapolation**.

Note that there exist methods with only even powers of h in their expansion, i.e.:

$$Q(h) = I(f) + \alpha_2 h^2 + \alpha_4 h^4 + \alpha_6 h^6 + \ldots$$

Then the scheme is even faster in obtaining accuracy, order would increase by 2 in every step!

10.8 Euler-Maclaurin Summation Formula

Let $f \in C^{2k+2}([a,b])$. We approximate $I(f) = \int_a^b f(x)\,dx$ by a composite trapezoidal rule.

Let T_m be the composite trapezoidal rule with $h = h_m = \frac{b-a}{m}$, i.e.:

$$T_m(f) = \frac{1}{2}h_m f(a) + h_m \sum_{j=1}^{m-1} f(a + jh_m) + \frac{1}{2}h_m f(b)$$

Then one can prove:

$$T_m(f) = I(f) + \sum_{i=1}^{k} \frac{B_{2i}}{(2i)!} h_m^{2i} [f^{(2i-1)}(b) - f^{(2i-1)}(a)] + R_{2k+2}$$

where:

- remainder $R_{2k+2} = \frac{B_{2k+2}}{(2k+2)!}(b-a)f^{(2k+2)}(\eta)$ for some $\eta \in (a,b)$
- Bernoulli numbers B_j are given by the generating (power) series (or Taylor series at $z = 0$):

$$\frac{z}{e^z - 1} = \sum_{k=0}^{\infty} \frac{B_k}{k!} z^k$$

and $B_{2k+1} = 0,\ k > 0$

10.9 Romberg Integration

Named after Werner Romberg (1909-2003), this method employs the extrapolation procedure with $\delta = \frac{1}{2}$.

Here we use $h_m = \frac{b-a}{2^m}$ and we use the trapezoidal rule initially:

$$A_{m,0} = T_{2^m}(f) = \frac{1}{2}h_m f(a) + h_m \sum_{j=1}^{2^m-1} f(a + jh_m) + \frac{1}{2}h_m f(b), \quad m = 0, 1, \ldots$$

Hence we get from Euler-Maclaurin:

$$A_{m,0} = I(f) + \sum_{i=1}^{\infty} \alpha_i h_m^{2i} \quad \text{and} \quad A_{m-1,0} = I(f) + \sum_{i=1}^{\infty} \alpha_i h_{m-1}^{2i} = \sum_{i=1}^{\infty} \alpha_i 2^{2i} h_m^{2i}$$

$\Rightarrow$ For $2^2 A_{m,0} = 2^2 I(f) + \sum_{i=1}^{\infty} \alpha_i 2^2 h_m^{2i}$ the terms for $i = 1$ agree. Hence:

$$2^2 A_{m,0} - A_{m-1,0} = (2^2 - 1)I(f) + \sum_{i=2}^{\infty} \beta_i h_m^{2i} \text{ for some } \beta_i$$

Define:

$$A_{m,1} := \frac{2^2 A_{m,0} - A_{m-1,0}}{2^2 - 1} = I(f) + \sum_{i=2}^{\infty} \tilde{\alpha}_i h_m^{2i}$$

10.9.1 General Romberg Formula

By induction for $q \geq 0$:

$$A_{m,q+1} = \frac{4^{q+1} A_{m,q} - A_{m-1,q}}{4^{q+1} - 1} = I(f) + \sum_{i=q+2}^{\infty} \alpha_{i,q+1} h_m^{2i}$$

10.9.2 Romberg Integration Table

We start with $A_{0,0}$ and $A_{1,0}$, i.e., (composite) trapezoidal rules with $h_0 = b - a$ and $h_1 = \frac{b-a}{2}$.

Similarly you can compute $A_{m,0}$, $m = 2, 3, \ldots$ by a trapezoidal rule.

We continue with the scheme:

$$\begin{matrix} A_{0,0} & & & & \\ A_{1,0} & A_{1,1} & & & \\ A_{2,0} & A_{2,1} & A_{2,2} & & \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} & \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{matrix}$$

Overall we have the error $A_{m,n} = I(f) + O(h_m^{2(n+1)})$.

10.9.3 Why Dyadic Steps?

Recall:

$$A_{m,0} = T_{2^m}(f) = \frac{1}{2} h_m f(a) + h_m \sum_{j=1}^{2^m - 1} f(a + j h_m) + \frac{1}{2} h_m f(b)$$

but we also have using $h_{m-1} = 2h_m$ that:

$$\frac{1}{2} A_{m-1,0} = T_{2^{m-1}}(f) = \frac{1}{2} h_m f(a) + h_m \sum_{j=1}^{2^{m-1} - 1} f(a + 2j h_m) + \frac{1}{2} h_m f(b)$$

So subtracting them yields:

$$A_{m,0} - \frac{1}{2} A_{m-1,0} = h_m \sum_{r=1}^{2^{m-1}} f(a + (2r - 1)h_m) = \frac{1}{2} U_{2^{m-1}}$$

Where $U_{2^{m-1}}$ is the midpoint rule on the new points, or:

The dyadic scheme lets us compute $A_{m,0} = \frac{1}{2}(A_{m-1,0} + U_{2^{m-1}})$, where the midpoint rule is computed only with the new points.

10.10 Improper Integrals

10.10.1 Computing Integrals - Overview

1. Jump discontinuity at $c \in [a, b] \Rightarrow$ split: $\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx \Rightarrow$ easy!
2. The case where $f(x)$ is unbounded, i.e., if $\lim_{x\to a^+} f(x) = \pm\infty$ (analogously $\lim_{x\to b^-} f(x) = \pm\infty$)
3. The case where a is unbounded, i.e.,

$$\lim_{a\to-\infty}\int_a^b f(x)\,dx = \int_{-\infty}^b f(x)\,dx \quad \text{analogously,} \quad \lim_{b\to\infty}\int_a^b f(x)\,dx = \int_a^\infty f(x)\,dx$$

   If both appear, use idea from 1.

10.10.2 Integrals with Infinite Functions

Not all of them exist, for example $\int_0^\infty \frac{1}{x}\,dx$ has no finite value.

Compare to $\int_0^1 \frac{1}{2\sqrt{x}}\,dx \Rightarrow$ the integral exists, antiderivative is $\sqrt{x}$.

We consider the special case where f is of the form (model):

$$f(x) = \frac{\phi(x)}{(x-a)^\mu}, \quad 0 \le \mu < 1$$

where $\phi(x)$ is bounded on $[a, b]$ and we need ϕ to be smooth, i.e., $\phi \in C^{p+1}([a, b])$. Let's denote the bound by $|\phi(x)| \le M$ for $x \in [a, b]$.

Then we can obtain plausibility:

$$\int_a^b f(x)\,dx \le M\int_a^b \frac{1}{(x-a)^\mu}\,dx = M\frac{1}{1-\mu}(x-a)^{1-\mu}\Big|_a^b = M\frac{(b-a)^{1-\mu}}{1-\mu}$$

10.10.3 Ansatz for this Model of Functions

For any $0 < \varepsilon < 1$ (and $a + \varepsilon < b$) we split:

$$I(f) = I_1 + I_2 = \int_a^{a+\varepsilon} \frac{\phi(x)}{(x-a)^\mu}\,dx + \int_{a+\varepsilon}^b \frac{\phi(x)}{(x-a)^\mu}\,dx$$

where:

- we can compute I_2 with methods we already know
- in I_1 we can use (ϕ smooth!) the Taylor expansion of ϕ:

$$\phi(x) = \Phi_p(x) + R_{p+1} = \sum_{k=0}^p \frac{1}{k!}\phi^{(k)}(a)(x-a)^k + \frac{(x-a)^{p+1}}{(p+1)!}\phi^{(p+1)}(\xi(x))$$

where $\xi(x) \in (a, x)$.

And obtain for:

$$f(x) = \frac{\Phi_p(x)}{(x-a)^\mu} + \frac{R_{p+1}(x)}{(x-a)^\mu} = \sum_{k=0}^{p} \frac{\phi^{(k)}(a)}{k!}(x-a)^{k-\mu} + \frac{(x-a)^{p+1-\mu}}{(p+1)!}\phi^{(p+1)}(\xi(x))$$

10.10.4 Solving the Integral I_1

The function f of the form above can now be integrated (on $[a, a+\varepsilon]$), they are just monomials. We get:

$$\begin{aligned} I_1 &= \varepsilon^{1-\mu} \sum_{k=0}^{p} \frac{\phi^{(k)}(a)}{k!(k+1-\mu)}\varepsilon^k + \frac{1}{(p+1)!}\int_a^{a+\varepsilon} (x-a)^{p+1-\mu}\phi^{(p+1)}(\xi(x))\,dx \\ &=: I_{1,p} + R_{1,p+1} \end{aligned}$$

We obtained an expression for I_1 where:

- we can compute $I_{1,p}$
- but we have to get an idea about the error $|R_{1,p+1}|$ we make

10.10.5 Error Bound for $|R_{1,p+1}|$

To bound:

$$R_{1,p+1} = \frac{1}{(p+1)!}\int_a^{a+\varepsilon} (x-a)^{p+1-\mu}\phi^{(p+1)}(\xi(x))\,dx$$

Idea: Replace/bound $\phi^{(p+1)}(\xi(x))$ by $C_\varepsilon = \max_{a\le x\le a+\varepsilon} |\phi^{(p+1)}(x)|$.

Then:

$$E_1 = |R_{1,p+1}| \le \frac{1}{(p+1)!}C_\varepsilon \int_a^{a+\varepsilon} (x-a)^{p+1-\mu}\,dx = \frac{\varepsilon^{p+2-\mu}}{(p+1)!(p+2-\mu)}C_\varepsilon$$

Note: The error bound is an increasing function in ε.

10.10.6 Back to the Complete Integral

For I_2 on $[a+\varepsilon, b]$ we can use any composite Newton-Cotes formula. Here we use:

- m subintervals, $h = \frac{b-a-\varepsilon}{m}$
- composite trapezoidal rule $T_n = \frac{h}{2}f(a+\varepsilon) + \sum_{j=1}^{n-1} f(a+\varepsilon+jh) + \frac{h}{2}f(b)$

We obtain an additional error (cf. QSS (9.12) and (9.26), or as a special case of Thm. 9.3):

$$E_T = -\frac{h^2(b-a-\varepsilon)|f''(\eta)|}{12}, \quad \eta \in (a+\varepsilon, b)$$

Goal: Choose ε and p such that the overall behavior:

- obtain a certain accuracy
- balance between E_1 and E_T

10.10.7 Unbounded Intervals

We again split:

$$\int_a^\infty f(x)\,dx = \int_a^c f(x)\,dx + \int_c^\infty f(x)\,dx = I_1 + I_2$$

Method 1:

- find a bound for the second integral I_2
- set c large enough such that we can omit the value of I_2
- use any method for the first term I_1

Method 2: Define $t = \frac{1}{x}$ and do a change of variable:

$$I_2 = \int_c^\infty f(x)\,dx = \int_0^{1/c} t^{-2} f\left(\frac{1}{t}\right)\,dt = \int_0^{1/c} g(t)\,dt$$

for $g(t) = t^{-2} f\left(\frac{1}{t}\right)$ ⇒ Solve that instead.

Note: $g(t)$ may be singular at $t = 0$, use Method from case 2.

Method 3: Gaussian Quadrature (next topic).

Example 32. Exam 2001, Problem 3 b/c
Compute:

$$I = \int_0^\infty \frac{e^{-x}}{1+x}\,dx = \int_0^A \frac{e^{-x}}{1+x}\,dx + \int_A^\infty \frac{e^{-x}}{1+x}\,dx = I_1 + I_2$$

by Method 1 using the composite trapezoidal rule $T(h)$ with step size h for the integral I_1.
1. Prove that there exist upper bounds for:

$$|I_1 - T(h)| \leq E_1(A, h) = \frac{5}{12} A h^2 \quad \text{and} \quad I_2 \leq E_2(A) = e^{-A}$$

2. Approximate the integral I to an error of $\varepsilon = 10^{-3}$ with as few as possible evaluations of the integrand (i.e., smallest n) in the trapezoidal rule ($h = \frac{A}{n}$).

Chapter 11

Gaussian Quadrature

11.1 Background: Orthogonal Polynomials

11.1.1 Jacobi Polynomials

We considered the orthogonal Jacobi polynomials on $[-1, 1]$, where we choose for some $\alpha, \beta > -1$ the weight:

$$w(x) = (1-x)^{\alpha}(1+x)^{\beta}$$

and construct orthogonal polynomials with respect to $(\cdot,\cdot)_w$. We even obtained a three-term recursion.

Special Cases:

- $\alpha = \beta = -\frac{1}{2}$ yields Chebyshev polynomials
- $\alpha = \beta = 0$ yields Legendre polynomials

11.1.2 Chebyshev Polynomials

On $[-1, 1]$ we use $w(x) = \frac{1}{\sqrt{1-x^2}}$ to obtain the Chebyshev polynomials:

$$T_k(x) = \cos(k \arccos(x)) = \cos(k\theta), \quad x = \cos\theta, \quad k \in \mathbb{N}$$

with:

- 3-term recursion: Starting with $T_0(x) = 1$, $T_1(x) = x$ we get:

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad k > 1$$

- $T_k(x) = 2^{k-1}x^k + \ldots$ lower order terms $\Rightarrow 2^{1-k}T_k$ is monic
- $(T_k, T_l)_w = \begin{cases} 0 & \text{for } k \neq l \\ \frac{\pi}{2} & \text{for } k = l > 0 \\ \pi & \text{for } k = l = 0 \end{cases}$
- $|T_k(x)| \leq 1$ for $x \in [-1, 1]$ and hence $\|T_k\|_\infty = \max_{-1\leq x\leq 1} |T_k(x)| = 1$

11.2 Real Zeros of Orthogonal Functions

Theorem 11.1: Real Zeros of Orthogonal Functions

Let $f \in C[a,b]$, $f \not\equiv 0$ and $n \in \mathbb{N}$ be given. Assume that:

$$(f,p)_w = 0 \text{ for all } p \in \mathbb{P}_{n-1}$$

(especially we could take $f = p_n$).
Then f changes sign at least n times on $[a,b]$, so we have n real zeros.

Proof. Suppose f has fewer than n sign changes on $[a,b]$. Let $x_1, \dots, x_k$ be the points where f changes sign, with $k < n$.
Consider the polynomial $q(x) = \prod_{i=1}^{k}(x - x_i) \in \mathbb{P}_k \subset \mathbb{P}_{n-1}$.
Since f and q have the same sign changes, the product $f(x)q(x)$ does not change sign on $[a,b]$.
Therefore: $(f,q)_w = \int_a^b f(x)q(x)w(x)\,dx \neq 0$.
But this contradicts our assumption that $(f,p)_w = 0$ for all $p \in \mathbb{P}_{n-1}$, since $q \in \mathbb{P}_{n-1}$.
Hence f must have at least n sign changes, and therefore at least n real zeros. □

11.3 Optimal Interpolation Nodes

11.3.1 Polynomial Interpolation Review

Remember: Interpolate $f \in C^{(n+1)}([-1,1])$ with $p_n \in \mathbb{P}_n$ means to satisfy:

$$p_n(x_i) = f(x_i), \quad i = 0, \dots, n$$

for some given nodes x_i.

We obtained the error bound for some $\xi = \xi(x) \in [-1,1]$:

$$f(x) - p_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)\omega(x), \quad \omega(x) = \prod_{i=0}^{n}(x - x_i)$$

where f is given and we have not much control over ξ.

But we can try to get $\omega(x)$ "small" when we are allowed to choose the interpolation nodes x_i.

We want to use known properties:

1. $\omega(x) \in \mathbb{P}_{n+1}$
2. $\omega(x)$ is monic
3. $\omega(x)$ has $n+1$ real zeros, namely the interpolation nodes $x_0, \dots, x_n$

11.3.2 Choose Chebyshev Polynomials!

If we set $\omega(x) = 2^{-n}T_{n+1}(x) \Rightarrow$ 1 & 2 are fulfilled.

So let's choose the nodes x_k as the zeros of T_{n+1}, i.e., $T_{n+1}(x_k) = 0$.

Zeros of T_{n+1}: We have $T_{n+1}(x) = \cos((n+1)\theta) = 0 \Rightarrow (n+1)\theta = \frac{\pi}{2} + k\pi$ for any $k \in \mathbb{Z}$.

We obtain the zeros for $k = 0, \dots, n$:

$$\theta_k = \frac{2k+1}{2(n+1)}\pi \quad \text{that is for } x_k = \cos\left(\frac{2k+1}{2(n+1)}\pi\right) \in [-1,1]$$

Since we know $\|T_k\|_\infty = 1$ we get $\|\omega\|_\infty = 2^{-n}$.

Can we do better in $\|\cdot\|_\infty$?

Theorem 11.2: Optimal Bound on $\omega(x)$

Let $\omega(x) = \prod_{i=0}^{n}(x - x_i) \in \mathbb{P}_n$.
Among all possible choices of distinct nodes $x_0, \dots, x_n \in [-1, 1]$ the resulting:

$$\|\omega\|_\infty = \max_{-1\leq x\leq 1} |\omega(x)|$$

is minimized if $\omega(x) = 2^{-n}T_{n+1}(x)$.

Proof. Suppose there exists another choice of nodes $y_0, \dots, y_n$ such that:

$$\tilde{\omega}(x) = \prod_{i=0}^{n}(x - y_i)$$

satisfies $\|\tilde{\omega}\|_\infty < 2^{-n}$.
Consider $p(x) = 2^{-n}T_{n+1}(x) - \tilde{\omega}(x)$.
Since both polynomials are monic of degree $n + 1$, we have $p \in \mathbb{P}_n$.
Now, $T_{n+1}(x)$ has $n + 1$ extremal points where $|T_{n+1}(x)| = 1$, alternating between +1 and −1.
At these points: $|2^{-n}T_{n+1}(x)| = 2^{-n} > \|\tilde{\omega}\|_\infty \geq |\tilde{\omega}(x)|$.
Therefore, $p(x)$ alternates sign at least $n + 1$ times, which means p has at least n zeros.
But $p \in \mathbb{P}_n$, so p can have at most n zeros unless $p \equiv 0$.
Since p has exactly n zeros and alternates sign $n + 1$ times, we must have $p \equiv 0$.
This contradicts our assumption that $\|\tilde{\omega}\|_\infty < 2^{-n}$. □

11.4 Gaussian Integration

Consider the problem:

$$I_w(f) = \int_a^b w(x)f(x)\,dx$$

We want to find a quadrature formula:

$$Q_w(f) = \sum_{i=1}^{n} \alpha_i f(x_i)$$

For some weights α_i, we still have to determine.

Remember: We call the highest polynomial degree m such that $I_w(p) = Q_w(p)$ for all $p \in \mathbb{P}_m$ its precision (degree).

For distinct $x_1, \dots, x_n$ we require the $Q_w(f)$ to have precision $n - 1$.

11.4.1 Idea: Reuse Lagrange Polynomials

We know that polynomials interpolate themselves: For any $p \in \mathbb{P}_{n-1}$ we get:

$$p(x) = \sum_{i=1}^{n} p(x_i)\ell_i(x), \quad \ell_i(x) = \prod_{\substack{j=1\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \quad i = 1, \dots, n$$

so by linearity of integration:

$$I_w(p) = \int_a^b w(x) \sum_{i=1}^n p(x_i)\ell_i(x)\,dx = \sum_{i=1}^n \alpha_i p(x_i)$$

where:

$$\alpha_i = \int_a^b w(x)\ell_i(x)\,dx$$

⇒ Q_w is exact for all $p \in \mathbb{P}_{n-1}$ if we choose α_i this way!

⇒ precision $n - 1$ accomplished

Since we can still choose $x_1, \ldots, x_n$ – can we increase the precision?

11.5 Exactness of Gaussian Quadrature

Theorem 11.3: Gaussian Quadrature Exactness

Let p_n be the orthogonal polynomial w.r.t. $(\cdot,\cdot)_w$. Let $x_1, \ldots, x_n$ be the distinct zeros of p_n (in $[a, b]$!). Then the quadrature formula:

$$I_w(f) \approx Q_w(f) = \sum_{i=1}^n \alpha_i f(x_i)$$

where:

$$\alpha_i = \int_a^b w(x)\ell_i(x)\,dx$$

has precision degree $2n - 1$.

Proof. We already know that the formula is exact for all $p \in \mathbb{P}_{n-1}$.
Now let $q \in \mathbb{P}_{2n-1}$. We can write:

$$q(x) = p_n(x)s(x) + r(x)$$

where $s, r \in \mathbb{P}_{n-1}$ (polynomial division).
Then:

$$\begin{aligned} I_w(q) &= \int_a^b w(x)[p_n(x)s(x) + r(x)]\,dx \\ &= \int_a^b w(x)p_n(x)s(x)\,dx + \int_a^b w(x)r(x)\,dx \\ &= (p_n, s)_w + I_w(r) \\ &= 0 + Q_w(r) \quad \text{(since } s \in \mathbb{P}_{n-1} \text{ and } r \in \mathbb{P}_{n-1}\text{)} \\ &= Q_w(r) \end{aligned}$$

Also:

$$\begin{aligned} Q_w(q) &= \sum_{i=1}^{n} \alpha_i q(x_i) \\ &= \sum_{i=1}^{n} \alpha_i [p_n(x_i)s(x_i) + r(x_i)] \\ &= \sum_{i=1}^{n} \alpha_i [0 \cdot s(x_i) + r(x_i)] \quad (\text{since } p_n(x_i) = 0) \\ &= \sum_{i=1}^{n} \alpha_i r(x_i) \\ &= Q_w(r) \end{aligned}$$

Therefore: $I_w(q) = Q_w(q)$ for all $q \in \mathbb{P}_{2n-1}$. □

11.6 Examples of Gaussian Quadrature

11.6.1 Simple Example

Consider the Legendre polynomial $p_1(x) = x$. Then $w \equiv 1$ on $[-1, 1]$ and we get $x_1 = 0$ and:

$$\alpha_1 = \int_{-1}^{1} w(x)\ell_1(x)\,dx = \int_{-1}^{1} 1\,dx = 2$$

This is the midpoint rule.

11.6.2 Second Example

For $n = 3$ and $w \equiv 1$ we have the Legendre polynomial $p_3(x) = x^3 - \frac{3}{5}x$ on $[-1, 1]$.

The zeros are $x_1 = -\sqrt{\frac{3}{5}}$, $x_2 = 0$, $x_3 = \sqrt{\frac{3}{5}}$.

The corresponding weights are $\alpha_1 = \alpha_3 = \frac{5}{9}$ and $\alpha_2 = \frac{8}{9}$.

11.7 Gaussian Quadrature Error Bound

If we write:

$$I_w(f) = Q_{w,n}(f) + E$$

remember that we had precision $2n - 1$.

Theorem 11.4: Gaussian Quadrature Error

Then the error E is of the form:

$$E = \frac{f^{(2n)}(\xi)}{(2n)!} \int_a^b w(x)\omega^2(x)\,dx, \quad \omega(x) = \prod_{i=1}^{n}(x - x_i)$$

for $f \in C^{(2n)}([a, b])$ and some $\xi \in [a, b]$.

Proof. Let $H_{2n-1}(x)$ be the Hermite interpolating polynomial of degree $2n-1$ that satisfies:

$$H_{2n-1}(x_i) = f(x_i), \quad H'_{2n-1}(x_i) = f'(x_i), \quad i = 1, \dots, n$$

The error in Hermite interpolation is:

$$f(x) - H_{2n-1}(x) = \frac{f^{(2n)}(\xi(x))}{(2n)!}\omega^2(x)$$

Since $Q_{w,n}$ is exact for polynomials of degree $2n-1$:

$$I_w(f) - Q_{w,n}(f) = I_w(f - H_{2n-1}) = \int_a^b w(x)\frac{f^{(2n)}(\xi(x))}{(2n)!}\omega^2(x)\,dx$$

By the integral mean value theorem:

$$= \frac{f^{(2n)}(\xi)}{(2n)!}\int_a^b w(x)\omega^2(x)\,dx$$

for some $\xi \in [a, b]$. □

This completes our comprehensive treatment of numerical integration, from basic Newton-Cotes formulas through Richardson extrapolation to the advanced Gaussian quadrature methods.

Chapter 12

Romberg Integration

Introduction

Numerical integration (or *quadrature*) is the art of estimating

$$I = \int_a^b f(x)\,dx$$

when an analytic antiderivative is unavailable or inconvenient. A good mental picture is slicing bread: each slice has a simple shape whose area you can sum, and thinner slices give a better approximation to the whole loaf. Romberg integration is a systematic way of cutting those slices thinner *and* blending successive approximations to cancel leading errors, much like stacking Russian dolls so that each larger doll hides the defects of the smaller one inside.

12.1 The Composite Trapezoidal Rule Revisited

Definition 12.1: Composite Trapezoidal Rule

Given $n = 2^k$ sub-intervals ($h_k = (b-a)/2^k$), the composite trapezoidal rule approximates the integral as

$$T_k = h_k\left[\frac{1}{2}f(a) + \sum_{i=1}^{2^k-1} f(a + ih_k) + \frac{1}{2}f(b)\right] = \frac{b-a}{2^{k+1}}\left[f(a) + 2\sum_{i=1}^{2^k-1} f(a + ih_k) + f(b)\right], \qquad h_k = \frac{b-a}{2^k}.$$

Its error admits an asymptotic expansion

$$T_k \;=\; I + c_2 h_k^2 + c_4 h_k^4 + c_6 h_k^6 + \cdots.$$

Think of T_k as a photograph that is blurry by an amount proportional to h_k^2; halving h_k halves the blur squared but leaves a faint ghost. We would like to remove that ghost *without* taking ever-smaller pictures.

12.2 Richardson Extrapolation

Richardson extrapolation blends two pictures of different resolutions and subtracts the shared blur:

$$\mathcal{A}_{k,1} \;=\; \frac{4T_k - T_{k-1}}{3}, \qquad k \geq 1.$$

Here T_k uses step h_k while T_{k-1} uses $2h_k$. Because the leading h^2 error cancels, $\mathcal{A}_{k,1}$ is $\mathcal{O}(h_k^4)$. Each application of the extrapolation "crank" raises the order by 2.

12.3 The Romberg Table

Romberg integration applies Richardson extrapolation *recursively* on a triangular table (Table 12.1).

Definition 12.2: Romberg Method

Let T_k denote the composite trapezoidal rule with 2^k subintervals. The Romberg tableau entries $\mathcal{A}_{k,j}$ are defined recursively as:

$$\mathcal{A}_{k,0} = T_k, \qquad k = 0, 1, \ldots$$

$$\mathcal{A}_{k,j} = \frac{4^j \mathcal{A}_{k,j-1} - \mathcal{A}_{k-1,j-1}}{4^j - 1}, \qquad j = 1, \ldots, k.$$

Here, k is the row (refinement level), and j is the column (extrapolation depth). The diagonal entry $\mathcal{A}_{k,k}$ achieves error $\mathcal{O}(h_k^{2(k+1)})$.

Corollary 3: (Recursive Trapezoid update). The first column of the Romberg table ($j = 0$) can be computed recursively

$$A_{k,0} = \frac{1}{2}\mathcal{A}_{k-1,0} + h_k \sum_{i=1}^{2^{k-1}} f\left(a + (2i-1)h_k\right), \qquad k \geq 1.$$

This allows us to easily compute the trapezoidal rule for each row without recalculating all previous values.

Each new row halves the step size (more bread slices); each new column cancels the dominant error term (removes blur). The procedure is like climbing a staircase: horizontal steps refine the mesh, vertical steps polish the estimate.

Table 12.1: Romberg tableau for $k_{\max} = 3$.

$k \backslash j$	0	1	2	3
0	$\mathcal{A}_{0,0}$			
1	$\mathcal{A}_{1,0}$	$\mathcal{A}_{1,1}$		
2	$\mathcal{A}_{2,0}$	$\mathcal{A}_{2,1}$	$\mathcal{A}_{2,2}$	
3	$\mathcal{A}_{3,0}$	$\mathcal{A}_{3,1}$	$\mathcal{A}_{3,2}$	$\mathcal{A}_{3,3}$

More bread slices (rows) and *ghost removal* (columns) lead to **high-order accuracy** at modest cost.

12.4 Romberg Algorithm

The complete Romberg algorithm consists of two main components: computing the trapezoidal rule estimates and applying Richardson extrapolation.

Algorithm 8 Trapezoidal Rule with 2^k intervals

```
function TRAPEZOID($f, a, b, n$)
    $h \leftarrow (b - a)/n$
    $sum \leftarrow \frac{1}{2}(f(a) + f(b))$
    for $i = 1$ to $n - 1$ do
        $sum \leftarrow sum + f(a + ih)$
    return $h \cdot sum$
```

Algorithm 9 Refined Trapezoidal Rule (using previous computation)

```
function REFINEDTRAPEZOID($f, a, b, n$)
    $h \leftarrow (b - a)/n$
    $sum \leftarrow 0$
    for $i = 1$ to $n/2$ do
        $sum \leftarrow sum + f(a + (2i - 1)h)$
    return $\frac{1}{2} \cdot T_{prev} + h \cdot sum$
```

Algorithm 10 Complete Romberg Integration

```
function ROMBERGINTEGRATION($f, a, b, k_{max}, \varepsilon$)
    Initialize array $\mathcal{A} \leftarrow$ zeros($k_{max} + 1, k_{max} + 1$)
    $\mathcal{A}_{[0][0]} \leftarrow$ TRAPEZOID($f, a, b, 1$)
    for $k = 1$ to $k_{max}$ do
        $\mathcal{A}_{[k][0]} \leftarrow$ REFINEDTRAPEZOID($f, a, b, 2^k$)
        for $j = 1$ to $k$ do
            $\mathcal{A}_{[k][j]} \leftarrow \frac{4^j \mathcal{A}_{[k][j-1]} - \mathcal{A}_{[k-1][j-1]}}{4^j - 1}$
        if $k > 0$ and $|\mathcal{A}_{[k][k]} - \mathcal{A}_{[k-1][k-1]}| < \varepsilon$ then
            return $\mathcal{A}_{[k][k]}$
    return $\mathcal{A}_{[k_{max}][k_{max}]}$
```

12.5 Adaptive Romberg

When f is highly variable on sub-intervals, naïvely halving the global step wastes work on easy regions. Adaptive Romberg splits $[a, b]$ recursively, applying Algorithm 11 on sub-intervals until the local error estimate meets tolerance, much like a photographer zooming in only where the picture is blurry.

Algorithm 11 Romberg Integration

```
Choose $k_{\max}$ (max depth) and tolerance $\varepsilon$
$\mathcal{A}_{0,0} \leftarrow$ TRAPEZOID$(f, a, b, 1)$
for $k = 1$ to $k_{\max}$ do
    $\mathcal{A}_{k,0} \leftarrow$ REFINEDTRAPEZOID$(f, a, b, 2^k)$
    for $j = 1$ to $k$ do
        $\mathcal{A}_{k,j} \leftarrow \dfrac{4^j \mathcal{A}_{k,j-1} - \mathcal{A}_{k-1,j-1}}{4^j - 1}$
    if $|\mathcal{A}_{k,k} - \mathcal{A}_{k-1,k-1}| < \varepsilon$ then
        return $\mathcal{A}_{k,k}$                                  ▷ converged
return $\mathcal{A}_{k_{\max},k_{\max}}$
```

Remark 21. Implementation Notes

- **Caching:** T_k re-uses all f evaluations from T_{k-1}; store them to avoid recomputation.
- **Round-off:** For large k the increment $\mathcal{A}_{k,0} - \mathcal{A}_{k-1,0}$ may underflow; stop refinement when machine precision dominates.
- **Vectorisation:** Modern CPUs/GPUs can evaluate $f(\mathbf{x})$ on a mesh in parallel, significantly speeding the first few rows.

12.6 Accuracy and Convergence

Under suitable smoothness ($f \in C^{2(k+1)}[a, b]$),

$$\mathcal{A}_{k,k} = I + \mathcal{O}\left(h_k^{2(k+1)}\right).$$

Thus every diagonal step increases the algebraic accuracy by two orders. Because $h_k = 2^{-k}(b - a)$, the error decays *super-geometrically*; doubling k squares the convergence factor.

Example 33. Romberg Integration Example

Integrate $I = \int_0^1 e^{-x^2}\, dx$ (the error function half-integral).

Table 12.6 lists $\mathcal{A}_{k,k}$ for $k = 0 \dots 4$. One sees quadratic accuracy on T_k (column 0) and quartic, sextic, etc. on higher columns, reaching machine precision with only 31 function evaluations.

k	$\mathcal{A}_{k,0}$	$\mathcal{A}_{k,1}$	$\mathcal{A}_{k,2}$	$\mathcal{A}_{k,3}$	$\mathcal{A}_{k,4}$
0	0.6839397206				
1	0.7471308777	0.7651801875			
2	0.7791906969	0.7853981635	0.7858731139		
3	0.7886993009	0.7904804897	0.7907357145	0.7907767155	
4	0.7911868057	0.7913186162	0.7913310420	0.7913316310	0.7913316515

Ground truth is 0.74682413 ...; the $k = 4$ diagonal estimate matches to full double-precision.

Romberg integration combines the simplicity of the trapezoidal rule with the power of Richardson extrapolation, delivering very high-order accuracy at modest cost for smooth integrands.

Remember the three guiding images:

1. **Bread slices** (*mesh refinement*).
2. **Ghost removal** (*extrapolation to cancel error*).
3. **Staircase ascent** (*iterating the process*).

When the function is smooth, a few steps up the Romberg staircase bring you astonishingly close to the true integral.

Part VI

Ordinary Differential Equations

Chapter 13

Linear Multistep Methods

13.1 Introduction and Motivation

Many initial-value problems

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0, \tag{13.1}$$

must be solved repeatedly in science and engineering. One-step methods (Euler, Runge-Kutta) march forward using *only the most recent point*. *Linear multistep methods (LMMs)* reuse computed information from several previous time levels. Think of them as driving a car while glancing not only at the speedometer right now, but also remembering the readings from the last few seconds: with more historical data you can estimate the distance travelled more accurately without pressing the gas pedal again (Figure 13.1).

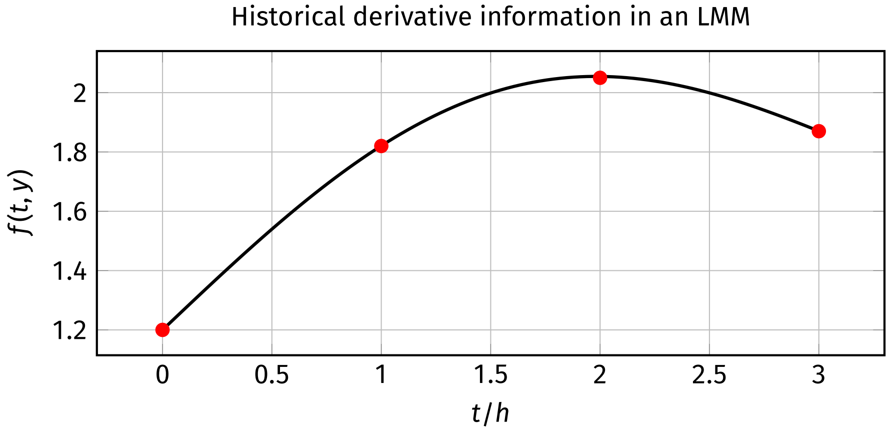


Figure 13.1: An LMM predicts the future integral of f using derivative values stored from several past steps.

Compared with high-order Runge-Kutta methods, multistep schemes:

- offer the **same maximal algebraic order** for fewer expensive evaluations of f;
- separate *accuracy* (the number of steps k) from the *cost per step* (still one right-hand side evaluation for explicit methods);
- trade accuracy for *reduced stability*, making the analysis of root locations in the complex plane a central theme.

13.2 General Formulation of a k-Step Linear Multistep Method

Definition 13.1: Linear Multistep Method

Let $\{t_n\}_{n\geq 0}$ be an equidistant grid with fixed step $h = t_{n+1} - t_n > 0$. A *k-step linear multistep method (LMM)* approximates the solution of (13.1) through the recurrence

$$\sum_{j=0}^{k} \alpha_j\, y_{n+j} = h \sum_{j=0}^{k} \beta_j\, f_{n+j}, \qquad \text{where } f_{n+j} = f(t_{n+j}, y_{n+j}), \tag{13.2}$$

with constant real coefficients α_j, β_j satisfying $\alpha_k \neq 0$ and $\gcd\big(\{\alpha_j\}_0^k\big) = 1$. We call the method *explicit* if $\beta_k = 0$ and *implicit* otherwise.

13.2.1 Characteristic Polynomials and Root Condition

Define the polynomials

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

The **root condition** states that all roots of ρ lie in $|\zeta| \leq 1$; any root on the unit circle must be *simple* (i.e., has multiplicity one $j = 1$). This condition is essential for zero-stability and therefore convergence (section 13.4).

13.3 Error Analysis

13.3.1 Local Truncation Error, Order and Consistency

Definition 13.2: Local Truncation Error (LTE)

Suppose the exact solution $y(t)$ of (13.1) is inserted into (13.2). The resulting defect

$$\tau_{n+k} := \sum_{j=0}^{k} \alpha_j\, y(t_{n+j}) - h \sum_{j=0}^{k} \beta_j\, y'(t_{n+j})$$

is called the *local truncation error*. The method is of (algebraic) *order* p if $\tau_{n+k} = \mathcal{O}(h^{p+1})$ as $h \to 0$.

Definition 13.3: Consistency

A linear multistep method is *consistent* if its local truncation error $\tau_{n+k} \to 0$ as $h \to 0$ for all sufficiently smooth $y(t)$. Equivalently, the method is consistent if

$$\sum_{j=0}^{k} \alpha_j = 0 \qquad \text{and} \qquad \sum_{j=0}^{k} j\, \alpha_j = \sum_{j=0}^{k} \beta_j.$$

Example 34. Exam H2022: Problem 5 d)

Consider the method

$$3y_{n+2} - 4y_{n+1} + y_n = 2hf_{n+2}.$$

This is a two-step ($k = 2$) linear multistep method with coefficients:

$$\alpha_0 = 1, \quad \alpha_1 = -4, \quad \alpha_2 = 3; \qquad \beta_0 = 0, \quad \beta_1 = 0, \quad \beta_2 = 2.$$

Consistency: A linear multistep method is consistent if

$$\sum_{j=0}^{k} \alpha_j = 0 \qquad \text{and} \qquad \sum_{j=0}^{k} j\,\alpha_j = \sum_{j=0}^{k} \beta_j.$$

Compute:

$$\begin{aligned}
\sum_{j=0}^{2} \alpha_j &= 1 + (-4) + 3 = 0, \\
\sum_{j=0}^{2} j\,\alpha_j &= 0 \cdot 1 + 1 \cdot (-4) + 2 \cdot 3 = -4 + 6 = 2, \\
\sum_{j=0}^{2} \beta_j &= 0 + 0 + 2 = 2.
\end{aligned}$$

Both conditions are satisfied, so the method is **consistent**.
Order: To find the order p, expand the local truncation error:

$$\tau_{n+2} = \sum_{j=0}^{2} \alpha_j y(t_{n+j}) - h \sum_{j=0}^{2} \beta_j y'(t_{n+j}).$$

Expand $y(t_{n+j})$ and $y'(t_{n+j})$ in Taylor series about t_n:

$$\begin{aligned}
y(t_{n+1}) &= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{6} y'''(t_n) + \mathcal{O}(h^4), \\
y(t_{n+2}) &= y(t_n) + 2hy'(t_n) + 2h^2 y''(t_n) + \frac{4h^3}{3} y'''(t_n) + \mathcal{O}(h^4), \\
y'(t_{n+2}) &= y'(t_n) + 2hy''(t_n) + 2h^2 y'''(t_n) + \mathcal{O}(h^3).
\end{aligned}$$

Plug in and collect terms:

$$\begin{aligned}
\tau_{n+2} &= \alpha_0 y(t_n) + \alpha_1 y(t_{n+1}) + \alpha_2 y(t_{n+2}) - h\beta_2 y'(t_{n+2}) \\
&= y(t_n) - 4y(t_{n+1}) + 3y(t_{n+2}) - 2hy'(t_{n+2}) \\
&= [1 - 4 + 3]\, y(t_n) \\
&\quad + [-4h + 3 \cdot 2h]\, y'(t_n) \\
&\quad + \left[-4\frac{h^2}{2} + 3 \cdot 2h^2\right] y''(t_n) \\
&\quad + \left[-4\frac{h^3}{6} + 3 \cdot \frac{4h^3}{3}\right] y'''(t_n) \\
&\quad - 2h\left[y'(t_n) + 2hy''(t_n) + 2h^2 y'''(t_n)\right] + \mathcal{O}(h^4)
\end{aligned}$$

Now, collect like terms:

$$\begin{aligned}
y(t_n) :&\quad 1 - 4 + 3 = 0 \\
y'(t_n) :&\quad -4h + 6h - 2h = 0 \\
y''(t_n) :&\quad -2h^2 + 6h^2 - 4h^2 = 0 \\
y'''(t_n) :&\quad -\frac{2}{3}h^3 + 4h^3 - 4h^3 = -\frac{2}{3}h^3
\end{aligned}$$

So,

$$\tau_{n+2} = -\frac{2}{3}h^3 y'''(t_n) + \mathcal{O}(h^4)$$

Thus, the method is of order $p = 2$.
Convergence: For convergence, we also require **zero-stability** (the root condition). That is, all roots of the characteristic polynomial $\rho(\zeta) = \alpha_0 + \alpha_1\zeta + \alpha_2\zeta^2$ must satisfy $|\zeta| \leq 1$, and any root on the unit circle must be simple. The characteristic polynomial is

$$\rho(\zeta) = 1 - 4\zeta + 3\zeta^2.$$

The roots are found using the quadratic formula:

$$\zeta_{1,2} = \frac{4 \pm \sqrt{(-4)^2 - 4 \cdot 3 \cdot 1}}{2 \cdot 3} = \frac{4 \pm \sqrt{16 - 12}}{6} = \frac{4 \pm 2}{6} = \frac{6}{6} = 1, \quad \frac{2}{6} = \frac{1}{3}.$$

The roots are $\zeta_1 = 1$ and $\zeta_2 = \frac{1}{3}$. Both roots satisfy $|\zeta| \leq 1$, and since they are distinct, the method is **zero-stable**. Therefore, the method is **convergent**.

13.3.2 Global Discretization Error

By expanding $y(t)$ in powers of h and matching coefficients, one finds

$$\tau_{n+1}^{(\mathrm{AB}k)} = C_k h^{k+1} y^{(k+1)}(\xi), \quad C_1 = \frac{1}{2}, C_2 = -\frac{5}{12}, C_3 = -\frac{3}{8}, \ldots$$

so the *global* error is $\mathcal{O}(h^k)$.

Definition 13.4: Global Discretization Error

The *global discretization error* is defined as

$$e_n := y(t_n) - y_n.$$

It is bounded by the LTE:

$$|e_n| \leq C_k\, h^{k+1}\, M,$$

where M is a bound on the $(k+1)$-th derivative of y in the interval $[t_0, t_n]$.

13.3.3 Zero-Stability

Definition 13.5: Zero-Stability

Consider the homogeneous difference equation obtained from (13.2) by setting $f \equiv 0$:

$$\sum_{j=0}^{k} \alpha_j\, y_{n+j} = 0.$$

The LMM is *zero-stable* if its solution remains bounded as $n \to \infty$ whenever the initial vector $(y_0, \ldots, y_{k-1})$ is fixed.

Zero-stability is equivalent to the root condition of subsection 13.2.1.

13.3.4 Absolute Stability

Applying an LMM to the test equation $y' = \lambda y$ ($\Re\lambda < 0$) yields an amplification factor $R(z) := \rho(1+z)/\sigma(1+z)$ with $z = \lambda h$. The set $\{z \in \mathbb{C} : |R(z)| \le 1\}$ is the *stability region*. Figure 13.2 illustrates why explicit multistep methods are unsuitable for stiff ODEs: their stability region is always bounded.

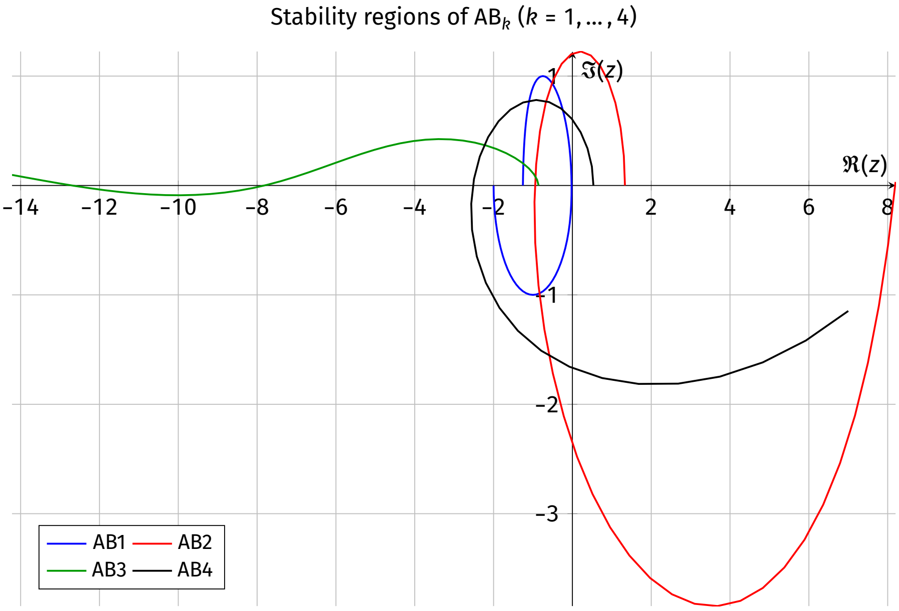


Figure 13.2: The explicit Adams-Bashforth stability windows shrink as the order k grows. None include the entire left half-plane $\Re(z) \le 0$, so AB methods are *not* A-stable.

13.4 Convergence Theory: Dahlquist's Equivalence Theorem

The fundamental question for any numerical method is: *does the computed solution approach the true solution as the step size decreases?* For linear multistep methods, this convergence behavior is completely characterized by Dahlquist's celebrated equivalence theorem.

Definition 13.6: Convergence

A linear multistep method *converges* if for any initial value problem (13.1) with sufficiently smooth solution, the numerical approximation satisfies

$$\lim_{\substack{h\to 0\\ nh=t-t_0}} y_n = y(t)$$

for all $t \in [t_0, t_f]$, provided the starting values satisfy $y_j \to y(t_j)$ as $h \to 0$ for $j = 0, 1, \dots, k-1$.

The power of Dahlquist's theorem lies in reducing the convergence question to two simpler, verifiable conditions:

Theorem 13.7: Dahlquist's Equivalence Theorem

A linear multistep method is convergent if and only if it is both:

1. **Consistent**: $\tau_{n+k} \to 0$ as $h \to 0$ (equivalently, the conditions in **??** hold);
2. **Zero-stable**: the root condition from subsection 13.2.1 is satisfied.

Proof sketch. **(⇒) Necessity:** If the method converges, then applying it to simple test problems (e.g., $y' = 0$ with $y(0) = 1$) shows that:

- The local truncation error must vanish as $h \to 0$, giving consistency;
- Perturbations in initial data cannot grow unboundedly, ensuring zero-stability.

(⇐) Sufficiency: Let $e_n = y(t_n) - y_n$ denote the global error. From the definitions of the exact solution and numerical method, e_n satisfies:

$$\sum_{j=0}^{k} \alpha_j e_{n+j} = \tau_{n+k} + h \sum_{j=0}^{k} \beta_j [f(t_{n+j}, y(t_{n+j})) - f(t_{n+j}, y_{n+j})].$$

Using the Lipschitz continuity of f and zero-stability (which bounds the fundamental solution of the homogeneous equation), a discrete Grönwall inequality yields:

$$|e_n| \leq C \left(\max_{0 \leq j \leq k-1} |e_j| + \sum_{m=0}^{n-k} |\tau_{m+k}| \right).$$

With consistent starting values ($e_j \to 0$) and consistency ($|\tau_{m+k}| = \mathcal{O}(h^{p+1})$), we obtain $|e_n| = \mathcal{O}(h^p)$, proving convergence. □

Remark 22. D

hlquist's theorem reveals why *both* conditions are essential:

- **Consistency alone** ensures that each step introduces only small errors, but without zero-stability, these errors can accumulate catastrophically;
- **Zero-stability alone** controls error propagation but is meaningless if large errors are introduced at each step.

The interplay between these conditions makes convergence analysis for multistep methods more subtle than for one-step methods.

13.4.1 Practical Implications

In practice, Dahlquist's theorem provides a systematic approach to analyzing any proposed multistep method:

1. **Check consistency** by verifying the algebraic conditions in **??**;
2. **Verify zero-stability** by computing the roots of the characteristic polynomial $\rho(\zeta)$ and ensuring they satisfy the root condition;
3. If both hold, the method is guaranteed to converge with order equal to the consistency order.

This explains why high-order Adams-Bashforth methods ($k \geq 5$) are avoided in practice: while they achieve high consistency order, they violate the root condition and hence fail to converge.

Example 35. Exam H2018: Problem 2 a)

Question: We approximate the solution to the initial value problem

$$y' = f(t, y), \quad y(t_0) = y_0, \quad t \in [t_0, t_f]$$

by a linear multistep method. Given k initial approximations, $y_0, \dots, y_{k-1}$, the general format of such a method is

$$\sum_{j=0}^{k} \alpha_j y_{n+j} = h \sum_{j=0}^{k} \beta_j f(t_{n+j}, y_{n+j}), \quad n \geq 0,$$

where $t_i = t_0 + ih$ and $\alpha_k \neq 0$.

What does it mean that the method converges at a point $t \in [t_0, t_f]$?
Answer: A linear multistep method *converges at a point* $t \in [t_0, t_f]$ if the numerical approximation approaches the exact solution as the step size decreases to zero. Formally, the method converges at t if

$$\lim_{\substack{h\to 0 \\ nh = t - t_0}} y_n = y(t),$$

where y_n is the numerical approximation at time $t_n = t_0 + nh$ and $y(t)$ is the exact solution of the initial value problem at time t.
This convergence must hold provided that:

- The starting values are consistent: $\lim_{h\to 0} y_j = y(t_j)$ for $j = 0, 1, \dots, k-1$;
- The solution $y(t)$ is sufficiently smooth in the interval $[t_0, t_f]$.

By Dahlquist's equivalence theorem (**??**), convergence occurs if and only if the method is both *consistent* and *zero-stable*.

13.5 The Adams Family of Multistep Methods

The most widely used multistep schemes interpolate the derivative $f(t, y)$ by low-degree polynomials.

13.5.1 Adams-Bashforth Methods (Explicit)

Definition 13.8: Adams-Bashforth Method

For $k \geq 1$ the k-step *Adams-Bashforth* (ABk) method reads

$$y_{n+1} = y_n + h \sum_{j=0}^{k-1} b_j f_{n-j}, \qquad b_j = \int_0^1 \ell_j(\theta)\, d\theta, \tag{13.3}$$

where ℓ_j denotes the Lagrange basis associated with the nodes $\theta = -j, \dots, 0$. The method is explicit because $b_{-1} = 0$.

Derivation. Integrate (13.1) over $[t_n, t_{n+1}]$:

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

Substitute $\theta = \frac{t - t_n}{h} \in [0, 1]$, so $t = t_n + \theta h$ and $dt = h\, d\theta$:

$$y_{n+1} = y_n + h \int_0^1 f(t_n + \theta h,\ y(t_n + \theta h))\, d\theta.$$

Approximate $f(t, y(t))$ by a degree-$(k-1)$ polynomial interpolant through the points (t_{n-j}, f_{n-j}):

$$f(t_n + \theta h,\ y(t_n + \theta h)) \approx p_{k-1}(t_n + \theta h) = \sum_{j=0}^{k-1} f_{n-j}\, \ell_j(\theta),$$

where $\ell_j(\theta)$ are the Lagrange basis polynomials for nodes $\theta = -j, \dots, 0$. Inserting this into the integral gives

$$y_{n+1} \approx y_n + h \sum_{j=0}^{k-1} f_{n-j} \int_0^1 \ell_j(\theta)\, d\theta,$$

which is precisely the Adams-Bashforth formula (13.3). □

Coefficients and Order. The first four members (plotted later in Figure 13.2) are:

$$\begin{aligned}
&\text{AB1: } y_{n+1} = y_n + h\, f_n,\\
&\text{AB2: } y_{n+1} = y_n + h\Big(\frac{3}{2} f_n - \frac{1}{2} f_{n-1}\Big),\\
&\text{AB3: } y_{n+1} = y_n + h\Big(\frac{23}{12} f_n - \frac{16}{12} f_{n-1} + \frac{5}{12} f_{n-2}\Big),\\
&\text{AB4: } y_{n+1} = y_n + h\Big(\frac{55}{24} f_n - \frac{59}{24} f_{n-1} + \frac{37}{24} f_{n-2} - \frac{9}{24} f_{n-3}\Big).
\end{aligned}$$

ABk attains order k for $k \le 5$, but loses zero-stability for $k \ge 5$.

Algorithm 12 Adams-Bashforth k-step Method

Require: Current value y_n, derivative history $\{f_{n-j}\}_{j=0}^{k-1}$, step size h, coefficients $\{b_j\}_{j=0}^{k-1}$

Ensure: Approximation $y_{n+1} \approx y(t_{n+1})$

```
function AdamsBashforth($y_n, \{f_{n-j}\}, h, \{b_j\}, k$)
    $s \leftarrow 0$
    for $j = 0$ to $k - 1$ do
        $s \leftarrow s + b_j \cdot f_{n-j}$
    $y_{n+1} \leftarrow y_n + h\, s$
    return $y_{n+1}$
```

13.5.2 Adams-Moulton Methods (Implicit)

Replacing the explicit extrapolation by an *implicit* interpolation that includes the yet unknown value f_{n+1} gives the k-step *Adams-Moulton* (AMk) formula

$$y_{n+1} = y_n + h \sum_{j=-1}^{k-1} a_j f_{n-j}, \qquad a_{-1} = \int_0^1 \tilde{\ell}_{-1}(\theta)\, d\theta \neq 0.$$

AMk is of order $k + 1$ and A-stable for $k \le 2$. Because it is implicit, a nonlinear solve is required at every step.

13.5.3 Predictor-Corrector Pairs

A popular compromise first *predicts* y_{n+1}^{P} with ABk, evaluates f_{n+1}^{P}, and then *corrects* using AMk. The pair (ABk, AM$(k-1)$) has order k, requires just one right-hand side evaluation more than ABk, and behaves robustly on mildly stiff problems.

Example 36. Non-stiff ODE with AB2

Solve the non-stiff problem

$$y' = -2y + \mathrm{e}^{-t}, \qquad y(0) = 1$$

on $t \in [0, 4]$ with AB2 and step size $h = \frac{1}{5}$.

The analytical solution is

$$y(t) = \tfrac{1}{3}\big(2\mathrm{e}^{-2t} + 1\big).$$

The AB2 iterates are obtained with a single Euler start-up step.

Figure 13.3 confirms the predicted second-order convergence.

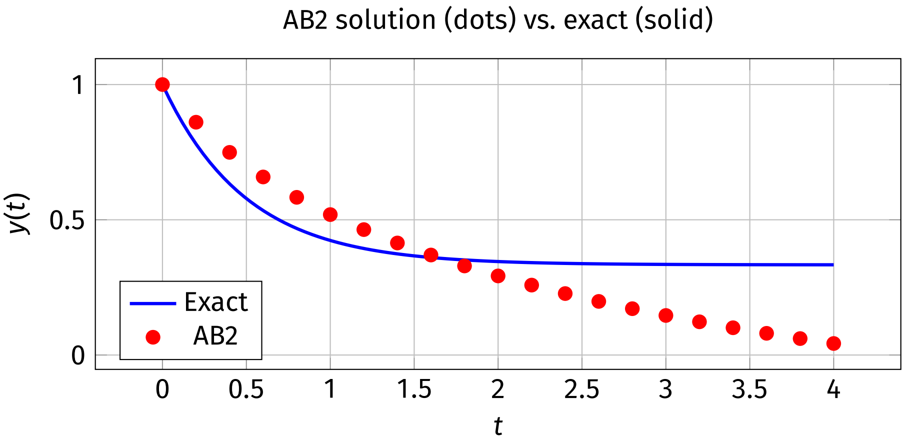


Figure 13.3: AB2 solution of 36 (dots) versus exact solution (solid). The global error behaves like $\mathcal{O}(h^2)$.

Summary

Linear multistep methods reach high algebraic order with only *one* function evaluation per step. Convergence demands consistency and the root condition. The explicit Adams-Bashforth family excels on non-stiff problems, while implicit Adams-Moulton and BDF schemes cope with stiffness at the price of solving nonlinear equations.

Chapter 14

Runge-Kutta Methods

14.1 Introduction and Motivation

Runge-Kutta (RK) methods are a family of one-step methods for solving initial value problems of the form

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0.$$

Unlike linear multistep methods, RK methods use only the current value (t_n, y_n) to advance the solution, but achieve high order by evaluating f at several intermediate points within each step. The classical fourth-order Runge-Kutta method (RK4) is especially popular due to its balance of accuracy and simplicity.

14.2 General Formulation of s-Stage Runge-Kutta Methods

An s-stage Runge-Kutta method advances the solution from y_n to y_{n+1} as follows:

$$\begin{aligned} k_i &= f\left(t_n + c_i h,\ y_n + h\sum_{j=1}^{s} a_{ij}k_j\right), \qquad i = 1, \dots, s, \\ y_{n+1} &= y_n + h\sum_{i=1}^{s} b_i k_i. \end{aligned}$$

The coefficients a_{ij}, b_i, and c_i fully determine the method.

14.3 The Butcher Tableau

The coefficients of a Runge-Kutta method are conveniently summarized in the *Butcher tableau*:

$$\begin{array}{c|ccc} c_1 & a_{11} & \cdots & a_{1s} \\ \vdots & \vdots & & \vdots \\ c_s & a_{s1} & \cdots & a_{ss} \\ \hline & b_1 & \cdots & b_s \end{array}$$

where a_{ij}, b_i, and c_i are the method's parameters. Explicit methods have $a_{ij} = 0$ for $j \geq i$.

14.3.1 Example: Classical RK4

The classical fourth-order Runge-Kutta method (RK4) has the tableau:

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & 1/6 & 1/3 & 1/3 & 1/6 \end{array}$$

This method is widely used for its accuracy and simplicity.

14.4 Order Conditions and Properties

The accuracy of a Runge-Kutta method is determined by its coefficients. A method is said to be of order p if its local truncation error is $\mathcal{O}(h^{p+1})$. Achieving higher order requires more stages and the satisfaction of increasingly intricate algebraic conditions, known as Butcher's order conditions.

For a Runge-Kutta method to be *consistent* (i.e., at least first-order accurate), the coefficients must satisfy

$$\sum_{i=1}^{s} b_i = 1,$$

which ensures the method reproduces the exact solution for constant functions.

More generally, the order conditions up to $p = 4$ are as follows:

- **Order 1 (Consistency):**

$$\sum_{i=1}^{s} b_i = 1$$

- **Order 2:**

$$\sum_{i=1}^{s} b_i c_i = \frac{1}{2}$$

- **Order 3:**

$$\sum_{i=1}^{s} b_i c_i^2 = \frac{1}{3}$$

$$\sum_{i=1}^{s} b_i \sum_{j=1}^{s} a_{ij} c_j = \frac{1}{6}$$

- **Order 4:**

$$\sum_{i=1}^{s} b_i c_i^3 = \frac{1}{4}$$

$$\sum_{i=1}^{s} b_i c_i \sum_{j=1}^{s} a_{ij} c_j = \frac{1}{8}$$

$$\sum_{i=1}^{s} b_i \left(\sum_{j=1}^{s} a_{ij} c_j^2 \right) = \frac{1}{12}$$

$$\sum_{i=1}^{s} b_i \sum_{j=1}^{s} a_{ij} \sum_{k=1}^{s} a_{jk} c_k = \frac{1}{24}$$

14.5 Stability and Applications

Stability analysis for RK methods often involves applying the method to the test equation $y' = \lambda y$. The stability region is the set of $z = \lambda h$ for which the numerical solution does not grow. Explicit RK methods are not suitable for stiff problems, while implicit RK methods (e.g., Gauss, Radau, Lobatto) are designed for such cases.

14.6 Exercises and Solutions

Problem 5. (Ordinary Differential Equations)

a) Stages and Formulation from Butcher Tableau

Given the tableau (reformatted for clarity):

$$\begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

There are $s = 4$ stages.

The method is:

$$\begin{aligned} K_1 &= f(t_n, y_n) \\ K_2 &= f\left(t_n + \frac{1}{2}h,\ y_n + \frac{1}{2}hK_1\right) \\ K_3 &= f\left(t_n + \frac{1}{2}h,\ y_n + \frac{1}{2}hK_2\right) \\ K_4 &= f(t_n + h,\ y_n + hK_3) \\ y_{n+1} &= y_n + \frac{h}{6}(K_1 + 2K_2 + 2K_3 + K_4) \end{aligned}$$

b) Order 3 Conditions for 3-stage RK

Given tableau:

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 1 & a_{31} & a_{32} & 0 \\ \hline & b_1 & b_2 & b_3 \end{array}$$

Order 3 conditions:

$$\begin{aligned} b_1 + b_2 + b_3 &= 1 \\ b_2 \cdot \frac{1}{2} + b_3 \cdot 1 &= \frac{1}{2} \\ b_2 \cdot \left(\frac{1}{2}\right)^2 + b_3 \cdot 1^2 &= \frac{1}{3} \\ b_3 a_{32} \cdot \frac{1}{2} &= \frac{1}{6} \end{aligned}$$

Gives the coefficients:

$$a_{31} = -1, \quad a_{32} = 2, \quad b_1 = \frac{1}{6}, \quad b_2 = \frac{2}{3}, \quad b_3 = \frac{1}{6}$$

c) Maximum Order for Explicit RK with $s = 4$

For the test equation $y' = y$, $y(0) = y_0$, the Taylor expansion of $y(h)$ is:

$$y(h) = y_0 + hy_0 + \frac{h^2}{2}y_0 + \frac{h^3}{6}y_0 + \frac{h^4}{24}y_0 + \cdots$$

An explicit s-stage RK method can match the Taylor expansion up to h^s but not higher, since it only uses s function evaluations per step. Thus, the order cannot exceed s.

14.7 Summary

Runge-Kutta methods provide a flexible and powerful class of one-step methods for ODEs. The Butcher tableau offers a compact notation for their coefficients. Explicit RK methods are easy to implement and widely used for non-stiff problems, while implicit variants are important for stiff systems.