Lecture Notes on Numerical Analysis

by

Ruennhwa Ferng

Department of Applied Mathematics

National Chiao Tung University

Hsin-Chu, TAIWAN

1995

Contents

List of Tables

List of Figures

Chapter 1

Mathematical Preliminaries

In this chapter, we review some of the most important topics from Calculus, Linear Algebra, and Matrix Analysis that will be required in the subsequent chapters throught this course. Throught this book, $\mathbb{R}$ will denote the set of all real numbers.

1.1 Review of Calculus

We start with the review of some definitions and theorems in Calculus including limit, continuity, derivative, and Riemann integral. We shall emphasize the various forms of Taylor's theorem.

1.1.1 Limit, Continuity, Derivative, and Integral

Unless otherwise stated, f will denote a real-valued function defined in a domain $\Omega \subseteq \mathbb{R}$, i.e., $f : \Omega \to \mathbb{R}$.

Definition 1.1 (Limit) *We say f has the limit L at c, written as $\lim_{x\to c} f(x) = L$, if*

$$\forall\, \varepsilon > 0,\ \exists\, \delta > 0,\ \ \textit{such that}\ \ |x - c| < \delta \Rightarrow |f(x) - L| < \varepsilon.$$

If there is no number L with this property, then we say the limit of f at c does not exist.

Definition 1.2 (Continuous) *The function f is said to be continuous at c if*

$$\lim_{x\to c} f(x) = f(c).$$

Definition 1.3 *Let $\{x_i\}_{i=1}^{\infty}$, where $x_i \in \mathbb{R}$ or $\mathbb{C}$, be an infinite sequence. The sequence converges to $x^\star$, write $\lim_{i\to\infty} x_i = x^\star$ or $x_i \to x^\star$ as $i \to \infty$, if for any $\varepsilon > 0$, there exists a positive integer $N(\varepsilon)$ such that $i > N$ implies $|x_i - x^\star| < \varepsilon$.*

Theorem 1.1 *Suppose $f : \Omega \to \mathbb{R}$, where $\Omega \subseteq \mathbb{R}$, is a function and $x^\star \in X$. Then the following are equivalent.*

1. *f is continuous at $x^\star$.*
2. *If $\{x_i\}_{i=1}^{\infty}$ is any sequence in X and $\lim_{i\to\infty} x_i = x^\star$, then $\lim_{i\to\infty} f(x_i) = f(x^\star)$.*

Definition 1.4 (Derivative) *The derivative of the function f at c is the limit, if it exists, defined by*

$$f'(c) = \lim_{x\to c} \frac{f(x) - f(c)}{x - c}.$$

If $f'(c)$ exists, then we say f is defferentiable at c.

Remarks 1.1 *Another form of derivative:*

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}.$$

Theorem 1.2 *If f is differentiable at c, then f must be continuous at c. But the converse is not true.*

Definition 1.5 *By convention, we use the following definitions and notations.*

- *$C(\Omega)$: the set of all functions that are continuous everywhere in Ω.*
- *$C^1(\Omega)$: the set of all functions f for which f' exists and is continuous everywhere in Ω.*
- *$C^n(\Omega)$: the set of all functions f for which $f^{(n)}$ exists and is continuous everywhere in Ω.*
- *$C^\infty(\Omega)$: the set of all functions each of whose derivatives is continuous.*

Property 1.1 $C^\infty(\Omega) \subset \ldots \subset C^2(\Omega) \subset C^1(\Omega) \subset C(\Omega)$

Remarks 1.2 *Similarly, we define $C^n[a,b]$ to be the set of all functions f defined on $[a,b]$ such that $f^{(n)}$ exists and is continuous everywhere in the interval (a,b).*

Theorem 1.3 (Rolle's Theorem) *If $f \in C[a,b]$, f' exists on (a,b), and $f(a) = f(b) = 0$, then there exists at least one $\xi \in (a,b)$ such that $f'(\xi) = 0$.*

Theorem 1.4 (Generalized Roll's Theorem) *Suppose $f \in C[a,b]$ is n times differentiable on (a,b). If $f(x) = 0$ for $x = x_0, x_1, \ldots, x_n$, where $x_i \in [a,b]$, are distinct, then there is a $\xi \in (a,b)$ such that $f^{(n)}(\xi) = 0$.*

Theorem 1.5 (Mean Value Theorem) *If $f \in C[a,b]$ and f' exists on (a,b), then for any $x, c \in [a,b]$, there exist a number ξ between x and c, such that*

$$f(x) = f(c) + f'(\xi)(x - c),$$

or, equivalently,

$$f'(\xi) = \frac{f(x) - f(c)}{x - c}.$$

Remarks 1.3 *Rolle's theorem is a special case of the Mean Value theorem.*

Theorem 1.6 (Intermediate Value Theorem) *If $f \in C[a,b]$ and η is some number between $f(a)$ and $f(b)$, then there exists $\xi \in (a,b)$ such that $f(\xi) = \eta$.*

Theorem 1.7 (Extreme Value Theorem) *If $f \in C[a,b]$, then there exist ξ and η in $[a,b]$ such that $f(\xi) \le f(x) \le f(\eta)$ for all $x \in [a,b]$. If, in addition, f is differentiable on (a,b), then ξ and η occur either at the end-points of $[a,b]$ or where f' is zero.*

Definition 1.6 (Riemann Integral) *Suppose f is a function defined on $[a,b]$. The Riemann integral of f on $[a,b]$ is the limit, if it exists,*

$$\int_a^b f(x)dx = \lim_{\max \triangle x_i \to 0} \sum_{i=1}^n f(z_i) \triangle x_i,$$

where $\triangle x_i = x_i - x_{i-1}$, $z_i \in [x_{i-1}, x_i]$ is arbitrary, and $a = x_0 \le x_1 \le \cdots \le x_n = b$ is a partition of $[a,b]$.

Remarks 1.4 *In the case of equally spaced partition and $z_i = x_i$, one has*

$$\int_a^b f(x)dx = \lim_{n\to\infty} \frac{b-a}{n} \sum_{i=1}^n f(x_i),$$

where $x_i = a + i(b-1)/n$.

Theorem 1.8 (Mean Value Theorem for Integral) *If f is continuous on $[a,b]$ and g is integrable and does not change sign on $[a,b]$, then there exists $\xi \in (a,b)$, such that*

$$\int_a^b f(x)g(x)dx = f(\xi) \int_a^b g(x)dx.$$

In the special case when $g(x) \equiv 1$ on $[a,b]$, then one has

$$\int_a^b f(x)dx = f(\xi)(b-a).$$

1.1.2 Taylor's Theorem

Theorem 1.9 (Taylor's Theorem with Lagrange Formula) *Suppose $f \in C^n[a,b]$ and $f^{(n+1)}$ exists on (a,b). Then for any x and c in $[a,b]$, there exists a number ξ, depending on x, between x and c such that*

$$f(x) = P_n(x) + E_n(x), \tag{1.1}$$

where

$$P_n(x) = \sum_{k=0}^n \frac{1}{k!} f^{(k)}(c)(x-c)^k \tag{1.2}$$

is called the n-th Taylor polynomial for f centered at c, and

$$E_n(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi)(x-c)^{n+1} \tag{1.3}$$

is called the remainder term, error term, or truncation error associated with $P_n(x)$. The infinite series by taking the limit of $P_n(x)$ as $n \to \infty$ is called the Taylor series for $f(x)$.

Remarks 1.5 *The mean-value theorem is a special case of Tayler's theorem by taking $n = 0$.*

Theorem 1.10 (Taylor's Theorem with Integral Form) *If $f \in C^{n+1}[a,b]$, then for any x and c in $[a,b]$,*

$$f(x) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(c)(x-c)^k + R_n(x), \tag{1.4}$$

where

$$R_n(x) = \frac{1}{n!} \int_c^x f^{(n+1)}(t)(x-t)^n dt. \tag{1.5}$$

Theorem 1.11 (Other forms of Taylor's Theorem) *If $f \in C^{n+1}[a,b]$, then for any $x \in (a,b)$ and $h \geq 0$ in a neighborhood of x, there exists ξ such that*

$$f(x+h) = \sum_{k=0}^{n} \frac{1}{k!} f^{(k)}(x)h^k + \frac{1}{(n+1)!} f^{(n+1)}(\xi)h^{n+1}, \tag{1.6}$$

if $x < \xi < x+h$, and

$$f(x-h) = \sum_{k=0}^{n} (-1)^k \frac{1}{k!} f^{(k)}(x)h^k + \frac{(-1)^{n+1}}{(n+1)!} f^{(n+1)}(\xi)h^{n+1}, \tag{1.7}$$

if $x - h < \xi < x$.

Theorem 1.12 (Taylor's Theorem in Two Variables) *Suppose that $f(x,y)$ and all its partial derivatives of order less than or equal to $n+1$ are continuous on $D = \{(x,y) | a \leq x \leq b, c \leq y \leq d\}$, and let $(x_0, y_0) \in D$. For every $(x,y) \in D$, there exists ξ between x and x_0 and μ between y and y_0 with*

$$f(x,y) = P_n(x,y) + R_n(x,y), \tag{1.8}$$

where

$$\begin{aligned} P_n(x,y) \;=\; & f(x_0,y_0) + \left[(x-x_0)\frac{\partial f}{\partial x}(x_0,y_0) + (y-y_0)\frac{\partial f}{\partial y}(x_0,y_0)\right] \\ & + \left[\frac{(x-x_0)^2}{2}\frac{\partial^2 f}{\partial x^2}(x_0,y_0) + (x-x_0)(y-y_0)\frac{\partial^2 f}{\partial x \partial y}(x_0,y_0) \right. \\ & \left. + \frac{(y-y_0)^2}{2}\frac{\partial^2 f}{\partial y^2}(x_0,y_0)\right] + \cdots\cdots \\ & + \left[\frac{1}{n!}\sum_{j=0}^{n}\binom{n}{j}(x-x_0)^{n-j}(y-y_0)^j \frac{\partial^n f}{\partial x^{n-j}\partial y^j}(x_0,y_0)\right] \end{aligned} \tag{1.9}$$

and

$$R_n(x,y) = \frac{1}{(n+1)!}\sum_{j=0}^{n+1}\binom{n+1}{j}(x-x_0)^{n+1-j}(y-y_0)^j\frac{\partial^{n+1}f}{\partial x^{n+1-j}\partial y^j}(\xi,\mu). \tag{1.10}$$

The function $P_n(x,y)$ is called the n-th Taylor polynomial in two variables for thefunction $f(x,y)$, and $R_n(x,y)$ is the remainder term associated with $P_n(x,y)$.

Example 1.1 *Consider the function* $f(x) = \ln x$ *in the interval* $[1,2]$. *Since* $f'(x) = x^{-1}$, $f''(x) = -x^{-2}$, $f'''(x) = 2x^{-3}, \cdots$ *and so on. In general,* $f^{(k)}(x) = (-1)^{k-1}(k-1)!x^{-k}$. *Hence, with* $c = 1$, *Taylor's theorem 1.9 gives,*

$$\ln x = \sum_{k=1}^{n}\frac{(-1)^{k-1}}{k}(x-1)^k + \frac{(-1)^n}{n+1}\xi^{-(n+1)}(x-1)^{n+1},$$

where $1 \le x \le 2$ *and* $1 < \xi < x$. *The error term is*

$$|E_n| = \frac{1}{n+1}\xi^{-(n+1)}(x-1)^{n+1} \le \frac{1}{n+1}(x-1)^{n+1},$$

since $\xi > 1$ *and* $\xi^{-(n+1)} < 1$. *To estimate* $\ln 2$ *with an error less than* 10^{-8}, *it would require*

$$|E_n| \le \frac{1}{n+1} \le 10^{-8}.$$

This means $n \ge 10^8 - 1$. *Similarly, to estimate* $\ln 1.5$ *with the same precision, it requires*

$$|E_n| \le \frac{1}{n+1}\left(\frac{1}{2}\right)^{n+1} \le 10^{-8}.$$

It is sufficient to attain the accuracy if $n \ge 26$. ■

Example 1.2 *Consider the function* $f(x) = \cos x$ *defined on* $\mathbb{R}$. *Since* $f \in C^\infty(\mathbb{R})$, *Taylor's theorem can be applied for any* $n > 0$. *With* $n = 2$ *and* $c = 0$, *we have*

$$f(x) = \cos x = 1 - \frac{1}{2}x^2 + \frac{1}{6}x^3\sin\xi,$$

where $0 < \xi < x$. *To estimate* $\cos(0.01)$ *using the Taylor's theorem, one has*

$$\cos(0.01) = 1 - \frac{1}{2}(0.01)^2 + \frac{1}{6}(0.01)^3\sin\xi = 0.99995 + \frac{1}{6}\times 10^{-6}\sin\xi,$$

where $0 < \xi < 0.01$. *Since* $|\sin\xi| < 1$, *the error is*

$$|\cos(0.01) - 0.99995| \le \frac{1}{6}\times 10^{-6} \approx 0.167\times 10^{-6}.$$

This means that the approximation 0.99995 *matches at least the first five digits of* $\cos(0.01)$. *On the other hand, with $n = 3$, we also have*

$$f(x) = \cos x = 1 - \frac{1}{2}x^2 + \frac{1}{24}x^4 \cos \xi.$$

The apporximation of $\cos(0.01)$ *with this formula remains the same, still* 0.99995*, but we now have much better accuracy assurance since the error is now*

$$|\frac{1}{24}x^4 \cos \xi| \leq \frac{1}{24}(0.01)^4 \approx 0.42 \times 10^{-9}.$$

This means that the approximation 0.99995 *matches at least the first eight digits when compared to the 11-digit accuracy* $\cos(0.01) = 0.99995000042$. ■

These examples illustrate the two objectives of numerical analysis. The first is to find approximation, which both Taylor polynomials provide. The second objective is to determine the accuracy of the approximation. In the previous example, the later analysis is much more informative, even though both gave the same approximation.

1.2 Review of Linear Algebra and Matrix Analysis

1.2.1 Matrices and Vectors

We let $\mathbb{R}^n$ denote the set of all real n-vectors (column vectors):

$$x \in \mathbb{R}^n \quad \Longleftrightarrow \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \qquad x_i \in \mathbb{R}, \quad \forall\, i = 1, 2, \ldots, n,$$

in which x_i is referred as the i-th component of x. By convection, $\mathbb{R}^{n\times 1} = \mathbb{R}^n$ denotes the set of column vectors and $\mathbb{R}^{1\times n}$ the row vectors, $x \in \mathbb{R}^{1\times n}$, $x = [x_1, x_2, \ldots, x_n]$, $x_i \in \mathbb{R}$.

$\mathbb{R}^{m\times n}$ will denote the set of all $m \times n$ matrices:

$$A \in \mathbb{R}^{m\times n} \quad \Longleftrightarrow \quad A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix},$$

in which $a_{ij} \in \mathbb{R}$ is called the (i, j) component of A. Frequently, we use the notation

$$A = [a_1, a_2, \ldots, a_n],$$

in which

$$a_j = \begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix} \in \mathbb{R}^m$$

denotes the j-th column of A. Analogously, we use $\mathbb{C}^n$ and $\mathbb{C}^{m\times n}$ to denote the set of complex n-vectors and complex $m \times n$ matrices, respectively.

The transpose of matrix A is $A^T \in \mathbb{R}^{n\times m}$ and

$$A^T = \begin{bmatrix} a_{11} & a_{21} & \cdots & a_{m1} \\ a_{12} & a_{22} & \cdots & a_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1n} & a_{2n} & \cdots & a_{mn} \end{bmatrix}.$$

For complex matrix $A \in \mathbb{C}^{m\times n}$, the Hermitian of A is

$$A^H = \begin{bmatrix} \bar{a}_{11} & \bar{a}_{21} & \cdots & \bar{a}_{m1} \\ \bar{a}_{12} & \bar{a}_{22} & \cdots & \bar{a}_{m2} \\ \vdots & \vdots & \ddots & \vdots \\ \bar{a}_{1n} & \bar{a}_{2n} & \cdots & \bar{a}_{mn} \end{bmatrix},$$

where $a_{ij} \in \mathbb{C}$ and$\bar{a}_{ij}$ denotes the complex conjugate of a_{ij}.

Some basic matrix-vector operations include

- **dot-product:** for x, $y \in \mathbb{R}^n$,

$$x^T y = x_1 y_1 + \cdots + x_n y_n = \sum_{i=1}^{n} x_i y_i \ \in \mathbb{R}.$$

- **outer product:** for $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$,

$$xy^T = [x_i y_j] = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix} \in \mathbb{R}^{m\times n}.$$

- **matrix-vector multiplication:** for $A = [a_1, a_2, \ldots, a_n] \in \mathbb{R}^{m\times n}$ and $x \in \mathbb{R}^n$,

$$y = Ax = x_1 a_1 + x_2 a_2 + \cdots + x_n a_n, \quad \text{or} \quad y_i = \sum_{k=1}^{n} a_{ik} x_k, \quad i = 1, 2, \ldots, n.$$

- **matrix-matrix multiplication:** for $A \in \mathbb{R}^{m\times p}$ and $B \in \mathbb{R}^{p\times n}$,

$$C = AB \in \mathbb{R}^{m\times n} \quad \text{with} \quad c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}.$$

- **saxpy:**

$$z = \alpha x + y, \qquad \text{or} \qquad z_i = \alpha x_i + y_i$$

  with x, y, $z \in \mathbb{R}^n$, $\alpha \in \mathbb{R}$.

- **outer-product update:**

$$A \leftarrow A + xy^T \qquad \text{or} \qquad a_{ij} \leftarrow a_{ij} + x_i y_j$$

  with $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$.

- **Gaxpy:**

$$z = Ax + y \qquad \text{or} \qquad z_i = \sum_{k=1}^{n} a_{ik}x_k + y_i$$

  with $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, $y \in \mathbb{R}^m$, $z \in \mathbb{R}^m$.

The identity matrix of order n, $I_n = [\delta_{ij}]$, is a special matrix with entries

$$\delta_{ij} = \begin{cases} 1, & \text{if } i = j, \\ 0 & \text{if } i \neq j. \end{cases}$$

The columns of I_n,

$$e_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad \cdots, \qquad e_n = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix},$$

are called the standard unit vectors. When the size of I_n is clear from the content, this matrix is generally written simply as I.

A square matrix $A \in \mathbb{R}^{n\times n}$ is said to be nonsingular (or invertible) if there exists $B \in \mathbb{R}^{n\times n}$ such that $BA = AB = I_n$, the $n \times n$ identity matrix. The matrix B is called the inverse of A and is denoted as A^{-1}.

When A and B are both n-by-n non-singular matrices, we have

$$(AB)^{-1} = B^{-1}A^{-1} \quad \text{and} \quad (A^{-1})^T = (A^T)^{-1} = A^{-T}.$$

Suppose $A \in \mathbb{R}^{n\times n}$ is nonsingular and A^{-1} is known. The the inverse $(A+xy^T)^{-1}$, where $x, y \in \mathbb{R}^n$, can be computed by the so-called Sherman-Morrison-Woodbury formula

$$(A + xy^T)^{-1} = A^{-1} - \frac{A^{-1}xy^TA^{-1}}{1 + y^TA^{-1}x}$$

if $1 + y^TA^{-1}x \neq 0$. The extension to rank-k formula for $U, V \in \mathbb{R}^{n\times k}$ is

$$(A + UV^T)^{-1} = A^{-1} - A^{-1}U(I + V^TA^{-1}U)^{-1}V^TA^{-1}$$

if $I + V^TA^{-1}U$ is nonsingular.

We list in the following some matrices which has either special structure or property.

Definition 1.7 *Let $A = [a_{ij}]$ be a square or rectangular matrix, A is called*

- *diagonal if $a_{ij} = 0, \quad \forall\, i \neq j$;*
- *tridiagonal if $a_{ij} = 0$ if $|i - j| > 1$;*
- *upper bidiagonal if $a_{ij} = 0, \quad \forall\, i > j$ or $j > i + 1$;*
- *upper triangular if $a_{ij} = 0, \quad \forall\, i > j$;*
- *strictly upper triangular if $a_{ij} = 0, \quad \forall\, i \geq j$;*
- *upper Hessenberg matrix if $a_{ij} = 0, \quad \forall\, i > j + 1$;*
- *symmetric if $A^T = A$;*
- *skew symmetric if $A^T = -A$;*
- *if positive if $a_{ij} > 0, \quad \forall\, i,\ j$;*
- *positive definite if $x^T A x > 0, \quad \forall\, x \neq 0, \ x \in \mathbb{R}^n$;*
- *non-negative definite or positive semi-definite if $x^T A x \geq 0, \quad \forall\, x \in \mathbb{R}^n$;*
- *indefinite if there exist $x, y \in \mathbb{R}^n$ such that $(x^T A x)(y^T A y) < 0$;*
- *idempotent if $A^2 = A$.*
- *nilpotent if $A^k = 0$ for some positive integer k;*
- *diagonal dominant if $|a_{ii}| > \sum_{j \neq i} |a_{ij}|, \ \forall\, i$;*
- *orthogonal if $A^T A = I$ (that is, $A^T = A^{-1}$) when $A \in \mathbb{R}^{n\times n}$ and unitary if $A^H A = I$ when $A \in \mathbb{C}^{n\times n}$;*
- *normal if $A^T A = A A^T$ when $A \in \mathbb{R}^{n\times n}$ and $A^H A = A A^H$ when $A \in \mathbb{C}^{n\times n}$.*

1.2.2 Vector Space, Range Space, and Null Space

Definition 1.8 (Linearly Independent) *We say vectors $x_1, x_2, \ldots, x_m \in \mathbb{R}^n$ are linearly independent if $\alpha_1, \alpha_2, \ldots, \alpha_m \in \mathbb{R}$*

$$\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_m x_m = 0 \quad \Longrightarrow \quad \alpha_1 = \alpha_2 = \ldots = \alpha_m = 0.$$

If there exists any $\alpha_k \neq 0$ but $\sum_{i=1}^{m} \alpha_i x_i = 0$, then $x_1, x_2, \ldots, x_m$ are said to be linearly dependent.

A subspace spanned by $x_1, x_2, \dots, x_m$ is denoted and defined as

$$\text{span}\{x_1, x_2, \dots, x_m\} = \left\{ y \in \mathbb{R}^n \mid y = \sum_{i=1}^{m} \alpha_i x_i \text{for some} \alpha_i \in \mathbb{R} \right\}.$$

Suppose $S \subseteq \mathbb{R}^n$ is a subspace and $B = \{b_1, \dots, b_m\} \subseteq S$ is a subset. If $b_1, \dots, b_m$ are linearly independent and $\text{span}(B) = S$, then B is called a basis for S and $\dim(S) = m$.

Definition 1.9 (Range and Null Space) *Let $A = [a_1, \dots, a_n] \in \mathbb{R}^{m \times n}$, where $a_i \in \mathbb{R}^m$ are columns of A. The range of A is denoted and defined as*

$$\mathcal{R}(A) = \{y \in \mathbb{R}^m \mid y = Ax, \text{ for some } x \in \mathbb{R}^n\} = \text{span}\{a_1, a_2, \dots, a_n\}, \tag{1.11}$$

and the null space of A is denoted and defined as

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n | Ax = 0\}. \tag{1.12}$$

Definition 1.10 (Rank and Nullity) *The rank of a matrix A,* $\text{rank}(A)$*, is the dimension of $\mathcal{R}(A)$ and the nullity of A,* $\text{nullity}(A)$*, is the dimension of $\mathcal{N}(A)$.*

Note 1.1 $\text{rank}(A) = \text{rank}(A^T)$ *is the maximal number of linearly independent columns (or rows) in A.*

Note 1.2 *Suppose $A \in \mathbb{R}^{m \times n}$ and $m \geq n$, then*

$$\text{rank}(A) + \text{nullity}(A) = n.$$

Note 1.3 *Suppose $A \in \mathbb{R}^{n \times n}$. Then*

$$\begin{aligned}
& A \text{ is non-singular (invertible, } A^{-1} \text{ exists)} \\
\iff & Ax = 0 \Rightarrow x = 0 \\
\iff & Ax = b \text{ has a unique solution} \\
\iff & \text{rank}(A) = n \\
\iff & \det(A) \neq 0
\end{aligned}$$

1.2.3 Orthogonality

Definition 1.11 (Orthogonal and Orthonormal) *A set of vectors $x_1, \dots, x_m \in \mathbb{R}^n$ are said to be orthogonal if*

$$x_i^T x_j = 0, \ \forall\, i \neq j,$$

and orthonormal if

$$x_i^T x_j = \delta_{ij} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

Definition 1.12 *Two subspaces* S_1 *and* S_2 *of* $\mathbb{R}^n$ *are said to be orthogonal if* $x^T y = 0$ *for all* $x \in S_1$ *and* $y \in S_2$.

Definition 1.13 (Orthogonal Complement) *Suppose* $S \subseteq \mathbb{R}^n$ *is a subspace. The orthogonal complement of* S *is defined as*

$$S^{\perp} = \left\{ y \in \mathbb{R}^n \mid y^T x = 0, \ \forall \, x \in S \right\}$$

Definition 1.14 (Direct Sum) *Suppose* U *and* V *are subspaces of a vector space* S. *If each* $s \in S$ *can be written uniquely as a sum* $u + v$, *where* $u \in U$ *and* $v \in V$, *then we say that* S *is a direct sum of* U *and* V *and we write* $S = U \oplus V$.

Theorem 1.13 *Suppose* $A \in \mathbb{R}^{m \times n}$. *Then*

1. $\mathcal{R}(A)^{\perp} = \mathcal{N}(A^T)$;
2. $\mathcal{N}(A) = \mathcal{R}(A^T)^{\perp}$;
3. $\mathbb{R}^n = \mathcal{R}(A^T) \oplus \mathcal{N}(A)$.

Proof: For any $y \in \mathcal{R}(A)^{\perp}$, $y^T x = 0, \forall \, x \in \mathcal{R}(A)$. Since x is in $\mathcal{R}(A)$, there exists $z \in \mathbb{R}^n$ such that $x = Az$. Then

$$0 = y^T x = y^T (Az) = (A^T y)^T z.$$

Since $z = A^T y$ is arbitrary, it must be $A^T y = 0$. That is, $y \in \mathcal{N}(A^T)$. Hence

$$\mathcal{R}(A)^{\perp} \subseteq \mathcal{N}(A^T).$$

Conversely, suppose $y \in \mathcal{N}(A^T)$. Then $A^T y = 0$ and hence $(A^T y)^T x = y^T (Ax) = 0$ for any $x \in \mathbb{R}^n$. This means $y \in \mathcal{R}(A)^{\perp}$. Therefore

$$\mathcal{N}(A^T) \subseteq \mathcal{R}(A)^{\perp}.$$

■

1.3 Norms

1.3.1 Vector Norm Definition and Properties

Definition 1.15 *A vector norm is a function* $\| \cdot \| : \mathbb{R}^n \to \mathbb{R}$ *satisfying the following conditions for all* $x, y \in \mathbb{R}^n$ *and* $\alpha \in \mathbb{R}$.

1. $\|x\| \geq 0 \quad (\|x\| = 0 \Leftrightarrow x = 0)$;
2. $\|x + y\| \leq \|x\| + \|y\|$;
3. $\|\alpha x\| = |\alpha| \|x\|$.

Definition 1.16 *For $x \in \mathbb{R}^n$, some of the most frequently used vector norms are*

- **1-norm:**

$$\|x\|_1 = \sum_{i=1}^{n} |x_i| = |x_1| + |x_2| + \ldots + |x_n|. \tag{1.13}$$

- **2-norm:**

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2} = \left(|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2\right)^{1/2} = \sqrt{x^T x}. \tag{1.14}$$

- **∞-norm:**

$$\|x\|_\infty = \max_{1 \le i \le n} |x_i|. \tag{1.15}$$

- **p-norm:**

$$\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{1/p} = \left(|x_1|^p + |x_2|^p + \ldots + |x_n|^p\right)^{1/p}. \tag{1.16}$$

Definition 1.17 (unit vector) *$x \in \mathbb{R}^n$ is called a unit vector if $\|x\| = 1$ with respect to some vector norm.*

Property 1.2 *For any $x, y \in \mathbb{R}^n$, the following two inequalities hold.*

- *Hölder inequality:*

$$|x^T y| \le \|x\|_p \|y\|_q, \quad \text{where} \quad \frac{1}{p} + \frac{1}{q} = 1.$$

- *Cauchy-Schwartz inequality:*

$$|x^T y| \le \|x\|_2 \|y\|_2.$$

Definition 1.18 *Two vector norms $\|\cdot\|_\alpha$ and $\|\cdot\|_\beta$ are equivalent if there exist constants $c_1, c_2 \in \mathbb{R}$ such that*

$$c_1 \|x\|_\alpha \le \|x\|_\beta \le c_2 \|x\|_\alpha$$

for any $x \in \mathbb{R}^n$.

In fact, all vector norms on $\mathbb{R}^n$ are equivalent, and it is easy to show the following equivalence properties for vector norms.

Property 1.3 *For all $x \in \mathbb{R}^n$,*

$$\|x\|_2 \leq \|x\|_1 \leq \sqrt{n}\|x\|_2. \tag{1.17}$$

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n}\|x\|_\infty. \tag{1.18}$$

$$\|x\|_\infty \leq \|x\|_1 \leq \quad n\|x\|_\infty. \tag{1.19}$$

Definition 1.19 (absolute error and relative error) *Suppose $x \in \mathbb{R}^n$ is the exact solution of some problem and $\tilde{x}$ is an approximation to x. We define*

$$\textit{absolute error} = \|x - \tilde{x}\| \tag{1.20}$$

and

$$\textit{relative error} = \frac{\|x - \tilde{x}\|}{\|x\|}, \quad \textit{if } x \neq 0. \tag{1.21}$$

Definition 1.20 (convergence) *Suppose $\{x_k\}$ is a sequence of vectors in $\mathbb{R}^n$. We say x_k converges to $x^\star$*

$$x_k \longrightarrow x^\star \quad \textit{iff} \quad \lim_{k\to\infty} \|x_k - x^\star\| = 0,$$

for some vector norm.

1.3.2 Matrix Norm Definition and Properties

Definition 1.21 *A matrix norm is a function $\|\cdot\| : \mathbb{R}^{m\times n} \to \mathbb{R}$ satisfying the following conditions for all $A, B \in \mathbb{R}^{m\times n}$ and $\alpha \in \mathbb{R}$.*

1. $\|A\| \geq 0 \quad (\|A\| = 0 \Leftrightarrow A = 0)$;
2. $\|A + B\| \leq \|A\| + \|B\|$;
3. $\|\alpha A\| = |\alpha|\|A\|$.

Definition 1.22 *Some of the most frequently used matrix norms are*

- **Frobenius norm:**

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}. \tag{1.22}$$

- **2-norm:**

$$\|A\|_2 = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|x\|_2=1} \|Ax\|_2. \tag{1.23}$$

- **1-norm:**

$$\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|. \tag{1.24}$$

- **∞-norm:**

$$\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|. \tag{1.25}$$

- **p-norm:**

$$\|A\|_p = \max_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p. \tag{1.26}$$

Remarks 1.6 *Frobenius norm, 1-norm, and ∞-norm are easy to compute, but not the 2-norm or general p-norm.*

Theorem 1.14 *Suppose $A \in \mathbb{R}^{m \times n}$. Then there exists $z \in \mathbb{R}^n$, $\|z\|_2 = 1$, such that $A^T A z = \mu^2 z$, where $\mu = \|A\|_2$.*

Proof: Let $z \in \mathbb{R}^n$, $\|z\|_2 = 1$, be a unit vector that satisfies $\|A\|_2 = \|Az\|_2 = \max\limits_{\|x\|_2=1} \|Ax\|_2$. Define

$$g(x) = \frac{1}{2}\frac{x^T A^T A x}{x^T x} = \frac{1}{2}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2}\left(\frac{\|Ax\|_2}{\|x\|_2}\right)^2,$$

for $x \in \mathbb{R}^n$. Then z is a maximizer of $g(x)$ which impliese $\bigtriangledown g(z) = 0$. Since

$$\bigtriangledown g(x) = \frac{(x^T x)(A^T A x) - (x^T A^T A x)(x)}{(x^T x)^2}.$$

Hence

$$\begin{aligned} \bigtriangledown g(z) = 0 \quad &\Rightarrow \quad (z^T z)(A^T A z) - (z^T A^T A z) z = 0 \\ &\Rightarrow \quad \|z\|_2^2 (A^T A z) - \|Az\|_2^2 z = 0 \\ &\Rightarrow \quad A^T A z = \|A\|_2^2 z = \mu^2 z. \end{aligned}$$

■

Remarks 1.7 *$\|A\|_2$ is the square root of the largest eigenvalue of $A^T A$. When A is symmetric, $\|A\|_2$ is the absolute value of the largest eigenvalue in magnitude.*

Definition 1.23 (submultiplicative property) *A matrix norm $\|\cdot\|$ is said to have the submultiplicative property if for any matrices $A \in \mathbb{R}^{m \times p}$ and $B \in \mathbb{R}^{p \times n}$ such that*

$$\|AB\| \le \|A\| \|B\|.$$

Not all matrix norms satisfy the submultiplicative property, e.g., define $\|A\|_\triangle = \max_{i,j} |a_{ij}|$, and let $A = B = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$. But we conly consider matrix norms that satisfy the submultiplicative property.

Property 1.4 *p-norm satisfies the submultiplicative property.*

$$\|Ax\|_p \leq \|A\|_p \|x\|_p \tag{1.27}$$
$$\|AB\|_p \leq \|A\|_p \|B\|_p \tag{1.28}$$

Property 1.5 *All matrix norms satisfying the submultiplicative property are equivalent.*

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n}\|A\|_2. \tag{1.29}$$
$$\frac{1}{\sqrt{n}}\|A\|_\infty \leq \|A\|_2 \leq \sqrt{m}\|A\|_\infty. \tag{1.30}$$
$$\frac{1}{\sqrt{m}}\|A\|_1 \leq \|A\|_2 \leq \sqrt{n}\|A\|_1. \tag{1.31}$$
$$\max_{i,j} |a_{ij}| \leq \|A\|_2 \leq \sqrt{mn} \max_{i,j} |a_{ij}|. \tag{1.32}$$

Property 1.6 *Vector* 2*-norm is invariant under orthogonal transformation, that is, if Q is an n-by-n orthogonal matrix, then*

$$\|Qx\|_2 = \|x\|_2 \qquad \forall\, x \in \mathbb{R}^n. \tag{1.33}$$

Proof: Since

$$\|Qx\|_2^2 = (Qx)^T(Qx) = x^T Q^T Q x = x^T x = \|x\|_2^2.$$

■

Property 1.7 *Matrix* 2*-norm and Frobenius norm are invariant under orthogonal transformation, that is, if Q is an n-by-n orthogonal matrix, then*

$$\|QA\|_2 = \|A\|_2 \tag{1.34}$$
$$\|QA\|_F = \|A\|_F \tag{1.35}$$

for all $A \in \mathbb{R}^{n\times m}$.

Proof: Since

$$\|QA\|_2 = \max_{\|x\|_2=1} \|QAx\|_2 = \max_{\|x\|_2=1} \|Ax\|_2 = \|A\|_2.$$

■

Property 1.8 *If $A \in \mathbb{R}^{m\times n}$, then $\|A\|_2 \leq \sqrt{\|A\|_1 \|A\|_\infty}$.*

Proof: From Theorem 1.14, there exists a vector $z \in \mathbb{R}^n$, $z \neq 0$, such that $\|A\|_2^2 z = A^T A z$. This implies

$$\|A\|_2^2 \|z\|_1 = \|A^T A z\|_1 \leq \|A^T\|_1 \|A\|_1 \|z\|_1 = \|A\|_\infty \|A\|_1 \|z\|_1.$$

Hence

$$\|A\|_2^2 \leq \|A\|_\infty \|A\|_1.$$

■

Theorem 1.15 *Suppose that $A \in \mathbb{R}^{n\times n}$ and $\|\cdot\|$ is a submultiplicative matrix norm. If $\|A\| < 1$, then $I - A$ is nonsingular and*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k \tag{1.36}$$

with

$$\|(I - A)^{-1}\| \leq \frac{1}{1 - \|A\|}. \tag{1.37}$$

Proof: Suppose $I - A$ is singular. Then there exists an $x \in \mathbb{R}^n$, $x \neq 0$ such that $(I - A)x = 0$. This implies

$$Ax = x \quad \Longrightarrow \quad \|Ax\| = \|x\| \quad \Longrightarrow \quad \frac{\|Ax\|}{\|x\|} = 1 \quad \Longrightarrow \quad \|A\| = \max_{y\neq 0} \frac{\|Ay\|}{\|y\|} \geq 1.$$

This contradicts to the assumption $\|A\| < 1$. Hence $I - A$ must be nonsingular. Moreover

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

Since $\|A\| < 1$ and $\|\cdot\|$ is a submultiplicative matrix norm, $\|A^N\| \leq \|A\|^N$ and $\lim_{N\to\infty} A^N = 0$. Therefore

$$(I - A)\left(\lim_{N\to\infty} \sum_{k=0}^{N} A^k\right) = I.$$

It follows that

$$(I - A)^{-1} = \lim_{N\to\infty} \sum_{k=0}^{N} A^k = \sum_{k=0}^{\infty} A^k$$

and

$$\|(I - A)^{-1}\| = \left\| \sum_{k=0}^{\infty} A^k \right\| \leq \sum_{k=0}^{\infty} \|A^k\| \leq \sum_{k=0}^{\infty} \|A\|^k = \frac{1}{1 - \|A\|}.$$

■

Theorem 1.16 *If A is nonsingular and $\|A^{-1}E\| < 1$, then $A+E$ is nonsingular and*

$$\|(A+E)^{-1} - A^{-1}\| \leq \frac{\|E\|\|A^{-1}\|^2}{1-\|A^{-1}E\|}. \tag{1.38}$$

Proof: Since A is nonsingular, $A+E = A(I+A^{-1}E)$. Since $\|A^{-1}E\| < 1$ it follows from Theorem 1.15 that $I+A^{-1}E$ is nonsingular and $\|(I+A^{-1}E)^{-1}\| \leq \frac{1}{1-\|A^{-1}E\|}$. Hence $A+E$ is nonsingular, $(A+E)^{-1} = (I+A^{-1}E)^{-1}A^{-1}$, and

$$\|(A+E)^{-1}\| = \|(I+A^{-1}E)^{-1}A^{-1}\| \leq \|(I+A^{-1}E)^{-1}\|\|A^{-1}\| \leq \frac{\|A^{-1}\|}{1-\|A^{-1}E\|}.$$

Now

$$(A+E)^{-1} - A^{-1} = \left(I - A^{-1}(A+E)\right)(A+E)^{-1} = -A^{-1}E(A+E)^{-1},$$

and so by taking norms we find

$$\|(A+E)^{-1} - A^{-1}\| \leq \|A^{-1}\|\|E\|\|(A+E)^{-1}\| \leq \frac{\|E\|\|A^{-1}\|^2}{1-\|A^{-1}E\|}.$$

■

1.4 SVD: The Singular Value Decomposition

The singular value decomposition (SVD) is a matrix factorization whose computation is a step in many algorithms. Many problems of linear algebra can be better understood by considering the SVD.

Theorem 1.17 (Existence of SVD) *If $A \in \mathbb{R}^{m\times n}$, then there exists orthogonal matrices*

$$U = [u_1, u_2, \dots, u_m] \in \mathbb{R}^{m\times m} \quad \textit{and} \quad V = [v_1.v_2, \dots, v_n] \in \mathbb{R}^{n\times n}$$

such that

$$A = U\Sigma V^T, \tag{1.39}$$

where

$$\Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \dots, \sigma_p), \qquad p = \min(m,n),$$

with

$$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_p \geq 0.$$

Proof: Let $\sigma_1 = \|A\|_2 \neq 0$. By the compactness arguement there exists $v_1 \in \mathbb{R}^n$, $\|v_1\|_2 = 1$ such that $\|Av_1\|_2 = \sigma_1$. Let $u_1 = \frac{1}{\sigma_1} Av_1$. Then $Av_1 = \sigma_1 u_1$, $\|u_1\|_2 = 1$, and $u_1^T Av_1 = \sigma_1$.

Let $V_1 = [v_1, \widehat{V}_1] \in \mathbb{R}^{n\times n}$ and $U_1 = [u_1, \widehat{U}_1] \in \mathbb{R}^{m\times m}$ be the orthogonal matrices using the extension of v_1 and u_1 to form orthonormal bases for $\mathbb{R}^n$ and $\mathbb{R}^m$, respectively. Note that this can always be done by Gram-Schmidt process. Then

$$\widehat{U}_1^T Av_1 = \widehat{U}_1^T(\sigma_1 u_1) = \sigma_1(\widehat{U}_1^T u_1) = 0$$

and

$$U_1^T AV_1 = \begin{bmatrix} u_1^T \\ \widehat{U}_1^T \end{bmatrix} A \left[v_1, \widehat{V}_1\right] = \begin{bmatrix} u_1^T Av_1 & u_1^T A\widehat{V}_1 \\ \widehat{U}_1^T Av_1 & \widehat{U}_1^T A\widehat{V}_1 \end{bmatrix} \equiv \begin{bmatrix} \sigma_1 & w^T \\ 0 & B \end{bmatrix} \equiv A_1.$$

Next we show that $w^T \equiv u_1^T A\widehat{V}_1 = 0$. Since

$$A_1 \begin{bmatrix} \sigma \\ w \end{bmatrix} = \begin{bmatrix} \sigma & w^T \\ 0 & B \end{bmatrix} \begin{bmatrix} \sigma \\ w \end{bmatrix} = \begin{bmatrix} \sigma^2 + w^T w \\ Bw \end{bmatrix},$$

hence

$$\begin{aligned} \|A_1\|_2 = \max \frac{\|A_1 x\|_2}{\|x\|_2} \quad &\geq \quad \frac{\left\| A_1 \begin{bmatrix} \sigma \\ w \end{bmatrix} \right\|_2}{\left\| \begin{bmatrix} \sigma \\ w \end{bmatrix} \right\|_2} = \frac{\left\| \begin{bmatrix} \sigma^2 + w^T w \\ Bw \end{bmatrix} \right\|_2}{\left\| \begin{bmatrix} \sigma \\ w \end{bmatrix} \right\|_2} \\ &= \quad \frac{((\sigma_1^2 + w^T w)^2 + w^T B^T Bw)^{1/2}}{(\sigma_1^2 + w^T w)^{1/2}} \\ &\geq \quad \frac{\left(\sigma_1^2 + w^T w\right)}{\left(\sigma_1^2 + w^T w\right)^{1/2}} = \left(\sigma_1^2 + w^T w\right)^{1/2} \geq \sigma_1. \end{aligned}$$

But

$$\|A_1\|_2 = \|U_1^T AV_1\|_2 = \|A\|_2 = \sigma_1.$$

Hence

$$w^T w = 0 \quad \Rightarrow \quad w = 0.$$

Therefore

$$U_1^T AV_1 = \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix} \quad \Rightarrow \quad A = U_1 \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix} V_1^T.$$

Using induction argument, there exist orthogonal matrices $U_2 \in \mathbb{R}^{(m-1)\times(m-1)}$ and $V_2 \in \mathbb{R}^{(n-1)\times(n-1)}$ such that $B = U_2 \Sigma_2 V_2^T$, where $\Sigma_2 \in \mathbb{R}^{(m-1)\times(n-1)}$ is diagonal. Then

$$\begin{aligned} A &= U_1 \begin{bmatrix} \sigma_1 & 0 \\ 0 & B \end{bmatrix} V_1^T \\ &= U_1 \begin{bmatrix} \sigma_1 & 0 \\ 0 & U_2 \Sigma_2 V_2^T \end{bmatrix} V_1^T \\ &= U_1 \begin{bmatrix} 1 & 0 \\ 0 & U_2 \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}^T V_1^T \\ &\equiv U\Sigma V^T, \end{aligned}$$

where

$$U = U_1 \begin{bmatrix} 1 & 0 \\ 0 & U_2 \end{bmatrix}, \quad V = V_1 \begin{bmatrix} 1 & 0 \\ 0 & V_2 \end{bmatrix}, \quad \Sigma = \begin{bmatrix} \sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix}.$$

This proves the theorem. ∎

The σ_i are called the singular values of A and the vectors u_i and v_i the i-th left singular vector and the i-th right singular vector, respectively. We usually use $\sigma_{max}(A)$ to denote the largest singular value of A and $\sigma_{min}(A)$ the smallest singular value of A.

Suppose $A \in \mathbb{R}^{m\times n}$ and $\text{rank}(A) = r$. Then

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > \sigma_{r+1} = \sigma_p = 0,$$

and A can be expressed as the summation of r rank-one matrices

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T. \tag{1.40}$$

The SVD gives complete information about the four fundamental subspaces associated with A:

$$\begin{aligned} \mathcal{R}(A) &= \text{span}\,\{u_1, u_2, \dots, u_r\}, & (1.41)\\ \mathcal{N}(A) &= \text{span}\,\{v_{r+1}, v_{r+2}, \dots, v_n\}, & (1.42)\\ \mathcal{R}(A^T) &= \text{span}\,\{v_1, v_2, \dots, v_r\}, & (1.43)\\ \mathcal{N}(A^T) &= \text{span}\,\{u_{r+1}, u_{r+2}, \dots, u_m\}. & (1.44) \end{aligned}$$

The matrix 2-norm and Frobenius norm can be characterized in terms of singular values.

$$\begin{aligned} \|A\|_2 &= \sigma_1, & (1.45)\\ \|A\|_F &= \sqrt{\sigma_1^2 + \cdots + \sigma_r^2}. & (1.46) \end{aligned}$$

If we write

$$A = U\Sigma V^T = U \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix} V^T,$$

where

$$\Sigma_r = \begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_r \end{bmatrix} \in \mathbb{R}^{r\times r},$$

then

$$A^T A = V\Sigma^T U^T U \Sigma V^T = V\Sigma^T \Sigma V^T = V \begin{bmatrix} \Sigma_r^2 & 0 \\ 0 & 0 \end{bmatrix} V^T$$

and

$$AA^T = U\Sigma V^T V \Sigma^T U^T = U\Sigma\Sigma^T U^T = U \begin{bmatrix} \Sigma_r^2 & 0 \\ 0 & 0 \end{bmatrix} U^T.$$

Thus $\sigma_1^2, \ldots, \sigma_r^2$ are the nonzero eigenvalues of $A^T A$ and AA^T, and v_j and u_j are the corresponding eigenvectors. When A is symmetric, $A^T A = AA^T = A^2$, and has real eigenvalues λ_i, then the singular values are given by $\sigma_i = |\lambda_i|$ for $i = 1, \ldots, r$. In principal, the SVD can be found from the eigenvalue decomposition of $A^T A$ and AA^T. However, this does not lead to a stable algorithm for computing the SVD. Finally, when A is $n \times n$ and square, then $|\det(A)| = \sigma_1 \cdots \sigma_n$.

Chapter 2

Computer Arithmetic

In this chapter, we explain the floating-point number system and develop basic facts about roundoff errors. Some topics such as loss of significance, stability of numerical algorithms, and condition of problems will be introduced with examples. Error analysis of computer arithmetic will be brifely discussed.

2.1 Floating-Point Number and Roundoff Error

In general, a nonzero real number x in the decimal number system can be written as

$$x = \pm r \times 10^n,$$

where

$$\frac{1}{10} \leq r < 1,$$

and n is an integer (positive, negative, or zero). This representation is the called normalized scientific notation, r is called the mantissa and n is the exponent. Note that the leading digit in the fraction is not zero (except when the number involved is zero). For example,

$$\begin{aligned} 42.965 &= 4 \times 10^1 + 2 \times 10^0 + 9 \times 10^{-1} + 6 \times 10^{-2} + 5 \times 10^{-3} \\ &= 0.42965 \times 10^2, \\ -0.00234 &= -0.234 \times 10^{-2}. \end{aligned}$$

Likewise, we can use the scientific notation for binary number system to express the number x as

$$x = \pm q \times 2^m$$

with

$$\frac{1}{2} \leq q < 1,$$

and some integer m. Both q and m will be expressed in terms of binary numbers. For example,

$$\begin{aligned} 1001.1101 &= 1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-4} \\ &= 0.10011101 \times 2^{100} \\ &= (9.8125)_{10} \end{aligned}$$

Allmost all microcomputers uses binary number system, but not necessary mainframe computers.

Example 2.1 *What is the binary representation of $\frac{2}{3}$?*

Sol: To determine the binary representation for $\frac{2}{3}$, we write

$$\frac{2}{3} = (0.a_1a_2a_3\dots)_2.$$

Multiply by 2 to obtain

$$\frac{4}{3} = (a_1.a_2a_3\dots)_2.$$

Therefore, we get $a_1 = 1$ by taking the integer part of both sides. Substracting 1, we have

$$\frac{1}{3} = (0.a_2a_3a_4\dots)_2.$$

Repeating the previous step, we arrive at

$$\frac{2}{3} = (0.101010\dots)_2.$$

■

Most computers work internally in binary system and output decimal system for human users. A number that has a terminating exapnsion in one base may have a nonterminating exapnsion in another. The conversion procedure frequently involves a small error as shown in the following example.

Example 2.2 *What is the binary representation of $\frac{1}{10}$?*

Sol:

$$\frac{1}{10} = (0.1)_{10} = (0.0001\ 1001\ 1001\ 1001\ \dots)_2.$$

■

Typically, only a relatively small subset of the real number system is used for the representation of all the real numbers. This subset, which are called the *floating-point numbers*, contains only rational numbers, both positive and negative. When a number can not be represented exactly with the fixed finite number of digits in a computer, a near-by floating-point number is chosed for approximate representation.

For any real number x, let

$$x = \pm 0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m, \quad a_1 \neq 0, \tag{2.1}$$

denote the normalized scientific binary representation of x. Note that the leading digit in mantissa $a_1 \neq 0$, hence $a_1 = 1$. If x is within the numerical range of the machine, the floating-point form of x, denoted $fl(x)$, is obtained by terminating the mantissa of x at t digits for some integer t. There are two ways of preforming this termination.

1. **chopping:** simply discard the excess bits $a_{t+1}, a_{t+2}, \ldots$ to obtain

$$fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m. \tag{2.2}$$

2. **rounding up:** add $2^{-(t+1)} \times 2^m$ to x and then chop the excess bits to obtain a number of the form

$$fl(x) = \pm 0.\delta_1 \delta_2 \cdots \delta_t \times 2^m. \tag{2.3}$$

   In this method, if $a_{t+1} = 1$, we add 1 to a_t to obtain $fl(x)$, and if $a_{t+1} = 0$, we merely chop off all but the first t digits.

Since computers can only store real numbers using fixed finite numbers of digits, this places a restriction on the precision with which real numbers can be represented. The error results from replacing a number with its floating-point form is called *roundoff error* or *rounding error*, regardless of whether the rounding or chopping method is used. The following definition describes two methods for measuring approximation errors.

Definition 2.1 (Absolute Error and Relative Error) *If x is an approximation to the exact value $x^\star$, the absolute error is $|x^\star - x|$ and the relative error is $\frac{|x^\star - x|}{|x^\star|}$, provided that $x^\star \neq 0$.*

Remark 2.1 *As a measure of accuracy, the absolute error may be misleading and the relative error more meaningful.*

If the floating-point representation $fl(x)$ for the number x is obtained by using t digits and chopping procedure, then the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.00 \cdots 0 a_{t+1} a_{t+2} \cdots \times 2^m|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots \times 2^m|} = \frac{|0.a_{t+1} a_{t+2} \cdots|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2} \cdots|} \times 2^{-t}.$$

Since $a_1 \neq 0$, the minimal value of the denominator is $\frac{1}{2}$. The numerator is bounded above by 1. As a consequence

$$\left| \frac{x - fl(x)}{x} \right| \leq 2^{-t+1}. \tag{2.4}$$

In a similar manner, if t-digit rounding arithmetic is used and $a_{t+1} = 0$, then $fl(x) = \pm 0.a_1 a_2 \cdots a_t \times 2^m$. A bound for the relative error is

$$\frac{|x - fl(x)|}{|x|} = \frac{|0.a_{t+1}a_{t+2}\cdots|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2}\cdots|} \times 2^{-t} \le 2^{-t}.$$

since the numerator is bounded above by $\frac{1}{2}$. But if $a_{t+1} = 1$, then $fl(x) = \pm(0.a_1 a_2 \cdots a_t + 2^{-t}) \times 2^m$, the upper bound for relative error becomes

$$\frac{|x - fl(x)|}{|x|} = \frac{|1 - 0.a_{t+1}a_{t+2}\cdots|}{|0.a_1 a_2 \cdots a_t a_{t+1} a_{t+2}\cdots|} \times 2^{-t} \le 2^{-t}.$$

since the numerator is bounded by $\frac{1}{2}$ due to $a_{t+1} = 1$. Therefore the relative error for rounding arithmetic is

$$\left|\frac{x - fl(x)}{x}\right| \le 2^{-t} = \frac{1}{2} \times 2^{-t+1}. \tag{2.5}$$

The number $\epsilon_M \equiv 2^{-t+1}$ is referred to as the *unit roundoff error* or *machine epsilon*. The floating-point representation, $fl(x)$, of x can be expressed as

$$fl(x) = x(1+\delta), \qquad |\delta| \le \varepsilon_M. \tag{2.6}$$

In 1985, the IEEE (Institute for Electrical and Electronic Engineers) published a report called *Binary Floating Point Arithmetic Standard 754-1985*. In this report, formats were specified for single, double, and extended precisions, and these standards are generally followed by microcomputer manufactures using floating-point hardware.

The single precision IEEE standard floating-point format allocates 32 bits for the normalized floating-point number $\pm q \times 2^m$ as shown in Figure 2.1. The first bit is a sign indicator, denoted s. This is followed by an 8-bit exponent c and a 23-bit mantissa f. The base for the exponent and mantissa is 2, and the actual exponent is $c - 127$. The value of c is restricted by the inequality $0 < c < 255$. The values 0 and 255 are reserved for special cases including ± 0 and $\pm\infty$, respectively. Hence the actual exponent of the number is restricted by the inequality $-126 \le c - 127 \le 127$. In addition, a normalization is imposed that requires that the leading digit in fraction be 1, and this digit is not stored as part of the 23-bit mantissa. Using this format gives a floating-point number of the form

$$(-1)^s \times (1.f)_2 \times 2^{c-127}.$$

Since the mantissa f actually corresponds to 24 binary digits (i.e., precision $t = 24$), the machine epsilon is

$$\epsilon_M = 2^{-24+1} = 2^{-23} \approx 1.192 \times 10^{-7}. \tag{2.7}$$

This approximately corresponds to 6 accurate decimal digits. And the first single precision floating-point number greater than 1 is $1 + 2^{-23}$.

The largest number that can be represented by the single precision format is approximately $2^{128} \approx 3.403 \times 10^{38}$, and the smallest positive number is $2^{-126} \approx 1.175 \times 10^{-38}$. This

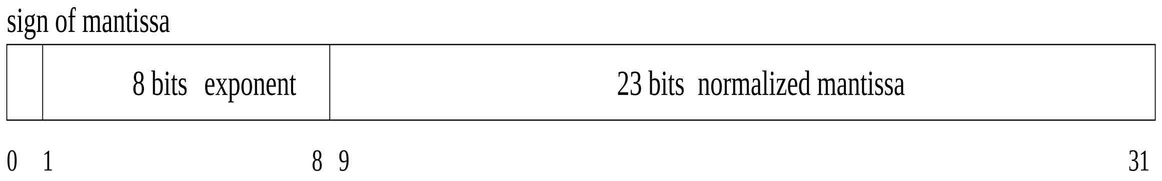


Figure 2.1: 32-bit single precision.

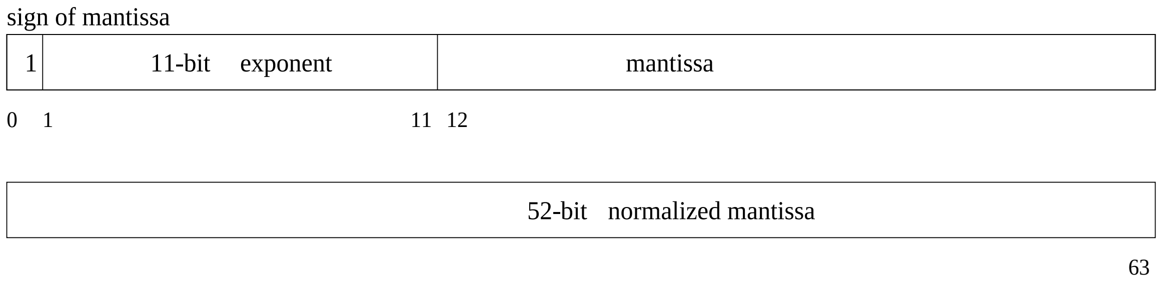


Figure 2.2: 64-bit double precision.

range is in general not sufficient for some scientific calculation. For this and other reason, we need "double precision" arithmetic.

A floating point number in double precision IEEE standard format uses two words (64 bits) to store the number as shown in Figure 2.2. The first bit is a sign indicator, denoted s, same as the single precision format. This is followed by an 11-bit exponent c and a 52-bit mantissa f. The actual exponent is $c-1023$. Using this format gives a floating-point number of the form

$$(-1)^s \times (1.f)_2 \times 2^{c-1023}.$$

which provides between 15 and 16 decimal digits of accruacy, since the machine epsilon

$$\epsilon_M = 2^{-52} \approx 2.220 \times 10^{-16},$$

and a range of approximately $2^{-1022} \approx 2.225 \times 10^{-308}$ to $2^{1024} \approx 1.798 \times 10^{308}$. The first double precision floating-point number greater than 1 is $1 + 2^{-52}$.

Table 2.1 summarizes some characteristics of IEEE standard floating-point representations. Clearly, for the most accuracy, computations should be done using double precision floating-point numbers, however, the execution time is much higher.

In the IEEE floating-point standard, the *round to nearest* or *correctly rounded* value of the real number x, denoted $round(x)$, is defined as follows. First, let x_+ be the closest foating-point number greater than x and x_- be the closest one less than x. Then $round(x)$ is either x_+ or x_-, whichever is nearer to x. If x is a floating-point number, then $round(x) = x$. This method is almost always used since it is the most useful and gives the floating-point number closest to x. Other rounding modes could be (1) *round towards 0*: $round(x)$ is either x_+ or x_-, whichever is between 0 and x; (2) *round towards* $-\infty$/*round dwon*: $round(x) = x_-$; (3) *round towards* ∞/*round up*: $round(x) = x_+$.

	single precision	double precision
ϵ_M	$2^{-23} \approx 1.192 \times 10^{-7}$	$2^{-52} \approx 2.220 \times 10^{-16}$
smallest positive number	$2^{-126} \approx 1.175 \times 10^{-38}$	$2^{-1022} \approx 2.225 \times 10^{-308}$
largest number	$2^{128} \approx 3.403 \times 10^{38}$	$2^{1024} \approx 1.798 \times 10^{308}$
decimal precision	6	15

Table 2.1: Some characteristics of IEEE standard floating-point numbers

If a number $x = \pm q \times 2^m$ with m outside the computer's possible range (too large or two small), then we say that an *overflow* or an *underflow* has occurred. Generally, an overflow results in a fatal error (or exception), and the normal excution of the program stops. An underflow, however, is usually treated automatically by setting x to zero without any interruption of the program but with a warning message in most computers.

There are some useful special numbers in the IEEE standard. For example, instead of terminating with an overflow when dividing a nonzero number by 0, the machine representation for ∞, Inf, is stored, which is the mathematically sensible thing to do. There are two different representations, +Inf and −Inf, that correspond to two quite different numbers, $+\infty$ and $-\infty$. A NaN stands for Not a Number and is an error pattern rather than a number. Table 2.2 lists the IEEE exception handling standard.

big*big	± Inf	overflow
number/0.0	± Inf	division
0.0/0.0	NaN	invalid
small/big	subnormal number	underflow
2.0/3.0	rounded	

Table 2.2: IEEE exception handling.

Because of the hidden bit representation, a special technique for storing zero is necessary. Note that all zero bits in the mantissa field represents the significant 1.0 rather than 0.0. Moreover, there are two different representations for the same number zero; namely +0 and −0.

For single precision integers, 31 bits are allocated for the integer, because onley 1 bit is needed for the sign. Consequently, the range for integers is from $-(2^{31}-1)$ to $(2^{31}-1) = 2147483647$. Double precision integers, on the other hand, use 63-bit storage and 1 bit for sign. Pure integer calculation is not common in numerical algorithms.

2.2 Loss of Significance, Stability, and Conditioning

Roundoff errors are inevitable and difficult to control. Other types of errors which occur in computation may be under our control. The subject of numerical analysis is largely preoccupied with understanding and controlling errors of various kinds. Here in this section we examine some of them.

2.2.1 Loss of Significance

Suppose that x is a real number expressed in normalized scientific notation in the decimal system

$$x = \pm a_1 a_2 a_3 \cdots \times 10^n \qquad a_1 \neq 0.$$

The digits $a_1, a_2, a_3, \ldots,$ used to express the fraction part of x do not all have the same significance because they represent different power of 10. Thus, we say that a_1 is the most significant digit, and the significance of the digits diminishes from left to right.

One of the most common error-producing calculations involves the cancellation of significant digits due to the subtraction of nearly equal numbers (or the addition of one very large number and one very small number). Assume that two nearly equal numbers x and y, with $x > y$, have the t-digit floating-point representations

$$fl(x) = 0.d_1 d_2 \cdots d_p \alpha_{p+1} \alpha_{p+2} \cdots \alpha_t \times 10^n,$$

and

$$fl(x) = 0.d_1 d_2 \cdots d_p \beta_{p+1} \beta_{p+2} \cdots \beta_t \times 10^n.$$

Then the floating-point form of $x - y$ is

$$fl(fl(x) - fl(y)) = 0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t \times 10^n,$$

where

$$0.\sigma_{p+1} \sigma_{p+2} \cdots \sigma_t = 0.\alpha_{p+1} \alpha_{p+2} \cdots \alpha_t - 0.\beta_{p+1} \beta_{p+2} \cdots \beta_t.$$

As a result, the floating-point number used to represent $x - y$ has at most $t - p$ digits of significance. However, in most computers, $x - y$ will be assigned t digits, with the last p digits being either zero or randomly assigned. Any further calculations involving the results of $x - y$ retain the problem of having only $t - p$ digits of significance, since a chain of calculations can not be expected to be more accurate than its weakest portion. The phenomenon can be illustrated with the following example.

Example 2.3 *If $x = 0.3721478693$ and $y = 0.3720230572$, what is the relative error in the computation of $x - y$ using five decimal digits of accuracy?*

Sol: In exact computation using ten decimal digits of accuracy,

$$x - y = 0.0001248121.$$

But both x and y will be rounded to five decimal digits before subtraction. Thus

$$\begin{aligned} fl(x) &= 0.37215 \\ fl(y) &= 0.37202 \\ fl(x) - fl(y) &= 0.00013 = 0.13000 \times 10^{-3} \end{aligned}$$

Therefore the relative error is

$$\frac{(x-y)-(fl(x)-fl(y))}{x-y} \approx 0.04 = 4\%.$$

■

An interesting question arised immediately would be how many significant binary bits are lost in the subtraction when x is close to y?

Theorem 2.1 *If $x \geq 0$ and $y \geq 0$ are normalized floating-point binary numbers such that $x > y$ and*

$$2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p},$$

then at most q and at least p significant binary digits are lost in the subtraction $x - y$.

Proof: Write

$$x = r \times 2^n, \qquad \frac{1}{2} \leq r < 1$$

and

$$y = s \times 2^m, \qquad \frac{1}{2} \leq s < 1.$$

Since $x > y$, we must sift the decimal digits of y to the right

$$y = (s \times 2^{m-n}) \times 2^n.$$

Then

$$x - y = (r - s \times 2^{m-n}) \times 2^n = r\left(1 - \frac{s \times 2^m}{r \times 2^n}\right) \times 2^n = r\left(1 - \frac{y}{x}\right) \times 2^n.$$

By assumption $2^{-q} \leq 1 - \frac{y}{x} \leq 2^{-p}$, hence

$$r\left(1 - \frac{y}{x}\right) < 1 \cdot 2^{-p} = 2^{-p}.$$

This means that to normalize the result $x - y$, a shift of at least p bits to the left is required. Similarly,

$$r\left(1 - \frac{y}{x}\right) \geq \frac{1}{2} \cdot 2^{-q} = 2^{-(q+1)},$$

and a shift of at most q bits to the right is required. ■

Sometimes, loss of significance can be avoided by using double precision. This at least doubles the number of bits in mantissa. But when both operands are already in double precision, this means quadruple precision would be needed, and this is not available in many machines and languages, or at least not cheaply. In some cases, rewriting the mathematical formula is an alternative and can make the difference.

Example 2.4 *Consider the two equivalent functions*

$$f(x) = x(\sqrt{x+1} - \sqrt{x}) \quad and \quad g(x) = \frac{x}{\sqrt{x+1} + \sqrt{x}}.$$

Compare the function evaluation of $f(500)$ and $g(500)$ using 6 digits and rounding.

Sol:

$$\begin{aligned} f(500) &= 0.500000 \times 10^3 \times (\sqrt{501} - \sqrt{500}) \\ &= 0.500000 \times 10^3 \times (0.223830 \times 10^2 - 0.223607 \times 10^2) \\ &= 0.500000 \times 10^3 \times 0.223000 \\ &= 0.111500 \times 10^3 \end{aligned}$$

and

$$\begin{aligned} g(500) &= \frac{500}{\sqrt{501} + \sqrt{500}} \\ &= \frac{0.500000 \times 10^3}{0.223830 \times 10^2 + 0.223607 \times 10^2} \\ &= \frac{0.500000 \times 10^3}{0.447437 \times 10^2} \\ &= 0.111748 \times 10^2 \end{aligned}$$

If more digits are used, we can calculated

$$\begin{aligned} f(500) &= 500 \times (\sqrt{501} - \sqrt{500}) \\ &= 500 \times (22.38302929 - 22.36067977) \\ &= 500 \times 0.022349516 \\ &= 11.1747553 \end{aligned}$$

Hence it can be argued that the formulation $g(x)$ is better. ∎

Example 2.5 *The quadratic formulas for computing the roots of $ax^2 + bx + c = 0$, when $a \neq 0$, are*

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad and \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}.$$

Consider the quadratic equation $x^2 + 62.10x + 1 = 0$ and discuss the numerical results.

Sol: Using the quadratic formula and 8-digit rounding arithmetic, one can obtain the roots

$$x_1 = -0.01610723 \qquad \text{and} \qquad x_2 = -62.08390.$$

We will use these values as "exact solutions". Now we perform the calculations with 4-digit rounding arithmethic. First we have

$$\sqrt{b^2 - 4ac} = \sqrt{62.10^2 - 4.000} = \sqrt{3856 - 4.000} = \sqrt{3852} = 62.06,$$

and

$$fl(x_1) = \frac{-62.10 + 62.06}{2.000} = \frac{-0.04000}{2.000} = -0.02000.$$

The relative error in computing x_1 is

$$\frac{|fl(x_1) - x_1|}{|x_1|} = \frac{|-0.02000 + 0.01610723|}{|-0.01610723|} = \frac{0.00389277}{0.01610723} \approx 0.2417.$$

In calculating x_2,

$$fl(x_2) = \frac{-62.1062.06}{2.000} = \frac{-124.2}{2.000} = -62.10,$$

and the relative error in computing x_2 is

$$\frac{|fl(x_2) - x_2|}{|x_2|} = \frac{|-62.10 + 62.08390|}{|-62.08390|} = \frac{0.0161}{62.08390} \approx 0.259 \times 10^{-3}.$$

In this equation, $b^2 = 62.10^2$ is much larger than $4ac = 4$. Hence b and $\sqrt{b^2 - 4ac}$ become two nearly equal numbers. The calculation of x_1 involves the subtraction of two nearly equal numbers, in contrast, the calculation of x_2 involves the addition of the nearly equal numbers which will not cause serious loss of significance.

To obtain a more accurate 4-digit rounding approximation for x_1, we change the formulation by rationalizing the numerator, that is,

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}.$$

Then

$$fl(x_1) = \frac{-2.000}{62.10 + 62.06} = \frac{-2.000}{124.2} = -0.01610.$$

The relative error in computing x_1 is now reduced to 0.62×10^{-3}. However, if we rationalize the numberator in x_2 to get

$$x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}.$$

The use of this formula results not only involve the subtraction of two nearly equal numbers but also the division by the small number. This would cause degrade in accuracy.

$$fl(x_2) = \frac{-2.000}{62.10 - 62.06} = \frac{-2.000}{0.04000} = -50.00.$$

The relative error in x_2 becomes 0.19. ■

The above example also shows that if a finite-precision representation or calculation introduces an error, further enlargement of the error occurs when dividing by a number with small magnitude, or, equivalently, when multiplying by a number with large magnitude.

There is another situation in which a drastic loss of significant digits will occur. This is in the evaluation of polynomials and certain functions.

Example 2.6 *Evaluate*

$$f(x) = x^3 - 6x^2 + 3x - 0.149$$

at $x = 4.71$ *using 3-digit arithmetic.*

Sol: The following table gives the intermediate results in the calculation.

	x	x^2	x^3	$6x^2$	$3x$
Exact	4.71	22.1841	104.487111	133.1046	14.13
3-digit chopping	4.71	22.1	104	132	14.1
3-digit rounding	4.71	22.2	105	133	14.1

The exact function value is

$$f(4.71) = -14.636489.$$

The result from 3-digit chopping arithmetic is

$$\hat{f}(4.71) = -14.0,$$

and 3-digit rounding

$$\tilde{f}(4.71) = -14.0.$$

Hence relative error for both calculation is

$$\frac{|-14.636489 + 14.0|}{|-14.636489|} \approx 0.4 \times 10^{-1}.$$

Alternatively, if we rewrite the formulation of $f(x)$ in nested manner as

$$f(x) = ((x - 6)x + 3)x - 0.149.$$

Then the 3-digit chopping arithmetic gives

$$\hat{f}(4.17) = -14.5$$

with relative error

$$\frac{|-14.636489 + 14.5|}{|-14.636489|} \approx 0.93 \times 10^{-2},$$

and 3-digit rounding

$$\tilde{f}(4.17) = -14.6$$

with relative error

$$\frac{|-14.636489 + 14.6|}{|-14.636489|} \approx 0.25 \times 10^{-2}.$$

■

Example 2.7 *Let*

$$\begin{aligned} p(x) &= ((x^3 - 3x^2) + 3x) - 1, \\ q(x) &= ((x - 3)x + 3)x - 1. \end{aligned}$$

Compare the function values at $x = 2.19$.

Sol: Use 3-digit and rounding for $p(2.19)$ and $q(2.19)$.

$$\begin{aligned} \hat{p}(2.19) &= ((2.19^3 - 3 \times 2.19^2) + 3 \times 2.19) - 1 \\ &= ((10.5 - 14.4) + 3 \times 2.19) - 1 \\ &= (-3.9 + 6.57) - 1 \\ &= 2.67 - 1 \\ &= 1.67 \end{aligned}$$

$$\begin{aligned} \hat{q}(2.19) &= ((2.19 - 3) \times 2.19 + 3) \times 2.19 - 1 \\ &= (-0.81 \times 2.19 + 3) \times 2.19 - 1 \\ &= (-1.77 + 3) \times 2.19 - 1 \\ &= 1.23 \times 2.19 - 1 \\ &= 2.69 - 1 \\ &= 1.69 \end{aligned}$$

With more digits, one can have

$$p(2.19) = g(2.19) = 1.685159$$

Hence the absolute errors are

$$|p(2.19) - \hat{p}(2.19)| = 0.015159$$

and

$$|q(2.19) - \hat{q}(2.19)| = 0.004841,$$

respectively. One can observe that the evaluation formula $q(x)$ is better than $p(x)$. ■

In fact, polynomials should always be expressed in nested form before performing an evaluation, since this form minimizes the number of required arithmetic calculations. One way to reduce roundoff error is to reduce the number of error-producing computations.

Example 2.8 *How to evaluate*

$$y = x - \sin x$$

when x *is small?*

Sol: Since $x \approx \sin x$ for small x, the computation will cause loss of significance. Alternatively, use Taylor series for $\sin x$ so that

$$\begin{aligned} y &= x - \left(x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots\right) \\ &= \frac{x^3}{3!} - \frac{x^5}{5!} + \frac{x^7}{7!} - \frac{x^9}{9!} + \cdots \\ &= \frac{x^3}{6} - \frac{x^5}{6 \times 20} + \frac{x^7}{6 \times 20 \times 42} - \frac{x^9}{6 \times 20 \times 42 \times 72} \cdots \\ &= \frac{x^3}{6}\left(1 - \frac{x^2}{20}\left(1 - \frac{x^2}{42}\left(1 - \frac{x^2}{72}(\cdots)\right)\right)\right) \end{aligned}$$

■

Finally, floating-point arithmetic is not always associative. This can be illustrated with the following example.

Example 2.9 *Performing the calculation with 3-decimal mantissa and chopping arithmetic will result*

$$fl(fl(10^{-3} + 1) - 1) = 0$$

and

$$fl(fl(10^{-3} + fl(1 - 1))) = 10^{-3}.$$

■

2.2.2 Numerical Stability

Another theme that occurs repeatedly in numerical analysis is the distiction between numerical algorithms that are stable and those that are not. Informally speaking, a numerical process is unstable if small errors made at one stage of the process are magnified and propagated in subsequent stages and seriously degrade the accuracy of the overall calculation. Whether a process is stable or unstable should be decided on the basis of relative error.

Example 2.10 *Consider the following recurrence algorithm*

$$\begin{cases} x_0 = 1, & x_1 = \frac{1}{3} \\ x_{n+1} = \frac{13}{3}x_n - \frac{4}{3}x_{n-1} \end{cases}$$

for computing the sequence of $\{x_n = (\frac{1}{3})^n\}$. *This algorithm is unstable.*

Sol: A computer implemention of the recurrence algorithm gives the following result.

n	x_n	n	x_n	n	x_n	n	x_n
0	1.0000000	4	0.0123466	8	0.0003757	12	0.0571502
1	0.3333333	5	0.0041187	9	0.0009437	13	0.2285939
2	0.1111112	6	0.0013857	10	0.0035887	14	0.9143735
3	0.0370373	7	0.0005153	11	0.0142927	15	3.6574934

The error present in x_n is multiplied by $\frac{13}{3}$ in computing x_{n+1}. For example, the error will be propagated with a factor of $\left(\frac{13}{3}\right)^{14}$ in computing x_{15}. Additional roundoff errors in computing $x_2, x_3, \ldots$ may also be propagated and added to that of x_{15}. ■

2.2.3 Conditioning

The words condition and conditioning are used informally to indicate how sensitive the solution of a problem may be to small relative changes in the input data. A problem is ill-conditioned if small changes in the data can produce large changes in the resluts. For certain types of problems, a condition number can be defined. If that number is large, it indicates an ill-conditioned problem. In contrast, if the number is modest, the problem is recognized as a well-conditioned problem.

For example, the condition number for a nonsingular square matrix A is defined as

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

with respect to some matrix norm. The matrix A is said to be ill-conditioned if $\kappa(A)$ is large, and well-conditioned when $\kappa(A)$ is modest. For a general rectangular matrix, the singular values are used to characterize the condition number in 2-norm

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

where $\sigma_{max}(A)$ is the largest singular value and $\sigma_{min}(A)$ the smallest singular value of A.

A well-known ill-conditioned matrix is the Hilbert matrix

$$H_n = [h_{ij}] \in \mathbb{R}^{n\times n}, \qquad \text{where} \qquad h_{ij} = \frac{1}{i+j-1}.$$

In general, ill-conditioning is not easy to detect.

In solving a system of linear equations $Ax = b$ in which A is ill-conditioned, small perturbation in b will cause large perturbation in x.

Example 2.11 *Consider solving the following system of linear equations*

$$\begin{bmatrix} 20514 & 4424 & 987 & 224 \\ 4424 & 987 & 224 & 54 \\ 987 & 224 & 54 & 14 \\ 224 & 54 & 14 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 20514 \\ 4424 \\ 987 \\ 224 \end{bmatrix}.$$

Sol: The numerical solution obtained by using Matlab command $A\backslash b$, where A is the coefficient matrix and b is the right-hand-side vector, is

$$x = \begin{bmatrix} 1.00000000000002 \\ -0.00000000000003 \\ -0.00000000000055 \\ 0.00000000000125 \end{bmatrix}.$$

However, if b_1 is changed from 20514 to 20515, then the numerical solution becomes

$$x = \begin{bmatrix} 0.99672906602257 \\ -0.00528381642514 \\ 0.16978663445968 \\ -0.33974939613341 \end{bmatrix}.$$

One may check that the 2-norm condition number of the coefficient is $\kappa_2(A) \approx 6.1191 \times 10^5$.
■

2.3 Floating-Point Error Analysis

In addition to inaccurate representation of numbers, the arithmetic performed in a computer is not exact. The arithmetic involves manipulating binary digits by various shifting, or logical, operations. We now consider the errors that are produced in the course of elementary arithmetic operations.

Let $\odot$ stand for any one of the four basic arithmetic operators $+$, $-$, $\star$, $\div$. Whenever two machine numbers x and y are to be combined arithmetically, the computer will produce $fl(x \odot y)$ instead of $x \odot y$. In the operation, $x \odot y$ is first correctly formed, normalized, and then rounded to become a machine number. Under this assumption and (2.6), the relative error of $fl(x \odot y)$ satisfies

$$fl(x \odot y) = (x \odot y)(1 + \delta), \quad \delta \le \epsilon_M, \tag{2.8}$$

where ϵ_M is the unit roundoff. But if x, y are not machine numbers, then they must first rounded to floating-point format before the arithmetic operation and the resulting relative error becomes

$$fl(fl(x) \odot fl(y)) = (x(1+\delta_1) \odot y(1+\delta_2))(1+\delta_3), \quad \delta_i \le \epsilon_M, \ \ i = 1, 2, 3. \tag{2.9}$$

Thus, the machine unit roundoff gives an upper bound for the relative error in any single basic arithmetic operation.

The analysis (2.8) can be extended to arithmetic operations on three floating-point numbers. For example,

$$\begin{aligned} fl(x(y+z)) &= (x \cdot fl(y+z))(1+\delta_1) \\ &= (x(y+z)(1+\delta_2))(1+\delta_1) \\ &= x(y+z)(1+\delta_1+\delta_2+\delta_1\delta_2) \\ &\approx x(y+z)(1+\delta_1+\delta_2) \\ &= x(y+z)(1+\delta_3) \end{aligned}$$

The generalization to the sum of n floating-point numbers is as follows.

Theorem 2.2 *Let $x_0, x_1, \ldots, x_n$ be positive floating-point numbers. Then the relative error in computing $\sum_{i=0}^{n} x_i$ in the usual way is at most $(1+\epsilon_M)^n - 1 \approx n\epsilon_M$, where ϵ_M is the machine unit roundoff.*

Proof: Let $S_k = x_0 + x_1 + \cdots + x_k$ and $S_k^\star = fl(S_k)$, and be computed in the usual way

$$\begin{cases} S_0 = x_0 \\ S_{j+1} = S_j + x_{j+1} \end{cases} \qquad \text{and} \qquad \begin{cases} S_0^\star = fl(x_0) \\ S_{j+1}^\star = fl(S_j^\star + x_{j+1}) \end{cases}$$

Define

$$\rho_k = \frac{S_k^* - S_k}{S_k} \qquad \text{and} \qquad \delta_k = \frac{S_{k+1}^* - S_{k+1}}{S_k^* + x_{k+1}},$$

where each $|\delta_k| \leq \epsilon_M$. Then

$$\begin{aligned}
\rho_{k+1} &= \frac{S_{k+1}^* - S_{k+1}}{S_{k+1}} \\
&= \frac{(S_k^* + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{(S_k(1+\rho_k) + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{(S_k + \rho_k S_k + x_{k+1})(1+\delta_k) - (S_k + x_{k+1})}{S_{k+1}} \\
&= \frac{S_k + \rho_k S_k + x_{k+1} + \delta_k S_k + \delta_k \rho_k S_k + \delta_k x_{k+1} - S_k - x_{k+1}}{S_{k+1}} \\
&= \frac{\rho_k S_k + \delta_k S_k + \delta_k \rho_k S_k \delta_k x_{k+1}}{S_{k+1}} \\
&= \frac{\delta_k (S_k + x_{k+1}) + \rho_k (S_k + \delta_k S_k)}{S_{k+1}} \\
&= \frac{\delta_k S_{k+1} + \rho_k S_k (1+\delta_k)}{S_{k+1}} \\
&= \delta_k + \rho_k \left(\frac{S_k}{S_{k+1}} \right) (1+\delta_k).
\end{aligned}$$

Since x_i are all positive, $S_k < S_{k+1}$, and

$$|\rho_{k+1}| \leq \epsilon_M + |\rho_k|(1+\epsilon_M) = \epsilon_M + \theta|\rho_k|,$$

where $\theta = 1 + \epsilon_M$. Thus

$$\begin{aligned}
|\rho_0| &= 0 \\
|\rho_1| &\leq \epsilon_M \\
|\rho_2| &\leq \epsilon_M + \epsilon_M \theta \\
|\rho_3| &\leq \epsilon_M + \epsilon_M \theta + \epsilon \theta^2 \\
&\vdots
\end{aligned}$$

In general,

$$\begin{aligned}
|\rho_n| &\leq \epsilon_M + \epsilon_M\theta + \epsilon\theta^2 + \cdots + \epsilon\theta^{n-1} \\
&= \epsilon\left(1 + \theta + \theta^2 + \cdots + \theta^{n-1}\right) \\
&= \epsilon\left(\frac{\theta^n - 1}{\theta - 1}\right) \\
&= \epsilon\left(\frac{(1+\epsilon_M)^n - 1}{1 + \epsilon_M - 1}\right) \\
&= (1+\epsilon_M)^n - 1 \\
&= 1 + \binom{n}{1}\epsilon_M + \binom{n}{2}\epsilon_M^2 + \cdots - 1 \\
&\approx \binom{n}{1}\epsilon_M \\
&= n\epsilon_M.
\end{aligned}$$

■

Chapter 3

Direct Methods for Solving Systems of Linear Equations

Systems of linear equations are associated with many problems in engineering and science, as well as with applications to the social sciences and the quantitative study of business and economic problems.

The principal objective of this chapter is to discuss the numerical aspects of solving linear systems of equations having the form

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{cases} \tag{3.1}$$

This is a linear system of n equations in n unknowns $x_1, x_2, \ldots, x_n$. This system can simply be written in the matrix equation form

$$Ax = b, \tag{3.2}$$

where

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}. \tag{3.3}$$

This equation has a unique solution $x = A^{-1}b$ when the coefficient matrix A is nonsingular. Unless otherwise stated, we shall assume that this is the case. If A^{-1} is already available, then $x = A^{-1}b$ provides a good method of computing the solution x. If A^{-1} is not available, then in general A^{-1} should not be computed solely for the purpose of obtaining x. More efficient numerical procedures will be developed.

Direct methods, which are techniques that give a solution in a fixed number of steps, subject only to round-off errors, are considered in this chapter. Guassian elimination is the principal tool in the direct solution of (3.2). An important technique is to use Gaussian elimination to factor the coefficient matrix into a product of matrices that are easier to manipulate. The factorization is called LU-factorization and has the form $A = LU$, where L is unit lower triangular and U is upper triangular. We shall construct a general-purpose algorithm, analyze the errors that are associated with the computer solution, and study methods for controlling and reducing the errors.

3.1 Triangular Systems

The Gaussian elimination and LU factorization methods for linear systems involve the conversion of the given square matrix to a triangular system that has the same solution. This section is about the solution of triangular systems.

3.1.1 Diagonal System

We begin by considering the case in which A has a diagonal structure. The easiest linear system of equations (3.2) is the one with diagonal coefficient matrix A, i.e., all the nonzero elements of A are on the main diagonal,

$$A = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}.$$

In this case, computing the solution x is trivial

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1/a_{11} \\ b_2/a_{22} \\ \vdots \\ b_n/a_{nn} \end{bmatrix}, \tag{3.4}$$

provided that all $a_{ii} \neq 0$. If $a_{ii} = 0$ and $b_i = 0$ for some index i, then x_i can be any real number. If $a_{ii} = 0$ but $b_i \neq 0$, no solution of the system exists.

However, a linear system with diagonal coefficient matrix is not common and reducing a general matrix to diagonal form is not practical.

3.1.2 Forward Substitution

When a linear system $Lx = b$ is lower triangular of the form

$$\begin{bmatrix} \ell_{11} & 0 & \cdots & 0 \\ \ell_{21} & \ell_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \cdots & \ell_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}, \tag{3.5}$$

where all diagonals $\ell_{ii} \neq 0$, x_i can be obtained by the following procedure

$$\begin{aligned}
x_1 &= b_1/\ell_{11} \\
x_2 &= (b_2 - \ell_{21}x_1)/\ell_{22} \\
x_3 &= (b_3 - \ell_{31}x_1 - \ell_{32}x_2)/\ell_{33} \\
&\vdots \\
x_n &= (b_n - \ell_{n1}x_1 - \ell_{n2}x_2 - \cdots - \ell_{n,n-1}x_{n-1})/\ell_{nn}
\end{aligned}$$

The general formulation for computing x_i is

$$x_i = \left(b_i - \sum_{j=1}^{i-1} \ell_{ij} x_j \right) \Bigg/ \ell_{ii}, \qquad i = 1, 2, \ldots, n. \tag{3.6}$$

This procedure is known as the forward substitution and is used for solving a lower triangular linear system. Algorithms for forward substitution in row-oriented and column-oriented oder are presented.

Algorithm 3.1 (Forward Substitution: Row Version) *Suppose that $L \in \mathbb{R}^{n\times n}$ is nonsingular lower triangular and $b \in \mathbb{R}^n$. This algorithm computes the solution of $Lx = b$ using row-oriented procedure.*

```
for i = 1, ... , n do
   tmp = 0.0
   for j = 1, ... , i − 1 do
      tmp = tmp + L(i, j) ∗ x(j)
   end for
   x(i) = (b(i) − tmp)/L(i, i)
end for
```

Algorithm 3.2 (Forward Substitution: Column version) *Suppose that $L \in \mathbb{R}^{n\times n}$ is nonsingular lower triangular and $b \in \mathbb{R}^n$. This algorithm computes the solution of $Lx = b$ using column-oriented procedure.*

```
for i = 1, ... , n do
   x(i) = b(i)
end for
for j = 1, ... , n do
   x(j) = x(j)/L(j, j)
   for i = (j + 1), ... , n do
      x(i) = x(i) − L(i, j) ∗ x(j)
   end for
end for
```

Since b_i is only involved in the formula for x_i, therefore the memory storage for b can be overwritten by x. The number of floating-point operations, flops, involved in the forward substitution are

$$\sum_{i=1}^{n}[2(i-1)+2] = n^2 + n.$$

Hence the forward substitution algorithm is an $O(n^2)$ algorithm.

3.1.3 Back Substitution

The analogous algorithm for upper triangular system $Ux = b$ of the form

$$\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.7}$$

is called back substitution. The solution x_i are computed in a reversed order by

$$\begin{aligned} x_n &= b_n/u_{nn} \\ x_{n-1} &= (b_{n-1} - u_{n-1,n}x_n)/u_{n-1,n-1} \\ x_{n-2} &= (b_{n-2} - u_{n-2,n-1}x_{n-1} - u_{n-2,n}x_n)/u_{n-2,n-2} \\ &\vdots \\ x_1 &= (b_1 - u_{12}x_2 - u_{13}x_3 - \cdots - u_{1n}x_n)/u_{11} \end{aligned}$$

provided that all $u_{ii} \neq 0$. The general formulation is

$$x_i = \left(b_i - \sum_{j=i+1}^{n} u_{ij}x_j \right) \Big/ u_{ii}, \qquad i = n, n-1, \ldots, 1. \tag{3.8}$$

Algorithm 3.3 (Back Substitution: Row Version) *Suppose that $U \in \mathbb{R}^{n\times n}$ is nonsingular upper triangular and $b \in \mathbb{R}^n$. This algorithm computes the solution of $Ux = b$ using row-oriented procedure.*

$x(n) = b(n)/U(n,n)$
for $i = (n-1), \ldots, 1$ **do**
  $tmp = 0.0$
  for $j = i+1 : n$ **do**
    $tmp = tmp + U(i,j) * x(j)$
  end for
  $x(i) = (b(i) - tmp)/U(i,i)$
end for

Algorithm 3.4 (Back Substitution: Column Version) *Suppose that $U \in \mathbb{R}^{n\times n}$ is non-singular upper triangular and $b \in \mathbb{R}^n$. This algorithm computes the solution of $Ux = b$ using row-oriented procedure.*

> **for** $i = 1, \dots, n$ **do**
> $x(i) = b(i)$
> **end for**
> **for** $j = n, \dots, 1$ **do**
> $x(j) = x(j)/U(j,j)$
>   **for** $i = 1 : (j-1)$ **do**
> $x(i) = x(i) - U(i,j) * x(j)$
>   **end for**
> **end for**

Once again the memory storage for b can be overwritten by x. Back substitution requires $n^2 + O(n)$ flops.

3.2 Gaussian Elimination and LU Factorization

In dealing with systems of linear equations there is a concept of equivalence that is important. If two systems have precisely the same solutions, they are equivalent systems. Thus, to solve a system of linear equations, we can instead solve any equivalent system, no solutions are lost and no new ones appear.

As we have just seen in the previous section that triangular systems are "easy" to solve. The motivation at the heart of Gaussian elimination is to convert a given system $Ax = b$ to an equivalent triangular system. The conversion is achieved by taking appropriate linear combination of the equations. In this section we will give a complete specification of this central procedure and derive an algorithm that computes a matrix factorization called LU factorization such that $A = LU$, where L is unit lower triangular and U is upper triangular. The solution to the original problem $Ax = LUx = b$ is then found by a two-step triangular solve process:

$$Ly = b, \qquad Ux = y. \tag{3.9}$$

3.2.1 Gaussian Elimination

Given a system of linear equations to be solved, it can be transformed into a simpler equivalent system by the following three types of elementary operations:

1. Interchange two equations in the system (or equivalently, interchange two rows in A):

$$\mathcal{E}_i \leftrightarrow \mathcal{E}_j;$$

2. Multiply an equation by a non-zero constant (multiply one row of A by a non-zero constant):

$$\mathcal{E}_i \leftarrow \lambda \mathcal{E}_i.$$

3. Add to an equation a multiple of some other equation (add to a row a multiple of some other row):
$$\mathcal{E}_i \leftarrow \mathcal{E}_i + \lambda \mathcal{E}_j.$$

Here $\mathcal{E}_i$ denotes the i-th equation in the system. If one system of equations is obtained from another by a finite sequence of elementary operations, then the two systems can be proved to be equivalent.

The first step in the Gaussian elimination process consists of performing, for each $i = 2, 3, \dots, n$, the elementary operations
$$\mathcal{E}_i \leftarrow (\mathcal{E}_i - m_{i,1}\mathcal{E}_1), \qquad \text{where} \quad m_{i,1} = \frac{a_{i1}}{a_{11}}. \tag{3.10}$$

These operations transform the system into one in which all the entries in the first column below the diagonal are zero. Then the process is repeated on the resulting equations $\mathcal{E}_2, \dots, \mathcal{E}_n$, and so on.

The elementary operation can be carried out with matrix multiplications. An elementary matrix is defined to be an $n \times n$ matrix obtained by applying an elementary operation to the $n \times n$ identity matrix. Each elementary operation on A can be accomplished by multiplying A on the left by an elementary matrix.

To describe formally the process of Gaussian elimination, we interpret it as a succession of $n-1$ major steps resulting in a sequence of matrices as follows:
$$A = A^{(1)} \to A^{(2)} \to \cdots \to A^{(n)},$$
where $A^{(n)}$ is upper triangular. At the conclusion of step $k-1$, the matrix $A^{(k)}$ is constructed from $A^{(k-1)}$ by elementary operations similar to (3.10), and has the following form:
$$A^{(k)} = \left[\begin{array}{ccc|c|ccccc}
a_{11}^{(k)} & \cdots & a_{1,k-1}^{(k)} & a_{1k}^{(k)} & \cdots & a_{1j}^{(k)} & \cdots & a_{1n}^{(k)} \\
\vdots & \ddots & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & a_{k-1,k-1}^{(k)} & a_{k-1,k}^{(k)} & \cdots & a_{k-1,j}^{(k)} & \cdots & a_{k-1,n}^{(k)} \\
\hline
0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kj}^{(k)} & \cdots & a_{kn}^{(k)} \\
\hline
\vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & a_{ik}^{(k)} & \cdots & a_{ij}^{(k)} & \cdots & a_{in}^{(k)} \\
\vdots & & \vdots & \vdots & & \vdots & & \vdots \\
0 & \cdots & 0 & a_{nk}^{(k)} & \cdots & a_{nj}^{(k)} & \cdots & a_{nn}^{(k)}
\end{array}\right]$$

In the k-th step, $a_{kk}^{(k)}$ is used as a pivot element and the k-th row as pivot row, and elementary operations are applied to rows $k+1$ through n so that zeros are produced in column k below the diagonal. That is, $A^{(k+1)}$ is obtained from $A^{(k)}$ in which $a_{k+1,k}^{(k+1)}, \cdots, a_{nk}^{(k+1)}$ are zero, row $k+1$ through n are modified but row 1 through row k are unchanged when compared to $A^{(k)}$. More precisely, the entries of $A^{(k+1)}$ are produced by the formula
$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)}, & \text{for } i = 1, \dots, k, \text{ and } j = 1, \dots, n; \\ 0, & \text{for } i = k+1, \dots, n, \text{ and } j = 1, \dots, k; \\ a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} * a_{kj}^{(k)}, & \text{for } i = k+1, \dots, n, \text{ and } j = k+1, \dots, n. \end{cases} \tag{3.11}$$

If we collect all the multipliers

$$\ell_{ik} = \begin{cases} 0, & \text{if } i < k; \\ 1, & \text{if } i = k; \\ \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, & \text{if } i > k, \end{cases} \tag{3.12}$$

and let $L = [\ell_{ik}]$ and $U = A^{(n)}$, then L is unit lower triangular, U is upper triangular, and later we shall show that matrix A has the factorization $A = LU$.

An algorithm to carry out the basic Gaussian elimination process just described is as follows.

Algorithm 3.5 (Gaussian elimination) *Given $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$, this algorithm implements the Gaussian elimination procedure to reduce A to upper triangular and modify the entries of b accordingly.*

for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $t = A(i,k)/A(k,k)$
    $A(i,k) = 0$
    $b(i) = b(i) - t * b(k)$
    for $j = k+1, \ldots, n$ **do**
      $A(i,j) = A(i,j) - t * A(k,j)$
    end for
  end for
end for

In the algorithm, it is assumed that all the pivot elements in the elimination process are nonzero. However, it is clear from (3.11) and (3.12) the algorithm will break down if a zero $A(k,k)$ is encountered. A check for nonzero pivot should be included in practical implementation. After the reduction, A becomes upper triangular and the back substitution can be used to obtain the solution vector x.

Example 3.1 *This example illustrates the application of Gaussian elimination in solving system of linear equations.*

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 34 \\ 27 \\ -38 \end{bmatrix}$$

Sol:

1^{st} **step** Use 6 as pivot element, the first row as pivot row, and multipliers $2, \frac{1}{2}, -1$ are produced to reduce the system to

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & -12 & 8 & 1 \\ 0 & 2 & 3 & -14 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ 21 \\ -26 \end{bmatrix}$$

2^{nd} **step** Use -4 as pivot element, the second row as pivot row, and multipliers $3, -\frac{1}{2}$ are computed to reduce the system to

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 4 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ -9 \\ -21 \end{bmatrix}$$

3^{rd} **step** Use 2 as pivot element, the third row as pivot row, and multipliers 2 is found to reduce the system to

$$\begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 12 \\ 10 \\ -9 \\ -3 \end{bmatrix}$$

Now the solution can be obtained by solving this triangular system with back substitution. Note that if we collect all the multipliers and let

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ \frac{1}{2} & 3 & 1 & 0 \\ -1 & -\frac{1}{2} & 2 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -4 & 2 & 2 \\ 0 & 0 & 2 & -5 \\ 0 & 0 & 0 & -3 \end{bmatrix},$$

then one can verify that $LU = A$. ■

3.2.2 Gaussian Transformation and LU Factorization

To obtain a matrix factorization description of Gaussian elimination, it is easier to use a matrix description of the zeroing process. For a given vector $v \in \mathbb{R}^n$ with $v_k \neq 0$ for some $1 \leq k \leq n$, let

$$\ell_{ik} = \frac{v_i}{v_k}, \quad i = k+1, \ldots, n, \qquad l_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \ell_{k+1,k} \\ \vdots \\ \ell_{n,k} \end{bmatrix},$$

and

$$M_k = I - l_k e_k^T = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & -\ell_{k+1,k} & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & -\ell_{n,k} & 0 & \cdots & 1 \end{bmatrix}. \tag{3.13}$$

Then one can verify that

$$M_k v = \begin{bmatrix} v_1 \\ \vdots \\ v_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}.$$

That is, M_k zeros $v_{k+1}, \ldots, v_n$ and keeps $v_1, \ldots, v_k$ unchanged. This unit lower triangular matrix M_k is called a Gaussian transformation, the vector l_k a Gauss vector, and the entries $\ell_{k+1}, \ldots, \ell_n$ the multipliers. Furthermore, one can verify that

$$M_k^{-1} = (I - l_k e_k^T)^{-1} = I + l_k e_k^T = \begin{bmatrix} 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & \cdots & \ell_{k+1,k} & 1 & \cdots & 0 \\ \vdots & & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \ell_{n,k} & 0 & \cdots & 1 \end{bmatrix}. \tag{3.14}$$

Given a nonsingular matrix $A \in \mathbb{R}^{n \times n}$, Gauss transformations $M_1, M_2, \ldots, M_{n-1}$ can usually be found such that $M_{n-1} \cdots M_2 M_1 A \equiv U$ is upper triangular. To see this, we denote $A^{(1)} = [a_{ij}^{(1)}] = A$. If $a_{11}^{(1)} \neq 0$, then the Gauss transformation

$$M_1 = I - l_1 e_1^T, \quad \text{where} \quad l_1 = \begin{bmatrix} 0 \\ \ell_{21} \\ \vdots \\ \ell_{n1} \end{bmatrix}, \qquad \ell_{i1} = \frac{a_{i1}^{(1)}}{a_{11}^{(1)}}, \quad i = 2, \ldots, n,$$

can be formed such that

$$A^{(2)} = M_1 A^{(1)} = \begin{bmatrix} a_{11}^{(2)} & a_{12}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{bmatrix},$$

where

$$a_{ij}^{(2)} = \begin{cases} a_{ij}^{(1)}, & \text{for } i = 1 \text{ and } j = 1, \ldots, n; \\ a_{ij}^{(1)} - \ell_{i1} * a_{1j}^{(1)}, & \text{for } i = 2, \ldots, n \text{ and } j = 2, \ldots, n. \end{cases}$$

The procedure can proceed to zero the entries in the second column below the diagonal, and so on. In general, at the k-th step, we are confronted with a matrix

$$
\begin{aligned}
A^{(k)} &= M_{k-1}\cdots M_2 M_1 A^{(1)} \\
&= \left[\begin{array}{cccc|ccc}
a_{11}^{(k)} & a_{12}^{(k)} & \cdots & a_{1,k-1}^{(k)} & a_{1k}^{(k)} & \cdots & a_{1n}^{(k)} \\
0 & a_{22}^{(k)} & \cdots & a_{2,k-1}^{(k)} & a_{2k}^{(k)} & \cdots & a_{2n}^{(k)} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & a_{k-1,k-1}^{(k)} & a_{k-1,k}^{(k)} & \cdots & a_{k-1,n}^{(k)} \\
\hline
0 & 0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\
\vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & a_{kn}^{(k)} & \cdots & a_{nn}^{(k)}
\end{array}\right] \\
&= \left[\begin{array}{cccc|ccc}
a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1,k-1}^{(1)} & a_{1k}^{(1)} & \cdots & a_{1n}^{(1)} \\
0 & a_{22}^{(2)} & \cdots & a_{2,k-1}^{(2)} & a_{2k}^{(2)} & \cdots & a_{2n}^{(2)} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\
0 & 0 & \cdots & a_{k-1,k-1}^{(k)} & a_{k-1,k}^{(k)} & \cdots & a_{k-1,n}^{(k)} \\
\hline
0 & 0 & \cdots & 0 & a_{kk}^{(k)} & \cdots & a_{kn}^{(k)} \\
\vdots & \vdots & & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 0 & a_{kn}^{(k)} & \cdots & a_{nn}^{(k)}
\end{array}\right]
\end{aligned}
$$

If the pivot $a_{kk}^{(k)} \neq 0$, then the multipliers

$$\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}, \quad i = k+1, \ldots, n, \tag{3.15}$$

can be computed and the Gaussian transformation

$$M_k = I - l_k e_k^T, \quad \text{where} \quad l_k = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \ell_{k+1,k} \\ \vdots \\ \ell_{nk} \end{bmatrix},$$

can be applied to the left of $A^{(k)}$ to obtain

$$A^{(k+1)} = M_k A^{(k)} = \left[\begin{array}{cccc|cccc}
a_{11}^{(k+1)} & a_{12}^{(k+1)} & \cdots & a_{1,k-1}^{(k+1)} & a_{1k}^{(k+1)} & \cdots & \cdots & a_{1n}^{(k+1)} \\
0 & a_{22}^{(k+1)} & \cdots & a_{2,k-1}^{(k+1)} & a_{2k}^{(k+1)} & \cdots & \cdots & a_{2n}^{(k+1)} \\
\vdots & \vdots & \ddots & \vdots & \vdots & & \cdots & \vdots \\
0 & 0 & \cdots & a_{k-1,k-1}^{(k+1)} & a_{k-1,k}^{(k+1)} & \cdots & \cdots & a_{k-1,n}^{(k+1)} \\
\hline
0 & 0 & \cdots & 0 & a_{kk}^{(k+1)} & \cdots & \cdots & a_{kn}^{(k+1)} \\
\vdots & \vdots & & \vdots & 0 & \ddots & & \vdots \\
\vdots & \vdots & & \vdots & \vdots & & \ddots & \vdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & \cdots & a_{nn}^{(k+1)}
\end{array}\right],$$

in which

$$a_{ij}^{(k+1)} = \begin{cases} a_{ij}^{(k)}, & \text{for } i = 1, \dots, k,\ j = 1, \dots, n; \\ 0, & \text{for } i = k+1, \dots, n,\ j = k; \\ a_{ij}^{(k)} - \ell_{ik} a_{kj}^{(k)}, & \text{for } i = k+1, \dots, n,\ j = k+1, \dots, n. \end{cases} \tag{3.16}$$

Upon the completion,

$$U \equiv A^{(n)} = M_{n-1} \cdots M_2 M_1 A$$

is upper triangular. Hence

$$A = M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} U \equiv LU, \tag{3.17}$$

where

$$\begin{aligned} L \equiv M_1^{-1} M_2^{-1} \cdots M_{n-1}^{-1} &= (I - l_1 e_1^T)^{-1} (I - l_2 e_2^T)^{-1} \cdots (I - l_{n-1} e_{n-1}^T)^{-1} \\ &= (I + l_1 e_1^T)(I + l_2 e_2^T) \cdots (I + l_{n-1} e_{n-1}^T) \\ &= I + l_1 e_1^T + l_2 e_2^T + \cdots + l_{n-1} e_{n-1}^T \\ &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix} \end{aligned} \tag{3.18}$$

is unit lower triangular. This matrix factorization is called the LU-factorization of A.

Equations (3.15) and (3.16) provide formulations for computing the LU-factorization. Alternatively, the algorithm can be derived directly by componentwise comparison in $A = LU$. Suppose that the factors L and U are written as

$$L = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \ell_{21} & 1 & 0 & \cdots & 0 \\ \ell_{31} & \ell_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \cdots & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{bmatrix}.$$

By comparing the components in $A = LU$ as

$$\begin{aligned} &\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{bmatrix} \\ &= \begin{bmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ \ell_{21} u_{11} & \ell_{21} u_{12} + u_{22} & \ell_{21} u_{13} + \ell_{22} u_{23} & \cdots & l_{21} u_{1n} + u_{2n} \\ \ell_{31} u_{11} & \ell_{31} u_{12} + \ell_{32} u_{22} & \ell_{31} u_{13} + \ell_{32} u_{23} + u_{33} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \ell_{n1} u_{11} & \ell_{n1} u_{12} + \ell_{n2} u_{2n} & \ell_{n1} u_{13} + \ell_{n2} u_{23} + \ell_{n3} u_{33} & \cdots & \ell_{n1} u_{1n} + \cdots + u_{nn} \end{bmatrix}, \end{aligned}$$

we see that $u_{11} = a_{11}, u_{12} = a_{12}, \ldots, u_{1n} = a_{1n}$. Once u_{11} is determined, $\ell_{21} = a_{21}/u_{11}, \ell_{31} = a_{31}/u_{11}, \ldots, \ell_{n1} = a_{n1}/u_{11}$ can be computed. Next, $a_{22} = \ell_{21}u_{12} + u_{22}$. Since ℓ_{21} and u_{12} are known at this stage, $u_{22} = a_{22} - \ell_{21}u_{12}$ is computed. Once u_{22} is determined, $u_{22}, \ldots, u_{2n}$, and $\ell_{22}, \ldots, \ell_{n2}$ can be obtained. Then we proceed to compute the third row of U and the third column of L, and so on.

The principal of the procedure follows from the identity

$$a_{ij} = \sum_{p=1}^{\min(i,j)} \ell_{ip}u_{pj}. \tag{3.19}$$

Suppose that after $k-1$ steps we have computed the first $k-1$ columns of L and the first $k-1$ rows of U. At the k-th step, we can first determine u_{kk} since

$$a_{kk} = \sum_{p=1}^{k} \ell_{kp}u_{pk} = \sum_{p=1}^{k-1} \ell_{kp}u_{pk} + u_{kk},$$

and ℓ_{kp}, u_{pk}, $p = 1, 2, \ldots, k-1$, are known. The computation is done by performing

$$u_{kk} = a_{kk} - \sum_{p=1}^{k-1} \ell_{kp}u_{pk}. \tag{3.20}$$

Once u_{kk} is determined and $u_{kk} \neq 0$, we can compute the k-th column of L and the k-th row of U. By using the identities

$$a_{ik} = \sum_{p=1}^{k-1} l_{ip}u_{pk} + l_{ik}u_{kk}, \qquad i = k+1, \ldots, n,$$

and

$$a_{kj} = \sum_{p=1}^{k-1} l_{kp}u_{pj} + u_{kj}, \qquad j = k+1, \ldots, n,$$

we can compute

$$\ell_{ik} = \left(a_{ik} - \sum_{p=1}^{k-1} \ell_{ip}u_{pk} \right) \Big/ u_{kk}, \qquad i = k+1, \ldots, n, \tag{3.21}$$

and

$$u_{kj} = a_{kj} - \sum_{p=1}^{k-1} \ell_{kp}u_{pj}, \qquad j = k+1, \ldots, n. \tag{3.22}$$

Formulations (3.20), (3.21), and (3.22) together give the Doolittle LU-factorization algorithm.

Algorithm 3.6 (Doolittle LU-Factorization) *Given a nonsingular square matrix $A \in \mathbb{R}^{n\times n}$, this algorithm computes a unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$.*

```
Initialize L = 0, U = 0
for k = 1, ... , n do
   L(k, k) = 1
   t = 0
   for p = 1, ... , k - 1 do
      t = t + L(k, p) * U(p.k)
   end for
   U(k, k) = A(k, k) - t
   for i = k + 1, ... , n do
      t = 0
      for p = 1, ... , k - 1 do
         t = t + L(i, p) * U(p, k)
      end for
      L(i, k) = (A(i, k) - t)/U(k, k)
   end for
   for j = k + 1, ... , n do
      t = 0
      for p = 1, ... , k - 1 do
         t = t + L(k, p) * U(p, j)
      end for
      U(k, j) = A(k, j) - t
   end for
end for
```

For practical implementation, the storage of A can be overwritten by the entries of L and U. The upper triangular components of U overwrite upper triangular part of A, and the multipliers (the strictly lower triangular part of L) can be stored in the strictly lower triangualr part of A. The diagonal part of L needs not to be stored since they are all 1. An outer-product version of the LU-factorization is as follows.

Algorithm 3.7 (LU Factorization) *Given a nonsingular square matrix $A \in \mathbb{R}^{n\times n}$, this algorithm computes a unit lower triangular matrix L and an upper triangular matrix U such that $A = LU$. The matrix A is overwritten by L and U.*

```
for k = 1, ... , n - 1 do
   for i = k + 1, ... , n do
      A(i, k) = A(i, k)/A(k, k)
      for j = k + 1, ... , n do
         A(i, j) = A(i, j) - A(i, k) ⋆ A(k, j)
      end for
   end for
end for
```

This algorithm requires

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

flops.

The LU-factorization together with forward substitution and back substitution is a standard procedure for solving the linear system of equations. This approach requires $\frac{2}{3}n^3+O(n^2)$ flops. The computation of the factorization dominates the cost.

3.2.3 Existence and Uniqueness of LU Factorization

Definition 3.1 (Leading principal minor) *Let A be an $n \times n$ matrix. The upper left $k \times k$ submatrix, denoted as*

$$A_k = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1k} \\ a_{21} & a_{22} & \cdots & a_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{k1} & a_{k2} & \cdots & a_{kk} \end{bmatrix},$$

is called the leading $k \times k$ principal submtrix, and the determinant of A_k, $\det(A_k)$, is called the leading principal minor.

Theorem 3.1 *If all leading principal minor of $A \in \mathbb{R}^{n\times n}$ are nonzero, that is, all leading principal submatrices are nonsingular, then A has an LU-factorization.*

Proof: Proof by mathematical induction. When $n = 1$, $A_1 = [a_{11}]$ is nonsingular, then $a_{11} \neq 0$. Let $L_1 = [1]$ and $U_1 = [a_{11}]$. Then $A_1 = L_1U_1$. The theorem holds.

Assume that the leading principal submatrices $A_1, \ldots, A_k$ are nonsingular and A_k has an LU-factorization $A_k = L_kU_k$, where L_k is unit lower triangular and U_k is upper triangular. We shall show that there exist an unit lower triangular matrix L_{k+1} and an upper triangular matrix U_{k+1} such that $A_{k+1} = L_{k+1}U_{k+1}$.

Write

$$A_{k+1} = \begin{bmatrix} A_k & v_k \\ w_k^T & a_{k+1,k+1} \end{bmatrix}, \quad \text{where} \quad v_k = \begin{bmatrix} a_{1,k+1} \\ a_{2,k+1} \\ \vdots \\ a_{k,k+1} \end{bmatrix} \quad \text{and} \quad w_k = \begin{bmatrix} a_{k+1,1} \\ a_{k+1,2} \\ \vdots \\ a_{k+1,k} \end{bmatrix}.$$

Since A_k is nonsingular, both L_k and U_k are nonsingular. Therefore the system $L_ky_k = v_k$ has a unique solution $y_k \in \mathbb{R}^k$, and $z^tU_k = w_k^T$ has a unique solution $z_k \in \mathbb{R}^k$. Let

$$L_{k+1} = \begin{bmatrix} L_k & 0 \\ z_k^T & 1 \end{bmatrix} \quad \text{and} \quad U_{k+1} = \begin{bmatrix} U_k & y_k \\ 0 & a_{k+1,k+1} - z_k^T y_k \end{bmatrix}.$$

Then L_{k+1} is unit lower triangular, U_{k+1} is upper triangular, and

$$L_{k+1}U_{k+1} = \begin{bmatrix} L_kU_k & L_ky_k \\ z_k^TU_k & z_k^Ty_k + a_{k+1,k+1} - z_k^Ty_k \end{bmatrix} = \begin{bmatrix} A_k & v_k \\ w_k^T & a_{k+1,k+1} \end{bmatrix} = A_{k+1}.$$

This proves the theorem. ∎

Theorem 3.2 *If A is nonsingular and the LU factorization exists, then the LU factorization is unique and* $\det(A) = u_{11}\cdots u_{nn}$.

Proof: Suppose both $A = L_1U_1$ and $A = L_2U_2$ are LU factorizations. Since A is nonsingular, L_1, U_1, L_2, U_2 are all nonsingular, and

$$A = L_1U_1 = L_2U_2 \Longrightarrow L_2^{-1}L_1 = U_2U_1^{-1}.$$

Since L_1 and L_2 are unit lower triangular which implies $L_2^{-1}L_1$ is unit lower triangular, and U_1 and U_2 are upper triangular which implies $U_2U_1^{-1}$ is upper triangular, it follows that $L_2^{-1}L_1 = I = U_2U_1^{-1}$. Therefore $L_1 = L_2$ and $U_1 = U_2$. ∎

3.3 Pivoting

It is clear from the formulations, e.g., (3.20), (3.21), and (3.22), presented in the previous section that Gaussian elimination may fail or may not give satisfactory results if a zero or small pivot entry encountered during the procedure. Therefore good algorithms for solving systems of linear equations must incorporate the interchanging of equations in a system when the circumstances require it. In this section we shall illustrate some of these situations and present the most commonly used pivoting strategies.

3.3.1 The Need for Pivoting

The Gaussian elimination algorithm 3.5 could fail on systems that are in fact easy to solve. For example, the algorithm would fail at the first step on the input system

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

since the first pivot element is zero. But if we interchange the rows, the system

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

becomes trivial to solve.

Another difficulty will arise when a pivot element encountered is a small number ε different from zero. For example, the simple Gaussian elimination algorithm would produce relatively large error on the system

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix},$$

where $\varepsilon < \epsilon_M$. Algorithm 3.5 would compute

$$\begin{bmatrix} \varepsilon & 1 \\ 0 & 1-\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 2-\frac{1}{\varepsilon} \end{bmatrix} \Longrightarrow \begin{bmatrix} \varepsilon & 1 \\ 0 & -\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -\frac{1}{\varepsilon} \end{bmatrix},$$

since in the computer, if ε is small enough, $1 - \frac{1}{\varepsilon}$ will be computed to be the same as $-\frac{1}{\varepsilon}$. Likewise, $2 - \frac{1}{\varepsilon}$ will be computed to be the same as $-\frac{1}{\varepsilon}$. Hence, under these circumstances, back substitution would produce

$$x_2 = \frac{-\frac{1}{\varepsilon}}{-\frac{1}{\varepsilon}} = 1 \quad \text{and} \quad x_1 = \frac{1-1}{\varepsilon} = 0.$$

The computed solution is accurate for x_2 but is extremely inaccurate for x_1 since

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

But actually $x_1 = x_2 = 1$ would be a much better solution since

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1+\varepsilon \\ 2 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 2 \end{bmatrix}.$$

On the other hand, it would produce a much better result if we interchange the rows since Gaussian elimination would compute

$$\begin{bmatrix} 1 & 1 \\ \varepsilon & 1 \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & 1 \\ 0 & 1-\varepsilon \end{bmatrix} \begin{bmatrix} x_2 \\ x_1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1-2\varepsilon \end{bmatrix},$$

and the back substitution will give

$$x_1 = \frac{1-2\epsilon}{1-\varepsilon} \approx 1 \quad \text{and} \quad x_2 = 2 - x_1 \approx 2 - 1 = 1.$$

The strategy of interchange rows/columns as described above is called "pivoting". And we have seen that pivoting is necessary for some systems.

3.3.2 Partial Pivoting and Complete Pivoting

As we have observed in the previous subsection that if $a_{kk}^{(k)}$ is small in magnitude compared to $a_{ik}^{(k)}$, $i = k+1, \ldots, n$, then the multipliers

$$\ell_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$$

will have magnitude much larger than 1. Roundoff introduced in computing

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \ell_{ik} * a_{kj}^{(k)}, \quad i = k+1, \ldots, n, \quad j = k+1, \ldots, n,$$

will be large. Also when performing the back substitution for

$$x_k = \left(\widetilde{b}_k - \sum_{j=k+1}^{n} a_{kj}^{(k)} \right) \Big/ a_{kk}^{(k)},$$

where $\widetilde{b}_k$ is the modified b_k by algorithm 3.5, any error in the numerator will be dramatically increased when dividing by a small $a_{kk}^{(k)}$.

To ensure that no large entries appear in the computed triangular factors, one can choose a pivot element to be the largest entry among $|a_{kk}^{(k)}|, \ldots , |a_{nk}^{(k)}|$. Then permute this row with the k-th row. The multipliers ℓ_{ik} are computed after the permutation.

To explain the pivoting techniques in matrix language, let $P_1, \ldots , P_{k-1}$ be the permutations chosen and $M_1, \ldots M_{k-1}$ denote the Gaussian transformations performed in the first $k-1$ steps. At the k-th step, a permutation matrix P_k is chosen so that

$$|(P_k M_{k-1} \cdots M_1 P_1 A)_{kk}| = \max_{k \le i \le n} |(M_{k-1} \cdots M_1 P_1 A)_{ik}| .$$

This row interchange strategy is called partial pivoting. As a consequence, no multipliers is greater than one in magnitude, that is, $|\ell_{ij}| \le 1$ for $i = 1, \ldots , n$, $j = 1, \ldots , i$. Upon completion, we obtain an upper triangular matrix

$$U \equiv M_{n-1} P_{n-1} \cdots M_1 P_1 A. \tag{3.23}$$

Since any permutation matrix P_k is symmetric and $P_k^T P_k = P_k^2 = I$, a manipulation of (3.23) reveals that

$$M_{n-1} P_{n-1} \cdots M_2 P_2 M_1 P_2 \cdots P_{n-1} P_{n-1} \cdots P_1 A = U,$$

therefore,

$$P_{n-1} \cdots P_1 A = (M_{n-1} P_{n-1} \cdots M_2 P_2 M_1 P_2 \cdots P_{n-1})^{-1} U.$$

In summary, Gaussian elimination with partial pivoting leads to the LU factorization

$$PA = LU, \tag{3.24}$$

where

$$P = P_{n-1} \cdots P_1$$

is a permutation matrix, and

$$L \equiv (M_{n-1} P_{n-1} \cdots M_2 P_2 M_1 P_2 \cdots P_{n-1})^{-1}$$

is unit lower triangular.

While in the partial pivoting algorithm, the k-th pivot is determined by scanning the current k-th subcolumn $A^{(k)}(k : n, k)$, another pivoting strategy called complete pivoting searches for the largest entry in magnitude in the current submatrix $A^{(k)}(k : n, k : n)$ and permutes to the (k,k) position. That is, at the k-th step two permutation matrices P_k and Q_k are determined so that

$$\left|(P_k A^{(k)} Q_k)_{kk}\right| = \max_{k \le i,j \le n} \left|(A^{(k)})_{ij}\right|.$$

Gaussian elimination with complete pivoting leads to the LU factorization

$$PAQ = LU, \tag{3.25}$$

where P, Q are permutation matrices, L is unit lower triangular, and U is upper triangular.

In practical implementation, no permutation matrix should ever be explicitly formed. The permutations can be efficiently represented by an integer n-vector. For example, an integer array p is usually used for partial pivoting to keep the indices of rows such that $p(k)$ stores the row index which interchanges with row k. In other words, (3.11) becomes

$$a_{p(i),j}^{(k+1)} = \begin{cases} a_{p(i),j}^{(k)}, & \text{for } i = 1, \ldots, k, \text{ and } j = 1, \ldots, n; \\ 0, & \text{for } i = k+1, \ldots, n, \text{ and } j = k; \\ a_{p(i),j}^{(k)} - \frac{a_{p(i),k}^{(k)}}{a_{p(k),k}^{(k)}} * a_{p(k),j}^{(k)}, & \text{for } i = k+1, \ldots, n, \text{ and } j = k+1, \ldots, n. \end{cases} \tag{3.26}$$

Two integer arrays will be required to store the permutation information for complete pivoting.

It should be noted that although there is no floating-point arithmetic involved when a permutation is applied to a matrix, only row/columns exchanged, it often causes irregular data movement and significant computational overhead. The LU factorization algorithm 3.7 can be modified to incorporate with partial pivoting as shown in the following. The one for complete pivoting is similar.

Algorithm 3.8 (LU-factorization with Partial Pivoting) *Given a nonsingular square matrix $A \in \mathbb{R}^{n\times n}$, this algorithm finds an appropriate permutation matrix P, and computes a unit lower triangular matrix L and an upper triangular matrix U such that $PA = LU$. The matrix A is overwritten by L and U, and the matrix P is not formed. An integer array p is instead used for storing the row/column indices.*

```
for i = 1, ..., n do
  p(i) = i        % initialize the pivoting vector
end for
for k = 1, ..., n-1 do
  m = k
  for i = k+1, ..., n do
    if |A(p(m), k)| < |A(p(i), k)| then
      m = i
    end if
  end for
  ℓ = p(k)
  p(k) = p(m)
  p(m) = ℓ
  for i = k+1, ..., n do
    A(p(i), k) = A(p(i), k)/A(p(k), k)
    for j = k+1, ..., n do
      A(p(i), j) = A(p(i), j) - A(p(i), k) * A(p(k), j)
    end for
  end for
end for
```

Since the Gaussian elimination with partial pivoting produces the factorization (3.24), the linear system problem should comply accordingly.

$$Ax = b \Longrightarrow PAx = Pb \Longrightarrow LUx = Pb.$$

Hence the permutation needs to be applied to the right-hand-side vetor b, and the forward substitution should be modified to

Algorithm 3.9 (Permuted Forward Substitution) *Given an unit lower triangular matrix L, a permutation matrix P, and an n-vector b, this algorithm solves the triangular system $Ly = Pb$. The matrix L is stored in the strictly lower triangular part of matrix A, and the premutation matrix P is represented by an integer array p.*

```
y(p(1)) = b(p(1))/A(p(1), 1)
for i = 1, ..., n do
  t = 0
  for j = 1, ..., i-1 do
    t = t + A(p(i), j) * y(p(j))
  end for
  y(p(i)) = b(p(i)) - t
end for
```

3.3.3 Scaled Row Pivoting

In the following example, one will see that it is not actually the smallness of the pivot element that is causing the trouble. Rather, it is the smallness of the pivot element relative to the other elements in its row. Consider

$$\begin{bmatrix} 1 & \frac{1}{\varepsilon} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\varepsilon} \\ 2 \end{bmatrix},$$

where $\varepsilon << \epsilon_M$. The algorithm of Gaussian elimination with or without pivoting would give

$$\begin{bmatrix} 1 & \frac{1}{\varepsilon} \\ 0 & 1-\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\varepsilon} \\ 2-\frac{1}{\varepsilon} \end{bmatrix} \Longrightarrow \begin{bmatrix} 1 & \frac{1}{\varepsilon} \\ 0 & -\frac{1}{\varepsilon} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \frac{1}{\varepsilon} \\ -\frac{1}{\varepsilon} \end{bmatrix},$$

and the back substitution will result

$$x_2 = \frac{\frac{-1}{\varepsilon}}{\frac{-1}{\varepsilon}} = 1, \quad \text{and} \quad x_1 = \frac{1}{\varepsilon} - \frac{1}{\varepsilon} \cdot x_2 = 0.$$

Again, the solution is accurate for x_2 but is extremely inaccurate for x_1 since

$$\begin{bmatrix} 1 & \frac{1}{\varepsilon} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 & \frac{1}{\varepsilon} \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{\varepsilon} \\ 1 \end{bmatrix}.$$

What causes the problem? One observes that the pivot element, 1, is too small relative to the other entries, $\frac{1}{\varepsilon}$, in the pivot row in this example.

Another pivoting strategy called scaled row pivoting is introduced. We begin the factorization by computing the scale of each row

$$s_i = \max_{1 \le j \le n} |a_{ij}|, \quad i = 1, \ldots, n. \tag{3.27}$$

These values are stored in an array s in the algorithm. To keep track of the row indices, we begin by setting the permutation vector, also called pivoting vector, $p = [1, 2, \ldots, n]^T$. In each Gaussian elimination step k, the algorithm scans the numbers

$$\frac{|a_{p(i),k}|}{s_{p(i)}}, \quad i = k, \ldots, n,$$

looking for a maximal one. Suppose m is the row index of the maximal ratio, we exchange indices $p(k)$ and $p(m)$, and use the new $a_{p(k),k}$ as the pivot element and row $p(k)$ as the pivot row to zero entries $a_{p(k+1),k}$ throught $a_{p(n),k}$, and modify the lower right submatrix, that is,

$$a_{p(i),j} \leftarrow a_{p(i),j} - \frac{a_{p(i),k}}{a_{p(k),k}} * a_{p(k),j} \qquad j = k+1, \ldots, n, \quad i = k+1, \ldots, n.$$

This procedure is repeated for $k = 1, \ldots, n-1$.

We use the following example to illustrate how the Gaussian elimination with scaled row pivoting works.

Example 3.2 *Consider solving the linear system of equations in Example 3.1 using Gaussian elimination with scaled row pivoting.*

Sol: Initially, we have

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 12 & -8 & 6 & 10 \\ 3 & -13 & 9 & 3 \\ -6 & 4 & 1 & -18 \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}, \quad s = \begin{bmatrix} 6 \\ 12 \\ 13 \\ 18 \end{bmatrix}.$$

Step through the Gaussian elimination with scaled row pivoting we have

$k = 1:$ Scan $\frac{6}{6}, \frac{12}{12}, \frac{3}{13}, \frac{6}{18}$, choose row 1 as the pivot row, vector p unchanged.

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 2 & -4 & 2 & 2 \\ \frac{1}{2} & -12 & 8 & 1 \\ -1 & 2 & 3 & -14 \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}.$$

$k = 2:$ Scan $\frac{4}{12}, \frac{12}{13}, \frac{2}{18}$, choose row 3 as the pivot row.

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 2 & \frac{1}{3} & -\frac{2}{3} & \frac{5}{3} \\ \frac{1}{2} & -12 & 8 & 1 \\ -1 & -\frac{1}{6} & \frac{13}{3} & -\frac{83}{6} \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 4 \end{bmatrix}.$$

$k = 3:$ Scan $\frac{2/3}{12}, \frac{13/3}{18}$, choose row 4 as the pivot row.

$$A = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 2 & \frac{1}{3} & -\frac{2}{13} & -\frac{18}{39} \\ \frac{1}{2} & -12 & 8 & 1 \\ -1 & -\frac{1}{6} & \frac{13}{3} & -\frac{83}{6} \end{bmatrix}, \quad p = \begin{bmatrix} 1 \\ 3 \\ 4 \\ 2 \end{bmatrix}.$$

Note that the entries of A are not actually moved and some positions of A are overwritten by the multipliers. If we let

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

denote the permutation matrix and

$$L = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ -1 & -\frac{1}{6} & 1 & 0 \\ 2 & \frac{1}{3} & -\frac{2}{13} & 1 \end{bmatrix}, \qquad U = \begin{bmatrix} 6 & -2 & 2 & 4 \\ 0 & -12 & 8 & 1 \\ 0 & 0 & \frac{13}{3} & -\frac{83}{6} \\ 0 & 0 & 0 & -\frac{18}{39} \end{bmatrix},$$

then one can verify that $PA = LU$. ■

Algorithm 3.10 (Gaussian Elimination with Scaled-Row Pivoting) *Let $A \in \mathbb{R}^{n\times n}$ be a nonsingular matrix. This algorithm performs the Gaussian elimination with scaled row pivoting to produce the factorization $PA = LU$. The matrix A is modified and the multipliers are stored in A. An integer array p is used to store the permutation information.*

for $i = 1, \ldots, n$ **do**
  $p(i) = i$
  $s(i) = \max_{1\le j\le n} |a_{ij}|$
end for
for $k = 1, \ldots, n-1$ **do**
  $m = k$
  for $i = k+1, \ldots, n$ **do**
    if $|A(p(m), k)|/s(p(m)) < |A(p(i), k)|/s(p(i))$ **then**
      $m = i$
    end if
  end for
  $\ell = p(k)$
  $p(k) = p(m)$
  $p(m) = \ell$
  for $i = k+1, \ldots, n$ **do**
    $A(p(i), k) = A(p(i), k)/A(p(k), k)$
    for $j = k+1, \ldots, n$ **do**
      $A(p(i), j) = A(p(i), j) - A(p(i), k) * A(p(k), j)$
    end for
  end for
end for

The number of arithmetic operations in the factorization can be approximately estimated as

$$\sum_{k=1}^{n-1}[(n-k) + 2(n-k)(n-k+1)] = 2\sum_{k=1}^{n-1} k^2 + 3\sum_{k=1}^{n-1} k + (n-1)$$
$$= \frac{2}{3}n^3 + \frac{1}{2}n^2 - \frac{1}{6}n - 1 \approx \frac{2}{3}n^3 + \frac{1}{2}n^2.$$

For large n, the term $\frac{2}{3}n^3$ is the dominant one.

Notice that the multipliers are being stored in A at the locations where 0's would have been created by the elimination process. That is, factors L and U overwrite A. The original matrix A should be saved in a separate array if it will be need again in later computation.

Once the factorization has been carried out on A, we apply the forward substitution algorithm using the final permutation array p and the multipliers that were determined by the factorization process and were stored in A. Finally we carry out the back substitution algorithm to solve for x.

Algorithm 3.11 (Updating & Back Substitution) *Given $b \in \mathbb{R}^n$, an integer array p, and $A \in \mathbb{R}^{n\times n}$, where A is the resulting matrix after Gaussian elimination with scaled row pivoting, this algorithm solves $Ax = b$ for x.*

```
for i = 1, ..., n − 1 do
  for j = i + 1, ..., n do
    b(p(j)) = b(p(j)) − A(p(j), i) * b(p(i))
  end for
end for
for i = n, n − 1, ..., 1 do
  t = 0
  for j = i + 1, ..., n do
    t = t + A(p(i), j) * x(j)
  end for
  x(i) = (b(p(i)) − t)/A(p(i), i)
end for
```

The number of floating-point operations involved in the solution phase is about

$$\sum_{i=1}^{n-1} 2(n-i) + \sum_{i=1}^{n}[2(n-i)+2] = 2\sum_{k=1}^{n-1} k + 2\sum_{k=1}^{n} k = 2n^2.$$

In summary, it costs $\frac{2}{3}n^3 + O(n^2)$ flops to solve the linear system of equations $Ax = b$ using Gaussian elimination with scaled row pivoting. Note that it would take $\frac{4}{3}n^3 + O(n^2)$ just to compute A^{-1}. Be advised that A^{-1} should not be computed when solving a linear system $Ax = b$, but rather x should be solved for directly.

3.4 Some Special Linear Systems

It is a basic tenet of numerical analysis that structure should be exploited whenever solving a problem. Algorithms for general matrix problems can be streamlined in the presence of such properties as symmetry, definiteness, and sparsity. This is the central theme of this section, where our principal aim is to discuss special algorithms for computing special variants of the LU factorization.

3.4.1 Symmetric Positive Definite System and Cholesky Factorization

An $n \times n$ matrix A is positive definite if $x^T Ax > 0$, for all $x \in \mathbb{R}^n$, $x \neq 0$. If A is both symmetric and positive definite (*spd*), then we can derive a stable LU factorization called the Choleseky factorization.

Lemma 3.1 *If $A \in \mathbb{R}^{n\times n}$ is positive definite, then A is nonsingular and $a_{ii} > 0$ for $i = 1, \ldots, n$.*

Proof: Suppose A is singular. Then there exists $x \in \mathbb{R}^n$ and $x \neq 0$ such that $Ax = 0$. This implies $x^T Ax = 0$, which contradicts the fact that A is positive definite. Therefore A is nonsingular. It is also easy to see that

$$a_{ii} = e_i^T Ae_i > 0,$$

where e_i is the i-th column of the $n \times n$ identify matrix. ■

Lemma 3.2 *If $A \in \mathbb{R}^{n\times n}$ is positive definite, then all leading principal submatrices of A are nonsingular.*

Proof: For $1 \le k \le n$, let $z_k = [x_1, \dots, x_k]^T \in \mathbb{R}^k$ and $x = [x_1, \dots, x_k, 0, \dots, 0]^T \in \mathbb{R}^n$, where $x_1, \dots, x_k \in \mathbb{R}$ are not all zero. Since A is positive definite,

$$z_k^T A_k z_k = x^T A x > 0,$$

where A_k is the $k \times k$ leading principal submatrix of A. This shows that A_k are also positive definite, hence A_k are nonsingular. ■

Theorem 3.3 *If $A \in \mathbb{R}^{n\times n}$ is symmetric positive definite, then there exists a unique lower triangular matrix $G \in \mathbb{R}^{n\times n}$ with positive diagonal entries such that A has the factorization*

$$A = GG^T. \tag{3.28}$$

Proof: From Lemma 3.2, all leading principal submatrices of a positive definite matrix are nonsingular, hence A has the LU factorization $A = LU$, where L is unit lower triangular and U is upper triangular. Since A is symmetric,

$$LU = A = A^T = U^T L^T \quad \Longrightarrow \quad U(L^T)^{-1} = L^{-1}U^T.$$

Note that $U(L^T)^{-1}$ is upper triangular and $L^{-1}U^T$ is lower triangular, this forces $U(L^T)^{-1}$ to be a diagonal matrix, say, $U(L^T)^{-1} = D$. Then $U = DL^T$. Hence

$$A = LDL^T.$$

Since A is positive definite,

$$x^T A x > 0 \quad \Longrightarrow \quad x^T LDL^T x = (L^T x)^T D (L^T x) > 0.$$

This means D is also positive definite, and hence $d_{ii} > 0$. Thus $D^{1/2}$ is well-defined and we have

$$A = LDL^T = LD^{1/2}D^{1/2}L^T \equiv GG^T,$$

where $G \equiv LD^{1/2}$. Since the LU factorization is unique, G is unique. ■

The factorization (3.28) is referred to as the Cholesky factorization and G is referred to as the Cholesky factor or Cholesky triangle, when A is both symmetric and positive definite.

To derive an algorithm for computing the Cholesky factorization (3.28) we assume that the first $k-1$ columns of G have been determined after $k-1$ steps. By componentwise comparison with equation (3.28), one has

$$a_{kk} = \sum_{j=1}^{k} g_{kj}^2,$$

which gives

$$g_{kk}^2 = a_{kk} - \sum_{j=1}^{k-1} g_{kj}^2. \tag{3.29}$$

Moreover,

$$a_{ik} = \sum_{j=1}^{k} g_{ij} * g_{kj}, \qquad i = k+1, \ldots, n,$$

hence the k-th column of G can be computed by

$$g_{ik} = \left(a_{ik} - \sum_{j=1}^{k-1} g_{ij} * g_{kj} \right) \Big/ g_{kk}, \qquad i = k+1, \ldots, n. \tag{3.30}$$

Equations (3.29) and (3.30) lay out the formulations for the Cholesky factorization algorithm.

Algorithm 3.12 (Cholesy Factorization) *Given an $n \times n$ symmetric positive definite matrix A, this algorithm computes the Cholesky factorization $A = GG^T$.*

Initialize $G = 0$
for $k = 1, \ldots, n$ **do**
$$G(k,k) = \sqrt{A(k,k) - \sum_{j=1}^{k-1} G(k,j) * G(k,j)}$$
for $i = k+1, \ldots, n$ **do**
$$G(i,k) = \left(A(i,k) - \sum_{j=1}^{k-1} G(i,j) * G(k,j) \right) \Big/ G(k,k)$$
end for
end for

In addition to n square root operations, there are approximately

$$\sum_{k=1}^{n} [2k - 1 + 2k(n-k)] = \frac{1}{3}n^3 + n^2 - \frac{1}{3}n$$

floating-point arithmetic required by the algorithm. In practical implementation, the lower triangular part of A can be overwritten by the Cholesy factor G.

It is clear from (3.29) that

$$a_{kk} = \sum_{j=1}^{k} g_{kj}^2 \geq g_{ki}^2, \qquad i = 1, \ldots, k.$$

Hence

$$|g_{ki}| \leq \sqrt{a_{kk}}, \qquad \forall\, i = 1, \ldots, k. \tag{3.31}$$

This means that the entries of G are nicely bounded and will not grow large relative to the diagonals of A. Therefore, pivoting will not be required, and the Cholesky factorization is a stable numerical algorithm.

3.4.2 Diagonally Dominant Systems

A matrix $A \in \mathbb{R}^{n\times n}$ is said to be diagonally dominant if

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|.$$

We shall show that Guassian elimination without pivoting can be safely used for this class of matrices.

Lemma 3.3 *If $A \in \mathbb{R}^{n\times n}$ is diagonally dominant, then A is nonsingular.*

Proof: Suppose A is singular. Then there exists $x \in \mathbb{R}^n$, $x \neq 0$ such that $Ax = 0$. Let k be the integer index such that

$$|x_k| = \max_{1\leq i\leq n} |x_i| \quad \Longrightarrow \quad \frac{|x_i|}{|x_k|} < 1, \quad \forall\, i \neq k.$$

Since $Ax = 0$, for the fixed k, we have

$$\sum_{j=1}^{n} a_{kj}x_j = 0 \quad \Longrightarrow \quad a_{kk}x_k = -\sum_{j=1,j\neq k}^{n} a_{kj}x_j \quad \Longrightarrow \quad |a_{kk}||x_k| \leq \sum_{j=1,j\neq k}^{n} |a_{kj}||x_j|,$$

which implies

$$|a_{kk}| \leq \sum_{j=1,j\neq k}^{n} |a_{kj}|\frac{|x_j|}{|x_k|} < \sum_{j=1,j\neq k}^{n} |a_{kj}|.$$

But this contradicts the assumption that A is diagonally dominant. Therefore A must be nonsingular. ■

Theorem 3.4 *Gaussian elimination without pivoting preserve the diagonal dominance of a matrix.*

Proof: Let $A \in \mathbb{R}^{n\times n}$ be a diagonally dominant matrix and $A^{(2)} = [a_{ij}^{(2)}]$ is the result of applying one step of Gaussian elimination to $A^{(1)} = A$ without any pivoting strategy.

After one step of Gaussian elimination, $a_{i1}^{(2)} = 0$ for $i = 2, \dots, n$, and the first row is unchanged. Therefore, the property

$$a_{11}^{(2)} > \sum_{j=2}^{n} |a_{1j}^{(2)}|$$

is preserved, and all we need to show is that

$$a_{ii}^{(2)} > \sum_{j=2,j\neq i}^{n} |a_{ij}^{(2)}|, \quad \text{for} \quad i = 2, \dots, n.$$

Using the Gaussian elimination formula (3.11), we have

$$\begin{aligned}
|a_{ii}^{(2)}| &= \left| a_{ii}^{(1)} - \frac{a_{i1}^{(1)}}{a_{11}^{(1)}} * a_{1i}^{(1)} \right| = \left| a_{ii} - \frac{a_{i1}}{a_{11}} * a_{1i} \right| \\
&\geq |a_{ii}| - \frac{|a_{i1}|}{|a_{11}|} * |a_{1i}| \\
&= |a_{ii}| - |a_{i1}| + |a_{i1}| - \frac{|a_{i1}|}{|a_{11}|} * |a_{1i}| \\
&= |a_{ii}| - |a_{i1}| + \frac{|a_{i1}|}{|a_{11}|} * (|a_{11}| - |a_{1i}|) \\
&> \sum_{j=2, j\neq i}^{n} |a_{ij}| + \frac{|a_{i1}|}{|a_{11}|} \sum_{j=2, j\neq i}^{n} |a_{1j}| \\
&= \sum_{j=2, j\neq i}^{n} |a_{ij}| + \sum_{j=2, j\neq i}^{n} \frac{|a_{i1}|}{|a_{11}|} |a_{1j}| \\
&\geq \sum_{j=2, j\neq i}^{n} \left| a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} \right| \\
&= \sum_{j=2, j\neq i}^{n} |a_{ij}^{(2)}|
\end{aligned}$$

Thus $A^{(2)}$ is still diagonally dominant. Since the subsequent steps of Gaussian elimination mimic the first, except for being applied to submatrices of smaller size, it suffices to conclude that Gaussian elimination without pivoting preserves the diagonal dominance of a matrix. ■

Theorem 3.5 *If the Gaussian elimination with scaled row pivoting algorithm is applied to a diagonally dominant matrix $A \in \mathbb{R}^{n\times n}$, then the pivot vector will be $p = [1, 2, \ldots, n]^T$. That is, no permutation is needed.*

Proof: By theorem 3.4, it is sufficient to prove that the first pivot row chosen by Algorithm 3.10 is row one. That is, it is enough to show that

$$\frac{|a_{11}|}{s_1} > \frac{|a_{i1}|}{s_i}, \quad \forall\ i = 2, \ldots, n.$$

Because of the diagonal dominance,

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}| \quad \Longrightarrow \quad |a_{ii}| > |a_{ij}|,\ \ 1 \leq j \leq n,\ \ j \neq i \quad \Longrightarrow \quad s_i = \max_{1\leq j\leq n} |a_{ij}| = |a_{ii}|,$$

for $i = 1, \ldots, n$, hence

$$\frac{|a_{11}|}{s_1} = \frac{|a_{11}|}{|a_{11}|} = 1, \quad \text{and} \quad \frac{|a_{i1}|}{s_i} = \frac{|a_{i1}|}{|a_{ii}|} < 1, \quad \forall\ i = 2, \ldots n.$$

Thus the theorem is proved. ■

3.4.3 Tridiagonal System

A square matrix $A = [a_{ij}]$ is said to be tridiagonal if $a_{ij} = 0$ for all i, j that satisfy $|i-j| > 1$, that is, A has the structure

$$A = \begin{bmatrix} a_{11} & a_{12} & & & \\ a_{21} & a_{22} & a_{23} & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & a_{n-1,n} \\ & & & a_{n,n-1} & a_{n,n} \end{bmatrix}.$$

In the display, matrix elements not shown are zero's. The factorization algorithm can be simplified considerably in the case of tridiagonal matrix if Guassian elimination can be applied safely without pivoting. This is true, for example, if the matrix is symmetric positive definite or diagonally dominant. The L and U factors would have the form

$$L = \begin{bmatrix} 1 & & & & \\ \ell_{21} & 1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & \ell_{n,n-1} & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} u_{11} & u_{12} & & & \\ & u_{22} & u_{23} & & \\ & & \ddots & \ddots & \\ & & & \ddots & u_{n-1,n} \\ & & & & u_{nn} \end{bmatrix},$$

and the entries are computed by the simple algorithm which only costs $3n$ flops.

Algorithm 3.13 (Tridiagonal LU Factorization) *This algorithm computes the LU factorization for a tridiagonal matrix without using pivoting strategy.*

$U(1,1) = A(1,1)$
for $i = 2, \dots, n$ **do**
  $U(i-1,i) = A(i-1,i)$
  $L(i,i-1) = A(i,i-1)/U(i-1,i-1)$
  $U(i,i) = A(i,i) - L(i,i-1) * U(i-1,i)$
end for

For practical implementation, A can be overwritten by the entries of L and U. Moreover, a tridiagonal matrix can be stored in an $n \times 3$ array. Alternatively we can arrange the notation like this

$$A = \begin{bmatrix} d_1 & c_1 & & & \\ a_1 & d_2 & c_2 & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ & & & a_{n-1} & d_n \end{bmatrix}, \tag{3.32}$$

and use three n-vectors for storage. A tridiagonal linear system arises in many applications, such as finite difference discretization to second order linear boundary-value problem and the cubic spline approximations. Such a system can be solved efficiently by the following algorithm, which costs only $8n$ flops.

Algorithm 3.14 (Solving Tridiagonal System) *This algorithm solves a tridiagonal linear sytem $Ax = b$ of the form (3.32) using Gaussian elimination without pivoting.*

for $i = 2, \ldots, n$ **do**
  $t = a(i-1)/d(i-1)$
  $d(i) = d(i) - t * c(i-1)$
  $b(i) = b(i) - t * b(i-1)$
end for
$x(n) = b(n)/d(n)$
for $i = n-1, \ldots, 1$ **do**
  $x(i) = (b(i) - c(i) * x(i+1))/d(i)$
end for

If partial pivoting is necessary, then, in addition to the pivoting vector, an extra storage should be arranged for the entries appear in the second superdiagonal because of pivoting.

3.4.4 General Banded Systems

In many applications that involve linear systems, the coefficient matrix is banded. Formally, we say that $A = [a_{ij}]$ has upper bandwidth q if $a_{ij} = 0$ whenever $j > i + q$ and lower bandwidth p if $a_{ij} = 0$ whenever $i > j+p$. Substantial economies can be realized when solving banded systems because the triangular factors in the LU factorization are also banded.

3.5 Perturbation Analysis

In this section, we develop some perturbation theory for the problem of solving linear systems $Ax = b$. If we solve such a system numerically, we obtain not the exact solution x but an approximate computed solution $\widehat{x}$. The difference

$$e = x - \widehat{x}$$

is called the error vector which is, however, not known. Instead one can test the accuracy of $\widehat{x}$ by forming $A\widehat{x}$ to see whether it is close to b. Thus we have the definition for the residual vector.

Definition 3.2 *Let $\widehat{x}$ be the computed solution to the linear system of equations $Ax = b$. Then the vector*

$$r = b - A\widehat{x} \tag{3.33}$$

is called the residual vector.

Then we can derive the residual equation

$$Ae = Ax - A\widehat{x} = b - A\widehat{x} = r \tag{3.34}$$

between the error vector and the residual vector.

Notice that $\widehat{x}$ is the exact solution of the linear system

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

which has a perturbed right-hand side

$$\widehat{b} = b - r.$$

Then

$$\begin{aligned} \|x - \widehat{x}\| &= \|A^{-1}b - A^{-1}\widehat{b}\| = \|A^{-1}(b - \widehat{b})\| \\ &\leq \|A^{-1}\|\|b - \widehat{b}\| = \|A^{-1}\|\|b\|\frac{\|b - \widehat{b}\|}{\|b\|} = \|A^{-1}\|\|Ax\|\frac{\|b - \widehat{b}\|}{\|b\|} \\ &\leq \|A^{-1}\|\|A\|\|x\|\frac{\|b - \widehat{b}\|}{\|b\|} \end{aligned}$$

Therefore

$$\frac{\|x - \widehat{x}\|}{\|x\|} \leq \kappa(A)\frac{\|b - \widehat{b}\|}{\|b\|} = \kappa(A)\frac{\|r\|}{\|b\|}, \tag{3.35}$$

where

$$\kappa(A) = \|A\|\|A^{-1}\| \tag{3.36}$$

is called the condition number of A.

On the other hand, by the residual vector, we have

$$\|r\|\|x\| = \|Ae\|\|A^{-1}b\| \leq \|A\|\|A^{-1}\|\|e\|\|b\| = \kappa(A)\|x - \widehat{x}\|\|b\|.$$

Hence

$$\frac{1}{\kappa(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|x - \widehat{x}\|}{\|x\|}. \tag{3.37}$$

Therefore we have established relationships between the relative error in x and b.

Theorem 3.6

$$\frac{1}{\kappa(A)}\frac{\|r\|}{\|b\|} \leq \frac{\|x - \widehat{x}\|}{\|x\|} \leq \kappa(A)\frac{\|r\|}{\|b\|}. \tag{3.38}$$

Next we further analyze the situation when there are both perturbations in the coefficient matrix and the right-hand-side vector.

Lemma 3.4 *Suppose that x and $\widetilde{x}$ satisfy*

$$Ax = b \qquad \text{and} \qquad (A + \triangle A)\widetilde{x} = b + \triangle b,$$

where $A \in \mathbb{R}^{n\times n}$*,* $\triangle A \in \mathbb{R}^{n\times n}$*,* $0 \neq b \in \mathbb{R}^n$*, and* $\triangle b \in \mathbb{R}^n$*, with*

$$\frac{\|\triangle A\|}{\|A\|} \leq \delta \qquad \text{and} \qquad \frac{\|\triangle b\|}{\|b\|} \leq \delta.$$

If $\kappa(A)\cdot\delta < 1$*, then* $A + \triangle A$ *is nonsingular and*

$$\frac{\|\widetilde{x}\|}{\|x\|} \leq \frac{1+\kappa(A)\delta}{1-\kappa(A)\delta}. \tag{3.39}$$

Proof: Since $\|A^{-1}\triangle A\| \leq \|A^{-1}\|\|\triangle A\| \leq \delta\|A^{-1}\|\|A\| = \delta\kappa(A) < 1$, it follows from Theorem 1.16 that $A + \triangle A$ is nonsingular. Now $(A+\triangle A)\widetilde{x} = b + \triangle b$,

$$(I + A^{-1}\triangle A)\widetilde{x} = A^{-1}b + A^{-1}\triangle b = x + A^{-1}\triangle b,$$

and so by taking norms and using Theorem 1.15 we find

$$\begin{aligned}
\|\widetilde{x}\| &\leq \|(I + A^{-1}\triangle A)^{-1}\|\left(\|x\| + \|A^{-1}\|\|\triangle b\|\right)\\
&\leq \|(I + A^{-1}\triangle A)^{-1}\|\left(\|x\| + \delta\|A^{-1}\|\|b\|\right)\\
&\leq \frac{1}{1-\|A^{-1}\triangle A\|}\left(\|x\| + \delta\|A^{-1}\|\|b\|\right)\\
&\leq \frac{1}{1-\delta\kappa(A)}\left(\|x\| + \delta\|A^{-1}\|\|b\|\right)\\
&= \frac{1}{1-\delta\kappa(A)}\left(\|x\| + \delta\|A^{-1}\|\|Ax\|\right)\\
&\leq \frac{1}{1-\delta\kappa(A)}\left(\|x\| + \delta\|A^{-1}\|\|A\|\|x\|\right)\\
&= \frac{1}{1-\delta\kappa(A)}\left(\|x\| + \delta\kappa(A)\|x\|\right)\\
&= \frac{1}{1-\delta\kappa(A)}\left(1 + \delta\kappa(A)\right)\|x\|.
\end{aligned}$$

Therefore

$$\frac{\|\widetilde{x}\|}{\|x\|} \leq \frac{1+\delta\kappa(A)}{1-\delta\kappa(A)}.$$

■

Theorem 3.7 *If the conditions of Lemma 3.4 hold then*

$$\frac{\|x-\widetilde{x}\|}{\|x\|} \leq \frac{2\delta}{1-\kappa(A)\delta}\kappa(A) \tag{3.40}$$

Proof: Since $\widetilde{x}$ satisfies $(A+\triangle A)\widetilde{x} = b + \triangle b$, $A\widetilde{x} = b + \triangle b - \triangle A\widetilde{x}$. Then we have

$$A\widetilde{x} - Ax = \triangle b + \triangle A\widetilde{x}$$

and

$$\widetilde{x} - x = A^{-1} \left(\triangle b + \triangle A \widetilde{x} \right).$$

Hence

$$\begin{aligned} \|\widetilde{x} - x\| &\leq \|A^{-1}\| \left(\|\triangle b\| + \|\triangle A\| \|\widetilde{x}\| \right) \\ &\leq \|A^{-1}\| \left(\delta \|b\| + \delta \|A\| \|\widetilde{x}\| \right) \\ &= \delta \|A^{-1}\| \left(\|Ax\| + \|A\| \|\widetilde{x}\| \right) \\ &\leq \delta \|A\| \|A^{-1}\| \left(\|x\| + \|\widetilde{x}\| \right), \end{aligned}$$

which gives

$$\frac{\|\widetilde{x} - x\|}{\|x\|} \leq \delta\kappa(A) \left(1 + \frac{\|\widetilde{x}\|}{\|x\|} \right) \leq \delta\kappa(A) \left(1 + \frac{1 + \kappa(A)\delta}{1 - \kappa(A)\delta} \right) = \frac{2\delta\kappa(A)}{1 - \delta\kappa(A)}.$$

■

Chapter 4

Iterative Methods for Solving Systems of Linear Equations

In this chapter we present some iterative methods for solving a linear system of equations $Ax = b$.

4.1 Classic Iterative Methods

4.1.1 Basic Concept

First of all we give an example to illustrate the process of iterative methods for solving systems of linear equations.

Consider solving

$$\begin{bmatrix} 3 & 2 \\ 1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 5 \\ 5 \end{bmatrix}.$$

This system has the exact solution $x_1 = x_2 = 1$. Equivalently we can write the system as

$$\begin{cases} 3x_1 + 2x_2 = 5 \\ x_1 + 4x_2 = 5 \end{cases}$$

This implies that

$$\begin{cases} x_1 = \frac{1}{3}(5 - 2x_2) \\ x_2 = \frac{1}{4}(5 - x_1) \end{cases}$$

A naieve idea is to solve the system by

$$\begin{cases} x_1^{(k)} = \frac{1}{3}(5 - 2x_2^{(k-1)}) \\ x_2^{(k)} = \frac{1}{4}(5 - x_1^{(k-1)}) \end{cases}$$

that is, to use the iterative formulation

$$\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} = \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k-1)} \\ x_2^{(k-1)} \end{bmatrix} \right)$$

If we choose the initial guess $x_1^{(0)} = x_2^{(0)} = 0$, we would otain

$$\begin{aligned} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} &= \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) = \begin{bmatrix} 1.6667 \\ 1.2500 \end{bmatrix} \\ \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} &= \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{4} \end{bmatrix} \left(\begin{bmatrix} 5 \\ 5 \end{bmatrix} - \begin{bmatrix} 0 & 2 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1.6667 \\ 1.2500 \end{bmatrix} \right) = \begin{bmatrix} 0.8333 \\ 0.8333 \end{bmatrix} \end{aligned}$$

By repeating the process, we have the following table

k	3	4	5	6	7
$x_1^{(k)}$	1.1111	0.9722	1.0185	0.9954	1.0031
$x_2^{(k)}$	1.0417	0.9722	1.0000	0.9954	1.0012

From this example, we observe that the basic idea is to split the coefficient matrix A into

$$A = M - (M - A), \tag{4.1}$$

for some matrix M, which is called the splitting matrix. Here we assume that A and M are both nonsingular. Then the original problem is rewritten in the equivalent form

$$Mx = (M - A)x + b.$$

This suggests an iterative process

$$x^{(k)} = (I - M^{-1}A)x^{(k-1)} + M^{-1}b \equiv Tx^{(k-1)} + c, \tag{4.2}$$

where T is usually called the iteration matrix. The initial vector $x^{(0)}$ can be arbitrary or be chosen according to certain conditions.

Two criteria for choosing the splitting matrix M are

1. $x^{(k)}$ is easily computed. More precisely, the system $Mx^{(k)} = y$ is easy to solve;

2. the sequence $\{x^{(k)}\}$ converges rapidly to the exact solution.

Note that one way to achieve the second goal is to choose M so that M^{-1} approximate A^{-1},

In the following subsections, we will introduce some of the mostly commonly used classic iterative methods.

4.1.2 Richard's Method

When we choose $M = I$ such that $A = I - (I - A)$, we obtain the iteration procedure

$$x^{(k)} = (I - A)x^{(k-1)} + b = x^{(k-1)} - Ax^{(k-1)} + b \equiv x^{(k-1)} + r^{(k-1)}, \tag{4.3}$$

This algorithm is called the Richard's method.

Algorithm 4.1 (Richard's Method)

for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    $r_i^{(k-1)} = b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k-1)}$
    $x_i^{(k)} = x_i^{(k-1)} + r_i^{(k-1)}$
  end for
end for

Two n-vectors are required to implement this algorithm.

4.1.3 Jacobi Method

If we decompose the coefficient matrix A as

$$A = L + D + U, \tag{4.4}$$

where D is the diagonal part, L is the strictly lower triangular part, and U is the strictly upper triangular part, of A, and choose $M = D$, then we derive the iterative formulation for Jacobi method:

$$x^{(k)} = -D^{-1}(L + U)x^{(k-1)} + D^{-1}b. \tag{4.5}$$

With this method, the iteration matrix $T = -D^{-1}(L + U)$ and $c = D^{-1}b$. Each component $x_i^{(k)}$ can be computed by

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Bigg/ a_{ii}. \tag{4.6}$$

Algorithm 4.2 (Jacobi Method)

for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, n$ **do**
    $x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Bigg/ a_{ii}$
  end for
end for

Two n-vectors are required to implement this algorithm. A big advantage of Jacobi method is that $x_i^{(k)}, i = 1, \ldots, n$, can be computed in parallel at each iteration k.

Another important observation is that in Jacobi method, only the components of $x^{(k-1)}$ are used to compute $x^{(k)}$. When computing $x_i^{(k)}$ for $i > 1$, $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ have already been computed and are likely to be better approximations to the exact $x_1, \ldots, x_{i-1}$ than $x_1^{(k-1)}, \ldots, x_{i-1}^{(k-1)}$. It seems reasonable to compute $x_i^{(k)}$ using these most recently computed values. This improvement induce the Gauss-Seidel method which is to be discussed in the next subsection.

4.1.4 Gauss-Seidel Method

The Gauss-Seidel method sets $M = D + L$ and defines the iteration as

$$x^{(k)} = -(D+L)^{-1}Ux^{(k-1)} + (D+L)^{-1}b. \tag{4.7}$$

That is, Gauss-Seidel method uses $T = -(D+L)^{-1}U$ as the iteration matrix. The formulation above can be rewritten as

$$x^{(k)} = -D^{-1}\left(Lx^{(k)} + Ux^{(k-1)} - b\right). \tag{4.8}$$

Hence each component $x_i^{(k)}$ can be computed by

$$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \Bigg/ a_{ii}. \tag{4.9}$$

Algorithm 4.3 (Gauss-Seidel Method)

for $k = 1, 2, \ldots$ **do**
  for $i = 1, 2, \ldots, n$ **do**
  $$x_i^{(k)} = \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k-1)}\right) \Bigg/ a_{ii}$$
  end for
end for

At each iteration, since $x_i^{(k)}$ can not be computed until $x_1^{(k)}, \ldots, x_{i-1}^{(k)}$ are available, Gauss-Seidel method is not a parallel algorithm in nature. Moreover, only one n-vector is required for implementation in theory, but two are usually used in practice.

4.1.5 Successive Over Relaxation (SOR) Method

The successive over relaxation (SOR) method choose $M = \omega^{-1}(D + \omega L)$, where $0 < \omega < 2$ is called the relaxation parameter, and defines the iteration

$$(D + \omega L)x^{(k)} = \left[(1-\omega)D - \omega U\right]x^{(k-1)} + \omega b. \tag{4.10}$$

Hence the iteration matrix $T = (D + \omega L)^{-1}((1-\omega)D - \omega U)$. Each component $x_i^{(k)}$ can be computed by the formulation

$$x_i^{(k)} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k-1)}. \tag{4.11}$$

The question of choosing a good relaxation parameter ω is a very complex topic.

4.1.6 Symmetric Successive Over Relaxation (SSOR) Method

In theory the symmetric successive over relaxation (SSOR) method chooses the splitting matrix $M = \frac{1}{\omega(2-\omega)}(D + \omega L)D^{-1}(D + \omega U)$ and iterates with the iteration matrix

$$T = (D + \omega U)^{-1} \left((1-\omega)D - \omega L\right) (D + \omega L)^{-1} \left((1-\omega)D - \omega U\right). \tag{4.12}$$

The idea is in fact to implement the SOR formulation twice, one forward and one backward, at each iteration. That is, SSOR method defines

$$\begin{aligned} (D + \omega L)x^{(k-\frac{1}{2})} &= \left((1-\omega)D - \omega U\right) x^{(k-1)} + \omega b &(4.13)\\ (D + \omega U)x^{(k)} &= \left((1-\omega)D - \omega L\right) x^{(k-\frac{1}{2})} + \omega b &(4.14) \end{aligned}$$

Each component $x_i^{(k)}$ is obtained by first computing

$$x_i^{(k-\frac{1}{2})} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k-\frac{1}{2})} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-1)} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k)} \tag{4.15}$$

followed by

$$x_i^{(k)} = \omega \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k-\frac{1}{2})} \right) \Big/ a_{ii} + (1-\omega) x_i^{(k-\frac{1}{2})}. \tag{4.16}$$

4.2 Convergence Analysis

Definition 4.1 (Eigenvalue) *Suppose A is a square matrix. The number λ is called an eigenvalue of A if there exists a nonzero vector x such that $Ax = \lambda x$. And x is called the corresponding eigenvector of A.*

Definition 4.2 *The polynomial defined by*

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

is called the characteristic polynomial of A.

Theorem 4.1 *λ is an eigenvalue of A if and only if λ is a root of the characteristic polynomial of A, that is, $p(\lambda) = 0$.*

Definition 4.3 (Spectrum and Spectral Radius) *The set of all eigenvalues of a matrix A is called the spectrum of A and is denoted by $\lambda(A)$. The spectral radius of A is*

$$\rho(\lambda) = \max\{|\lambda| | \lambda \in \lambda(A)\}. \tag{4.17}$$

Lemma 4.1 *If $A \in \mathbb{R}^{n\times n}$, then*

1. $\|A\|_2 = \sqrt{\rho(A^T A)}$;
2. $\rho(A) \leq \|A\|$ *for any subordinate matrix norm.*

Proof: Proof for the second part. Suppose λ is an eigenvalue of A and $x \neq 0$ is a corresponding eigenvector such that $Ax = \lambda x$ and $\|x\| = 1$. Then

$$|\lambda| = |\lambda|\|x\| = \|\lambda x\| = \|Ax\| \leq \|A\|\|x\| = \|A\|,$$

that is, $|\lambda| \leq \|A\|$. Since λ is arbitrary, this implies that $\rho(A) = \max|\lambda| \leq \|A\|$. ∎

Theorem 4.2 *For any A and any $\epsilon > 0$, there exists a subordinate norm such that*

$$\rho(A) < \|A\| < \rho(A) + \epsilon. \tag{4.18}$$

Lemma 4.2 *If $\rho(A) < 1$, then $(I - A)^{-1}$ exists and $(I - A)^{-1} = \sum_{i=0}^{\infty} A^i = I + A + A^2 + \cdots$.*

Proof: Suppose that λ is an eigenvalue of A, then $1 - \lambda$ is an eigenvalue of $I - A$. Since $|\lambda| \leq \rho(A) < 1$, this implies $1 - \lambda \neq 0$. Hence 0 is not an eigenvalue of $I - A$, which means $(I - A)$ is nonsingular.

Next we show that $(I - A)^{-1} = I + A + A^2 + \cdots$. Since

$$(I - A)\left(\sum_{i=0}^{m} A^i\right) = I - A^{m+1},$$

and $\rho(A) < 1$ implies $\|A^m\| \to 0$ as $m \to \infty$, we have

$$(I - A)\left(\lim_{m\to\infty} \sum_{i=0}^{m} A^i\right) = I.$$

This proves $(I - A)^{-1} = \sum_{k=1}^{\infty} A^k$.

Lemma 4.3 *Suppose that $A \in \mathbb{R}^{n\times n}$ and $\|\cdot\|$ is a subordinate matrix norm. If $\|A\| < 1$, then $I - A$ is nonsingular and*

$$(I - A)^{-1} = \sum_{k=0}^{\infty} A^k, \tag{4.19}$$

with

$$\|(I - A)^{-1}\| \leq \frac{1}{1 - \|A\|}. \tag{4.20}$$

Proof: Suppose $I-A$ is singular. Then there exist $x \in \mathbb{R}^n$, $x \neq 0$ (so $\|x\| \neq 0$) such that $(I-A)x = 0$. Thus $x = Ax$ and $\|x\| = \|Ax\| \leq \|A\|\|x\|$, which gives $\|A\| > 1$. However, this contradicts to the assumption that $\|A\| < 1$. Hence $I-A$ is nonsingular.

Next, one can verify that

$$(I-A)\left(\sum_{k=0}^{m} A^k\right) = I - A^{m+1}.$$

Since $\|A\| < 1$, $\lim\limits_{m\to\infty} A^m = 0$, hence

$$(I-A)\left(\sum_{k=0}^{\infty} A^k\right) = (I-A)\left(\lim_{m\to\infty}\sum_{k=0}^{m} A^k\right) = I - \lim_{m\to\infty} A^{m+1} = I.$$

This shows that $(I-A)^{-1} = \sum\limits_{k=0}^{\infty} A^k$. Finally, since $\|A\| < 1$,

$$\|(I-A)^{-1}\| = \left\|\sum_{k=0}^{\infty} A^k\right\| \leq \sum_{k=0}^{\infty}\|A^k\| \leq \sum_{k=0}^{\infty}\|A\|^k = \frac{1}{1-\|A\|}.$$

■

Theorem 4.3 *The following statements are equivalent.*

1. *A is a convergent matrix, i.e., $A^k \to \infty$ as $k \to \infty$;*
2. *$\lim\limits_{k\to\infty}\|A^k\| = 0$ for some subordinate matrix norm;*
3. *$\lim\limits_{k\to\infty}\|A^k\| = 0$ for all subordinate matrix norm;*
4. *$\rho(A) < 1$;*
5. *$\lim\limits_{k\to\infty} A^k x = 0$ for any x.*

Theorem 4.4 *For any $x^{(0)} \in \mathbb{R}^n$, the sequence produced by*

$$x^{(k)} = Tx^{(k-1)} + c, \quad k = 1, 2, \dots, \tag{4.21}$$

converges to the unique solution of $x = Tx + c$ if and only if

$$\rho(T) < 1. \tag{4.22}$$

Proof: Suppose $\rho(T) < 1$. The sequence of vectors $x^{(k)}$ produced by the iterative formulation are

$$\begin{aligned}
x^{(1)} &= Tx^{(0)} + c \\
x^{(2)} &= Tx^{(1)} + c = T^2x^{(0)} + (T+I)c \\
x^{(3)} &= Tx^{(2)} + c = T^3x^{(0)} + (T^2+T+I)c \\
&\vdots
\end{aligned}$$

In general

$$x^{(k)} = T^k x^{(0)} + (T^{k-1} + T^{k-2} + \cdots T + I)c.$$

Since $\rho(T) < 1$, $\lim_{k=0}^{\infty} T^k x^{(0)} = 0$ for any $x^{(0)} \in \mathbb{R}^n$. Then

$$x^{(k)} \to (T^{k-1} + T^{k-2} + \cdots T + I)c, \quad \text{as} \quad k \to \infty.$$

By Lemma

$$x^{(k)} \to (I - T)^{-1} c \quad \text{as} \quad k \to \infty.$$

Conversely, suppose that the sequence of vectors $x^{(k)}$ converges to $x = (I - T)^{-1}c$. Since

$$x - x^{(k)} = Tx + c - Tx^{(k-1)} - c = T(x - x^{(k-1)}) = T^2(x - x^{(k-2)}) = \cdots = T^k(x - x^{(0)}).$$

Let $z = x - x^{(0)}$. Then

$$\lim_{k\to\infty} T^k z \lim_{k\to\infty} (x - x^{(k)}) = 0.$$

It follows from theorem $\rho(T) < 1$.

Corollary 4.1 *If $\|T\| < 1$ for some subordinate matrix norm, then the sequence produced by*

$$x^{(k)} = Tx^{(k-1)} + c$$

converges to the solution of $Ax = b$ for any initial vector $x^{(0)}$.

Proof: Since $\rho(T) < \|T\|$ for any subordinate matrix norm, the result follows immediately from the previous theorem.

Theorem 4.5 *If $\delta = \|T\| < 1$, then*

$$\|x^{(k)} - x\| \leq \frac{\delta}{1-\delta} \|x^{(k)} - x^{(k-1)}\|. \tag{4.23}$$

Proof: Since $x^{(k)} - x = T(x^{(k-1)} - x)$,

$$\|x^{(k)} - x\| \leq \|T\| \|x^{(k-1)} - x\| = \delta \|x^{(k-1)} - x^{(k)} + x^{(k)} - x\| \leq \delta \|x^{(k-1)} - x^{(k)}\| + \delta \|x^{(k)} - x\|,$$

and $1 - \delta > 0$, we obtain

$$\|x^{(k)} - x\| \leq \frac{\delta}{1-\delta} \|x^{(k)} - x^{(k-1)}\|.$$

∎

This theorem implies that we can stop the iteration if $\|x^{(k)} - x^{(k-1)}$ is less than a small tolerance.

Theorem 4.6 *If $\|T\| < 1$, then the sequence $x^{(k)}$ converges to x for any initial $x^{(0)}$ and*

1. $\|x - x^{(k)}\| \le \|T\|^k \|x - x^{(0)}\|$
2. $\|x - x^{(k)}\| \le \frac{\|T\|^k}{1-\|T\|} \|x^{(1)} - x^{(0)}\|$.

Proof: Since $x = Tx + c$ and $x^{(k)} = Tx^{(k-1)} + c$,

$$\begin{aligned} x - x^{(k)} &= Tx + c - Tx^{(k-1)} - c \\ &= T(x - x^{(k-1)}) \\ &= T^2(x - x^{(k-2)}) = \cdots\cdots = T^k(x - x^{(0)}). \end{aligned}$$

The first statement can then be derived

$$\|x - x^{(k)}\| = \|T^k(x - x^{(0)})\| \le \|T\|^k \|x - x^{(0)}\|.$$

For the second result, we first show that $\|x^{(n)} - x^{(n-1)}\| \le \|T\|^{n-1} \|x^{(1)} - x^{(0)}\|$ for any $n \ge 1$. Since

$$\begin{aligned} x^{(n)} - x^{(n-1)} &= Tx^{(n-1)} + c - Tx^{(n-2)} - c \\ &= T(x^{(n-1)} - x^{(n-2)}) \\ &= T^2(x^{(n-2)} - x^{(n-3)}) = \cdots\cdots = T^{n-1}(x^{(1)} - x^{(0)}), \end{aligned}$$

$$\|x^{(n)} - x^{(n-1)}\| \le \|T\|^{n-1} \|x^{(1)} - x^{(0)}\|.$$

Let $m \ge k$,

$$\begin{aligned} x^{(m)} - x^{(k)} &= \left(x^{(m)} - x^{(m-1)}\right) + \left(x^{(m-1)} - x^{(m-2)}\right) + \cdots + \left(x^{(k+1)} - x^{(k)}\right) \\ &= T^{m-1}\left(x^{(1)} - x^{(0)}\right) + T^{m-2}\left(x^{(1)} - x^{(0)}\right) + \cdots + T^k\left(x^{(1)} - x^{(0)}\right) \\ &= \left(T^{m-1} + T^{m-2} + \cdots T^k\right)\left(x^{(1)} - x^{(0)}\right), \end{aligned}$$

hence

$$\begin{aligned} \|x^{(m)} - x^{(k)}\| &\le \left(\|T\|^{m-1} + \|T\|^{m-2} + \cdots + \|T\|^k\right) \|x^{(1)} - x^{(0)}\| \\ &= \|T\|^k \left(\|T\|^{m-k-1} + \|T\|^{m-k-2} + \cdots + 1\right) \|x^{(1)} - x^{(0)}\|. \end{aligned}$$

Since $\lim_{m\to\infty} x^{(m)} = x$,

$$\begin{aligned} \|x - x^{(k)}\| &= \lim_{m\to\infty} \|x^{(m)} - x^{(k)}\| \\ &\le \lim_{m\to\infty} \|T\|^k \left(\|T\|^{m-k-1} + \|T\|^{m-k-2} + \cdots + 1\right) \|x^{(1)} - x^{(0)}\| \\ &= \|T\|^k \|x^{(1)} - x^{(0)}\| \lim_{m\to\infty} \left(\|T\|^{m-k-1} + \|T\|^{m-k-2} + \cdots + 1\right) \\ &= \|T\|^k \frac{1}{1 - \|T\|} \|x^{(1)} - x^{(0)}\|. \end{aligned}$$

This proves the second result. ∎

Theorem 4.7 *If A is strictly diagonal domainant, then both the Jacobi and Gauss-Seidel methods converges for any initial vector $x^{(0)}$.*

Proof: By assumption, A is strictly diagonal dominant, hence $a_{ii} \neq 0$ (otherwise A is singular) and

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \ldots, n.$$

For Jacobi method, the iteration matrix $T_J = -D^{-1}(L+U)$ has entries

$$[T_J]_{ij} = \begin{cases} -\frac{a_{ij}}{a_{ii}} & i \neq j \\ 0 & i = j \end{cases}$$

Hence

$$\|T_J\|_\infty = \max_{1\leq i\leq n} \left| \sum_{j=1,j\neq i}^{n} \frac{a_{ij}}{a_{ii}} \right| = \max_{1\leq i\leq n} \frac{1}{|a_{ii}|} \sum_{j=1,j\neq i}^{n} |a_{ij}| < 1,$$

and this implies that the Jacobi method converges.

For Gauss-Seidel method, the iteration matrix $T_{GS} = -(D+L)^{-1}U$. Let λ be any eigenvalue of T_{GS} and y, $\|y\|_\infty = 1$, is a corresponding eigenvector. Thus

$$T_{GS}y = \lambda y \quad \Longrightarrow \quad -Uy = \lambda(D+L)y.$$

Hence for $i = 1, \ldots, n$,

$$-\sum_{j=i+1}^{n} a_{ij}y_j = \lambda \sum_{j=1}^{i} a_{ij}y_j = \lambda a_{ii}y_i + \sum_{j=1}^{i-1} a_{ij}y_j.$$

This gives

$$\lambda a_{ii}y_i = -\lambda \sum_{j=1}^{i-1} a_{ij}y_j + \sum_{j=i+1}^{n} a_{ij}y_j$$

and

$$|\lambda||a_{ii}||y_i| \leq |\lambda| \sum_{j=1}^{i-1} |a_{ij}||y_j| + \sum_{j=i+1}^{n} |a_{ij}||y_j|.$$

Choose the index k such that $|y_k| = 1 \geq |y_j|$ (this index can always be found since $\|y\|_\infty = 1$). Then

$$|\lambda||a_{kk}| \leq |\lambda| \sum_{j=1}^{k-1} |a_{kj}| + \sum_{j=k+1}^{n} |a_{kj}|$$

which gives

$$|\lambda| \leq \frac{\sum_{j=k+1}^{n} |a_{kj}|}{|a_{kk}| - \sum_{j=1}^{k-1} |a_{kj}|} < \frac{\sum_{j=k+1}^{n} |a_{kj}|}{\sum_{j=k+1}^{n} |a_{kj}|} = 1$$

Since λ is arbitrary, $\rho(T_{GS}) < 1$. This means the Gauss-Seidel method converges. ∎

Theorem 4.8 *If A is positive definite and the relaxation parameter ω satisfying $0 < \omega < 2$, then the SOR iteration converges for any initial vector $x^{(0)}$.*

Theorem 4.9 *If A is positive definite and tridiagonal, then $\rho(T_{GS}) = [\rho(T_J)]^2 < 1$ and the optimal choice of ω for the SOR iteration is*

$$\omega = \frac{2}{1 + \sqrt{1 - [\rho(T_J)]^2}}. \tag{4.24}$$

With this choice of ω, $\rho(T_{SOR}) = \omega - 1$.

Chapter 5

Solutions of Non-linear Equations

This chapter is devoted to the problem of determining roots of nonlinear equations (zeros of functions). It is a problem of frequent occurence in scientific work. The general question, posed in the simplest case of a real-valued function of one real variable, is this: Given a function $f : \mathbb{R} \to \mathbb{R}$, find $x \in \mathbb{R}$ such that$f(x) = 0$.

The extension of the simple formulation is a system of nonlinear equations of the form

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \qquad\vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

where $x_i \in \mathbb{R}$ are real variables and each f_i is a function from $\mathbb{R}^n$ to $\mathbb{R}$. To simplify the representation, we let

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

and $F : \mathbb{R}^n \to \mathbb{R}^n$ be the function

$$F(x) = F(x_1, x_2, \ldots, x_n) = \begin{bmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{bmatrix}.$$

Then the problem of solving system of nonlinear equations becomes finding $x \in \mathbb{R}^n$ such that $F(x) = 0$.

5.1 Preliminaries

Definition 5.1 *Let $\{x_n\}$ be a sequence of real numbers that converges to x^*. We say that the rate of convergence is*

1. *linear if there exist a constant* $0 < c < 1$ *and an integer* $N > 0$ *such that*

$$|x_{n+1} - x^*| \leq c|x_n - x^*|, \quad \forall\, n \geq N; \tag{5.1}$$

2. *superlinear if there exist a sequence* $\{c_n\}$, $c_n \to 0$ *as* $n \to \infty$, *and an integer* $N > 0$ *such that*

$$|x_{n+1} - x^*| \leq c_n|x_n - x^*|, \quad \forall\, n \geq N, \tag{5.2}$$

or, equivalently,

$$\lim_{n\to\infty} \frac{|x_{n+1} - x^*|}{|x_n - x^*|} = 0; \tag{5.3}$$

3. *quadratic if there exist a constant* $c > 0$ *(not necessarily less than 1) and an integer* $N > 0$ *such that*

$$|x_{n+1} - x^*| \leq c|x_n - x^*|^2, \quad \forall\, n \geq N. \tag{5.4}$$

In general, if there are positive constants c *and* α *and an integer* $N > 0$ *such that*

$$|x_{n+1} - x^*| \leq c|x_n - x^*|^\alpha, \quad \forall\, n \geq N, \tag{5.5}$$

then we say the rate of convergence is of order α.

Definition 5.2 *Suppose* $\{\beta_n\}$ *is a sequence known to converge to zero and* $\{x_n\}$ *converge s to* x^*. *If there exist a postive constant* c *and an integer* $N > 0$ *such that*

$$|x_n - x^*| \leq c|\beta_n|, \quad \forall\, n \geq N, \tag{5.6}$$

then we say $\{x_n\}$ *converges to* x^* *with rate of convergence* $O(\beta_n)$, *and write* $x_n = x^* + O(\beta_n)$.

Example 5.1 *Compare the convergence behavior of the sequences* $\{x_n\}$ *and* $\{y_n\}$, *where*

$$x_n = \frac{n+1}{n^2}, \quad \text{and} \quad y_n = \frac{n+3}{n^3}.$$

Sol: Note that both

$$\lim_{n\to\infty} x_n = 0 \quad \text{and} \quad \lim_{n\to\infty} y_n = 0.$$

Let $\alpha_n = \frac{1}{n}$ and $\beta_n = \frac{1}{n^2}$. Then

$$\begin{aligned} |x_n - 0| &= \frac{n+1}{n^2} \leq \frac{n+n}{n^2} = \frac{2}{n} = 2\alpha_n, \\ |y_n - 0| &= \frac{n+3}{n^3} \leq \frac{n+3n}{n^3} = \frac{4}{n^2} = 4\beta_n. \end{aligned}$$

Hence

$$x_n = 0 + O(\frac{1}{n}) \quad \text{and} \quad y_n = 0 + O(\frac{1}{n^2}).$$

This shows that $\{y_n\}$ converges to 0 much faster than $\{x_n\}$. ∎

5.2 Bisection Method

The bisection method is based on the intermediate value theorem. The idea behind the method is that if $f(x) \in C[a,b]$ and $f(a)f(b) < 0$, then there exists a root $c \in (a,b)$ such that $f(c) = 0$.

Algorithm 5.1 (Biesection Method) *Given a function $f(x)$ defined on an interval (a,b), the maximal number of iterations M, and stop criteria δ and ϵ, this algorithm tries to locate one root of $f(x)$.*

```
compute u = f(a), v = f(b), and e = b − a.
if sign(u) = sign(v) then
  stop
end if
for k = 1, 2, ..., M do
  e = e/2
  c = a + e
  w = f(c)
  if |e| < δ or |w| < ϵ then
    stop
  end if
  if sign(w) ≠ sign(u) then
    b = c
    v = w
  else
    a = c
    u = w
  end if
end for
```

Let $c_1, c_2, \ldots$ be the sequence of numbers produced. The algorithm should stop if one of the following conditions is satisfied.

1. the iteration number $k > M$,
2. $|c_k - c_{k-1}| < \delta$, or
3. $|f(c_k)| < \epsilon$.

In fact, these criteria can be applied to any iterative techniques considered in this chapter. Also note that the algorithm computes $c = a + (b-a)/2$, instead of $c = (a+b)/2$, for numerical reason. Furthermore, the bisection method find one zero, but not all the zeros, in the given interval $[a,b]$.

Let $[a_0, b_0], [a_1, b_1], \ldots$ denote the successive intervals produced by the bisection algorithm. Then

$$\begin{aligned} a &= a_0 \le a_1 \le a_2 \le \cdots \le b_0 = b \\ b &= b_0 \ge b_1 \ge b_2 \ge \cdots \ge a_0 = a. \end{aligned}$$

This means that the sequences $\{a_n\}$ and $\{b_n\}$ are bounded. Hence $\lim\limits_{n\to\infty} a_n$ and $\lim\limits_{n\to\infty} b_n$ exist. Since

$$\begin{aligned} b_1 - a_1 &= \frac{1}{2}(b_0 - a_0) \\ b_2 - a_2 &= \frac{1}{2}(b_1 - a_1) = \frac{1}{4}(b_0 - a_0) \\ &\vdots \\ b_n - a_n &= \frac{1}{2^n}(b_0 - a_0) \end{aligned}$$

hence

$$\lim_{n\to\infty} b_n - \lim_{n\to\infty} a_n = \lim_{n\to\infty}(b_n - a_n) = \lim_{n\to\infty}\frac{1}{2^n}(b_0 - a_0) = 0.$$

Therefore

$$\lim_{n\to\infty} a_n = \lim_{n\to\infty} b_n \equiv z.$$

Since f is a continuous function

$$lim_{n\to\infty} f(a_n) = f(\lim_{n\to\infty} a_n) = f(z) \quad \text{and} \quad lim_{n\to\infty} f(b_n) = f(\lim_{n\to\infty} b_n) = f(z).$$

The bisection method ensures that $f(a_n)f(b_n) \le 0$. This implies that $\lim\limits_{n\to\infty} f(a_n)f(b_n) = f^2(z) \le 0.$, and, consequently, $f(z) = 0$. That is, the limit of the sequences $\{a_n\}$ and $\{b_n\}$ is a zero of f in $[a, b]$.

Let $c_n = \frac{1}{2}(a_n + b_n)$. Then

$$|z - c_n| = |\lim_{n\to\infty} a_n - \frac{1}{2}(a_n + b_n)| \le |b_n - \frac{1}{2}(a_n + b_n)| = \frac{1}{2}|b_n - a_n| = \frac{1}{2^{n+1}}|b_0 - a_0|.$$

This proves the following theorem.

Theorem 5.1 *Let $[a_0, b_0], [a_1, b_1], \ldots$ denote the intervals produced by the bisection algorithm. Then $\lim\limits_{n\to\infty} a_n$ and $\lim\limits_{n\to\infty} b_n$ exist, are equal, and represent a zero of $f(x)$. If $z = \lim\limits_{n\to\infty} a_n = \lim\limits_{n\to\infty} b_n$ and $c_n = \frac{1}{2}(a_n + b_n)$, then*

$$|z - c_n| \le \frac{1}{2^{n+1}}\left(b_0 - a_0\right).$$

Remarks 5.1 *The implication of the theorem above is that $\{c_n\}$ converges to z with the rate of $O(2^{-n})$.*

Example 5.2 *If bisection method starts with interval $[50, 75]$, then how many steps should be taken to compute a root with relative error that is less than 10^{-12}?*

Sol: We want to seek an n such that

$$\frac{|z - c_n|}{|z|} \leq 10^{-12}.$$

Since the bisection method starts with the interval $[50, 75]$, this implies that $z \geq 50$, hence it is sufficient to show

$$\frac{|z - c_n|}{|z|} \leq \frac{|z - c_n|}{50} \leq 10^{-12}.$$

That is, we solve

$$2^{-(n+1)}(75 - 50) \leq 50 \times 10^{-12}$$

for n, which gives $n \geq 38$. ■

5.3 Newton's Method

5.3.1 Derivation of Newton's Method

Newton-Raphson's method is widely applied to problems of finding a zero of a function. It is simply called Newton's method. Suppose that $f : \mathbb{R} \to \mathbb{R}$ and $f \in C^2[a, b]$, i.e., f" exists and is continuous. If x_* is a zero of f and x is a point close to x_* such that $x_* = x + h$, then by Taylor's theorem

$$\begin{aligned} 0 = f(x_*) = f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f"(x)h^2 + \frac{1}{3!}f'''(x)h^3 + \cdots \\ &= f(x) + f'(x)h + O(h^2). \end{aligned}$$

Since we assume that x is close to x_*, i.e., h is small, $O(h^2)$ is negligible. It is reasonable to drop $O(h^2)$ terms. This implies

$$f(x) + f'(x)h \approx 0 \quad \text{and} \quad h \approx -\frac{f(x)}{f'(x)},$$

if $f'(x) \neq 0$. Hence

$$x + h = x - \frac{f(x)}{f'(x)}$$

is a better approximation to x_*. This sets the stage for the Newton-Raphson's method, which starts with an initial approximation x_0 and generates the sequence $\{x_k\}_{k=0}^{\infty}$ defined by

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}. \tag{5.7}$$

Figure gives a graphic interpretation of the Newon's method. Since the Taylor's expansion of $f(x)$ at x_k is given by

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2}f"(x_k)(x - x_k)^2 + \cdots .$$

At x_k, one uses the tangent line

$$y = \ell(x) = f(x_k) + f'(x_k)(x - x_k)$$

to approximate the curve of $f(x)$ and uses the zero of the tangent line to approximate the zero of $f(x)$, which gives the formulation, and tehn uses this point, x_{k+1}, as a new starting point to repeat the process. Hence the Newton's method involving linearizing the function, that is, the function $f(x)$ is replaced by a linear function $\ell(x)$ such that $\ell(x_k) = f(x_k)$ and $\ell'(x_k) = f'(x_k)$.

Algorithm 5.2 (Newton's Method) *Given a function $f : \mathbb{R} \to \mathbb{R}$, an initial guess x_0 to the zero of f, and stop criteria M, δ, and ϵ, this algorithm performs the Newton's iteration to approximate one root of f.*

$u = f(x_0)$
$v = f'(x_0)$
$x_1 = x_0 - \frac{u}{v}$
$k = 1$
$u = f(x_k)$
while $(k < M)$ or $(|x_k - x_{k-1}| < \delta)$ or $(|f(x_k)| < \epsilon$ **do**
  $v = f'(x_k)$
  $x_{k+1} = x_k - \frac{u}{v}$
  $k = k + 1$
  $u = f(x_k)$
end while

Note that there are two function evaluations, $f(x_k)$ and $f'(x_k)$, in each Newton iteration. The stop criteria δ and ϵ are usually chosen to be a constant times of square root of the machine epsilon, i.e., $\sqrt{\epsilon_M}$. Deciding when to stop is somewhat difficult, and can not be perfect for every problem, yet it needs some careful attention.

5.3.2 Convergence Analysis

Suppose that f'' is continuous and x_* is a simple zero of f, i.e., $f(x_*) = 0$ but $f'(x_*) \neq 0$. Choose $\delta > 0$ and let

$$D = \{x||x - x_*| \leq \delta\}$$

and

$$\gamma = \frac{1}{2} \cdot \frac{\max_{x\in D} |f''(x)|}{\min_{x\in D} |f'(x)|}.$$

Note that it is possible to choose δ such that $\rho = \delta\gamma < 1$ since f'' is continuous and when $\delta \to 0$, $x \to x_*$ and $\gamma \to \frac{1}{2} \cdot \frac{f''(x_*)}{f'(x_*)}$.

Now suppose we start Newton's iteration with x_0 satisfying $|e_0| = |x_0 - x_*| \le \delta$. By Taylor's theorem

$$0 = f(x_*) = f(x_0) + f'(x_0)(x_* - x_0) + \frac{1}{2} f''(\xi_0)(x_* - x_0)^2,$$

where ξ_0 is between x_* and x_0. Consequently $|\xi_0 - x_*| \le \delta$ and

$$-f(x_0) - f'(x_0)(x_* - x_0) = \frac{1}{2} f''(\xi_0)(x_* - x_0)^2.$$

One iteration of Newton's algorithm gives

$$x_1 = x_0 - \frac{f(x_0}{f'(x_0)}.$$

Hence

$$\begin{aligned} e_1 = x_1 - x_* &= x_0 - \frac{f(x_0)}{f'(x_0)} - x_* \\ &= \frac{1}{f'(x_0)} \left(-f(x_0) - x_* f'(x_0) + x_0 f'(x_0)\right) \\ &= \frac{1}{f'(x_0)} \left(-f(x_0) - f'(x_0)(x_* - x_0)\right) \\ &= \frac{1}{f'(x_0)} \cdot \frac{1}{2} \cdot f''(\xi_0)(x_* - x_0)^2 \\ &= \frac{f''(\xi_0)}{2f'(x_0)} e_0^2, \end{aligned}$$

and

$$|e_1| \le \gamma |e_0|^2 \le \gamma\delta |e_0| = \rho |e_0|.$$

Repeat the argument, we have, in general,

$$|e_k| \le \rho |e_{k-1}| \le \cdots \le \rho^k |e_0|.$$

Since $\rho < 1$, $|e_k| \to 0$ as $k \to 0$, that is, $x_k \to x_*$.

In summary, Newton's method will generate a seqence of numbers $\{x_k\}_{k \ge 0}$ that converges to the zero, x_*, of f if

1. f'' is continuous;

2. x_* is a simple zero of f; and

3. x_0 is close enough to x_*.

To investigate the convergence rate, we start with

$$x_{k+1} = x_{k+1} - x_* = x_k - \frac{f(x_k)}{f'(x_k)} - x_* = \frac{f'(x_k)e_k - f(x_k)}{f'(x_k)}.$$

Using Taylor's theorem

$$0 = f(x_*) = f(x_k - e_k) = f(x_k) - f'(x_k)e_k + \frac{1}{2}f''(\xi_k)e_k^2,$$

where ξ_k is between x_k and x_*, one has

$$f'(x_k)e_k - f(x_k) = \frac{1}{2}f''(\xi_k)e_k^2.$$

Hence

$$|e_{k+1}| = \frac{|f''(\xi_k)|}{2|f'(x_k)|}|e_k|^2 \approx \frac{|f''(x_*)|}{2|f'(x_*)|}|e_k|^2 \equiv C|e_k|^2.$$

This shows that Newton's method is quadratic convergent.

Theorem 5.2 *Suppose f'' is continous and x_* is a simple zero of f. Then there exists a neighborhood D of x_* and a constant C such that when Newton's methdo is applied with starting point $x_0 \in D$, the seqence $\{x_k\}$ generated converges to x_* and satisfies*

$$|x_{k+1} - x_*| \leq C|x_k - x_*|. \tag{5.8}$$

Definition 5.3 (Lipschitz Continuous) *A function $f(x)$ is Lipschitz continuous with Lipschitz contant γ in a set X, written $f \in \mathcal{L}ip_\gamma(X)$, if*

$$|f(x) - f(y)| \leq \gamma|x - y|,$$

for all $x, y \in X$.

Lemma 5.1 *Suppose $f : \Omega \to \mathbb{R}$ for some open interval $\Omega \subseteq \mathbb{R}$ and $f' \in \mathcal{L}ip_\gamma(\Omega)$. Then for all $x, y \in \Omega$,*

$$|f(y) - f(x) - f'(x)(y - x)| \leq \frac{\gamma}{2}(y - x)^2. \tag{5.9}$$

Proof: From Calculus

$$f(y) - f(x) - f'(x)(y - x) = \int_x^y (f'(u) - f'(x))\, du.$$

Apply changing variable $u = x + t(y - x)$, $du = (y - x)\,dt$ and the fact that $f' \in \mathcal{L}ip_\gamma(\Omega)$, we have

$$\begin{aligned} |f(y) - f(x) - f'(x)(y - x)| &= \left|\int_0^1 (f'(x + t(y - x)) - f'(x))\,(y - x)\,dt\right| \\ &\leq |y - x|\int_0^1 \gamma|t(y - x)|\,dt \\ &= \frac{\gamma}{2}|y - x|^2. \end{aligned}$$

$\blacksquare$

Theorem 5.3 *Let $f : \Omega \to \mathbb{R}$ for some open interval $\Omega \subseteq \mathbb{R}$. Assume*

1. *exists $x_* \in \Omega$ such that $f(x_*) = 0$;*
2. *$f' \in \mathcal{L}ip_\gamma(\Omega)$;*
3. *$\exists\, \rho > 0$ such that $|f'(x)| \geq \rho$ for all $x \in \Omega$, that is, $f'(x) \neq 0$ for all $x \in \Omega$.*

Then there exists $\eta > 0$ such that if $|x_0 - x_*| < \eta$, then the sequence produced by Newton's iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, 2, \dots,$$

exists and converges to x_*. Furthermore,

$$|x_{k+1} - x_*| \leq \frac{\gamma}{2\rho}|x_k - x_*|^2 \tag{5.10}$$

or, equivalently,

$$|e_{k+1}| \leq \frac{\gamma}{2\rho}|e_k|^2 \tag{5.11}$$

Proof: With the result of previous Lemma,

$$\begin{aligned}
|x_{k+1} - x_*| &= |x_k - \frac{f(x_k)}{f'(x_k)} - x_*| \\
&= |x_k - x_* - \frac{f(x_k) - f(x_*)}{f'(x_k)}| \\
&= \frac{1}{|f'(x_k)|}\,|f(x_*) - f(x_k) - f'(x_k)(x_k - x_*)| \\
&\leq \frac{\gamma}{2|f'(x_k)|}|x_k - x_*|^2 \\
&\leq \frac{\gamma}{2\rho}|x_k - x_*|^2.
\end{aligned}$$

■

Note that this convergence property is local since it requires the starting point $x_0 \in D$, that is, Newton's method only guarantee the convergence from a good starting point x_0 that is close enought to x_*. In fact, Newton's iteration may not converge at all if $|x_0 - x_*|$ is large. Therefore some globally convergent algorithm needs to be incorporated with Newton's method. Furthermore, $f'(x_*)$ must be nonzero for the method to converge quadratically. Indeed, if $f'(x_*) = 0$, i.e., x_* is a multiple root of f, then Newton's method converges only linearly. However, when f is a linear function, Newton's method will converge in one step.

A simple strategy to modify Newton's method for global convergent is to incorporate a back-tracking procedure in each Newton step. The idea is that in the direction of the tangent line, we find a point where the function value decreases in magnitude. More precisely, one involves

$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
while $|f(x_{k+1})| \geq |f(x_k)|$ **do**
  $x_{k+1} = \frac{1}{2}(x_{k+1} + x_k)$
end while

x_0	2.0
x_1	1.25
x_2	1.025
x_3	1.0003048780488
x_4	1.0000000464611
x_5	1.0

5.3.3 Examples and Pitfalls

Example 5.3 *The following table shows the convergence behavior of Newton's method applied to solving $f(x) = x^2 - 1 = 0$. Observe the quadratic convergence rate.*

■

Example 5.4 *Newton's method will fail (the sequence $\{x_k\}$ does not converge) on solving $f(x) = x^2 - 4x + 5 = 0$ since f does not have any real root.*

Example 5.5 *When Newton's method applied to $f(x) = \cos x$ with starting point $x_0 = 3$, which is close to the root $\frac{\pi}{2}$ of f, it produces $x_1 = -4.01525, x_2 = -4.8526, \cdots$, which converges to another root $-\frac{3\pi}{2}$.*

Example 5.6 *When Newton's method applied to $f(x) = xe^{-x}$ with starting point $x_x = 2.0$, it produces $x_1 = 4.0, x_2 = 5.333, \ldots$. The sequence $\{x_k\}$ diverges to ∞ slowly, however, $f(x_k)$ goes to zero rapidly as x_k gets larger in a finite precision environment, and could be mistaken as a zero of f.*

Example 5.7 *When Newton's method applied to $f(x) = x^3 - x - 3$ with starting point $x_0 = 0$, it produces $x_1 = -3, x_2 = -1.961538, x_3 = -1.147176, x_4 = -0.00679, x_5 = -3.000389, x_6 = -1.961818, x_7 = -1.147430, \ldots$. The sequence will not converge. But if the algorithm starts with $x_0 = 2$, then it produces $x_1 = 1.727272, x_2 = 1.67369, x_3 = 1.6717025, \ldots$. The sequence converges to the root* 1.671699881. *This example illustrates that the starting point x_0 must be close enough to the zero of f.*

5.3.4 System of Nonlinear Equations

First consider solving the following system of nonlinear equations:

$$\begin{cases} f_1(x_1, x_2) = 0, \\ f_2(x_1, x_2) = 0. \end{cases} \tag{5.12}$$

Suppose $(x_1^{(k)}, x_2^{(k)})$ is an approximation to the solution of the system above, and we try to compute $h_1^{(k)}$ and $h_2^{(k)}$ such that $(x_1^{(k)} + h_1^{(k)}, x_2^{(k)} + h_2^{(k)})$ satisfies the system. By the Taylor's theorem for two variables,

$$\begin{aligned}
0 = f_1(x_1^{(k)} + h_1^{(k)}, x_2^{(k)} + h_2^{(k)}) &\approx f_1(x_1^{(k)}, x_2^{(k)}) + h_1^{(k)} \frac{\partial f_1}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) + h_2^{(k)} \frac{\partial f_1}{\partial x_2}(x_1^{(k)}, x_2^{(k)}) \\
0 = f_2(x_1^{(k)} + h_1^{(k)}, x_2^{(k)} + h_2^{(k)}) &\approx f_2(x_1^{(k)}, x_2^{(k)}) + h_1^{(k)} \frac{\partial f_2}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) + h_2^{(k)} \frac{\partial f_2}{\partial x_2}(x_1^{(k)}, x_2^{(k)})
\end{aligned}$$

Put this in matrix form

$$\begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) & \frac{\partial f_1}{\partial x_2}(x_1^{(k)}, x_2^{(k)}) \\ \frac{\partial f_2}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) & \frac{\partial f_2}{\partial x_2}(x_1^{(k)}, x_2^{(k)}) \end{bmatrix} \begin{bmatrix} h_1^{(k)} \\ h_2^{(k)} \end{bmatrix} + \begin{bmatrix} f_1(x_1^{(k)}, x_2^{(k)}) \\ f_2(x_1^{(k)}, x_2^{(k)}) \end{bmatrix} \approx \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

The matrix

$$J(x_1^{(k)}, x_2^{(k)}) \equiv \begin{bmatrix} \frac{\partial f_1}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) & \frac{\partial f_1}{\partial x_2}(x_1^{(k)}, x_2^{(k)}) \\ \frac{\partial f_2}{\partial x_1}(x_1^{(k)}, x_2^{(k)}) & \frac{\partial f_2}{\partial x_2}(x_1^{(k)}, x_2^{(k)}) \end{bmatrix} \tag{5.13}$$

is called the Jacobian matrix. Set $h_1^{(k)}$ and $h_2^{(k)}$ be the solution of the linear system

$$J^{-1}(x_1^{(k)}, x_2^{(k)}) \begin{bmatrix} h_1^{(k)} \\ h_2^{(k)} \end{bmatrix} = - \begin{bmatrix} f_1(x_1^{(k)}, x_2^{(k)}) \\ f_2(x_1^{(k)}, x_2^{(k)}) \end{bmatrix}$$

then

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{bmatrix} = \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \end{bmatrix} + \begin{bmatrix} h_1^{(k+1)} \\ h_2^{(k+1)} \end{bmatrix}$$

is expected to be a better approximation.

In general, we solve the system of n nonlinear equations $f_i(x_1, \cdots, x_n) = 0, i = 1, \ldots, n$. Let

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \text{and} \quad F(X) = \begin{bmatrix} f_1(X) \\ f_2(X) \\ \vdots \\ f_n(X) \end{bmatrix}.$$

The problem can be formulated as solving

$$F(X) = 0, \quad F : \mathbb{R}^n \to \mathbb{R}^n. \tag{5.14}$$

Let $F'(X)$, where the (i, j) entry is $\frac{\partial f_i}{\partial x_j}(X)$, be the $n \times n$ Jacobian matrix. Then the Newton's iteration is defined as

$$X^{(k+1)} = X^{(k)} + S^{(k)}, \tag{5.15}$$

where $S^{(k)} \in \mathbb{R}^n$ is the solution of the linear system

$$F'(X^{(k)}) S^{(k)} = -F(X^{(k)}). \tag{5.16}$$

Newton's method for solving systems of nonlinear equations still enjoys the property of rapid quadratic convergence if the starting point is near the exact solution point in terms of vector norm. However, a significant weakness of Newton's method is the requirement that, at each iteration, a Jacobian matrix has to be evaluated and an $n \times n$ linear system involving this matrix must be solved. To construct the Jacobian matrix, n^2 partial derivatives have to be available and evaluated. Then an LU-factorization has to be performed for the solution of the the linear system involved.

5.4 Quasi-Newton's Method (Secant Method)

5.4.1 The Secant Method

The Newton's iteration

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

involves the evaluation of derivative of f. In many applications (situations), the derivative $f'(x)$ is very expensive to compute, or the function $f(x)$ is not given in an algebraic formula so that $f'(x)$ is not available.

An alternative is to replace the derivative evaluation by finite difference approximation. Since

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h},$$

it is reasonable to approximate $f'(x_k)$ in the Newton's iteration by

$$f'(x_k) \approx \frac{f(x_k + h_k) - f(x_k)}{h_k}$$

for some small h_k. This leads to the so-called finite-difference Newton's iteration

$$x_{k+1} = x_k - f(x_k) \frac{h_k}{f(x_k + h_k) - f(x_k)} \tag{5.17}$$

From geometric point of view, we use a secant line through x_k and a near-by point $x_k + h_k$ instead of the tangent line to approximate the function at the point x_k. The slope of the secant line is

$$s_k = \frac{f(x_k + h_k) - f(x_k)}{h_k} \tag{5.18}$$

and the equation is

$$M(x) = f(x_k) + s_k(x - x_k). \tag{5.19}$$

	finite-difference Newton	secant method
x_0	2	2
x_1	1.2500000266453	1.2500000266453
x_2	1.0250000179057	1.0769230844910
x_3	1.0003048001120	1.0082644643823
x_4	1.0000000464701	1.0003048781354
x_5	1	1.0000012544523
x_6		1.0000000001912
x_7		1

The zero of the secant line

$$x = x_k - \frac{f(x_k)}{s_k} = x_k - f(x_k)\frac{h_k}{f(x_k + h_k) - f(x_K)} \tag{5.20}$$

is then used as a new approximate x_{k+1}. This leads to exactly the same formulation for the finite-difference Newton's iteration.

Two questions arise so far: "Will this method work?" and "How should h_k be chosen?" From the limit definition of derivative, it seems reasonable to assume that the finite-difference formulation should work almost as well as Newton's method if h_k is chosen small enough. However, it requires two function evaluations $f(x_k + h_k)$ and $f(x_k)$ in each iteration. If the function evaluation is expensive, we may set

$$h_k = x_k - x_{k-1}.$$

This leads to the so-called secant method or quasi-Newton method.

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \tag{5.21}$$

Note that the secant method requires two initial guesses x_0 and x_{-1}. However, there is only one function evaluation required in each iteration.

It turns out that the secant method requires only a few more iterations to converge than the finite difference approach with a properly chosen h_k, but it's usually more efficient in terms of total number of function evaluations.

Example 5.8 *The following table shows the convergence history for the finite-difference Newton's method with $h_k = 10^{-7} \star x_k$ and secant method for solving $f(x) = x^2 - 1 = 0$.*

5.4.2 Error Analysis of Secant Method

Let x_* denote the exact solution of $f(x) = 0$, $e_k = x_k - x_*$ be the errors at the k-th step. Then

$$\begin{aligned}
e_{k+1} &= x_{k+1} - x_* \\
&= x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} - x_* \\
&= \frac{1}{f(x_k) - f(x_{k-1})}\left[x_k f(x_k) - x_k f(x_{k-1}) - x_k f(x_k) + x_{k-1} f(x_k) - x_* f(x_k) + x_* f(x_{k-1})\right] \\
&= \frac{1}{f(x_k) - f(x_{k-1})}\left[(x_{k+1} - x_*) f(x_k) - (x_k - x_*) f(x_{k-1})\right] \\
&= \frac{1}{f(x_k) - f(x_{k-1})}\left(e_{k-1} f(x_k) - e_k f(x_{k-1})\right) \\
&= e_k e_{k-1}\left(\frac{\frac{1}{e_k} f(x_k) - \frac{1}{e_{k-1}} f(x_k)}{x_k - x_{k-1}} \cdot \frac{x_k - x_{k-1}}{f(x_k - f(x_{k-1})}\right)
\end{aligned}$$

To estimate the numerator $\frac{\frac{1}{e_k} f(x_k) - \frac{1}{e_{k-1}} f(x_k)}{x_k - x_{k-1}}$, we apply the Taylor's theorem

$$f(x_k) = f(x_* + e_k) = f(x_*) + f'(x_*)e_k + \frac{1}{2} f''(x_*)e_k^2 + O(e_k^3),$$

to get

$$\frac{1}{e_k} f(x_k) = f'(x_*) + \frac{1}{2} f''(x_*)e_k + O(e_k^2).$$

Similarly,

$$\frac{1}{e_{k-1}} f(x_{k-1}) = f'(x_*) + \frac{1}{2} f''(x_*)e_{k-1} + O(e_{k-1}^2).$$

Hence

$$\frac{1}{e_k} f(x_k) - \frac{1}{e_{k-1}} f(x_{k-1}) \approx \frac{1}{2}(e_k - e_{k-1}) f''(x_*).$$

Since $x_k - x_{k-1} = e_k - e_{k-1}$ and

$$\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \to \frac{1}{f'(x_*)},$$

we have

$$e_{k+1} \approx e_k e_{k-1}\left(\frac{\frac{1}{2}(e_k - e_{k-1}) f'(x_*)}{e_k - e_{k-1}} \cdot \frac{1}{f'(x_*)}\right) = \frac{1}{2}\frac{f''(x_*)}{f'(x_*)} e_k e_{k-1} \equiv C e_k e_{k-1} \tag{5.22}$$

To estimate the convergence rate, we assume

$$|e_{k+1}| \approx \eta |e_k|^\alpha,$$

where $\eta > 0$ and $\alpha > 0$ are constants, i.e.,

$$\frac{|e_{k+1}|}{\eta |e_k|^\alpha} \to 1 \quad \text{as} \quad k \to \infty.$$

Then $|e_k| \approx \eta |e_{k-1}|^\alpha$ which implies $e_{k-1} \approx \eta^{-1/\alpha} |e_k|^{1/\alpha}$. Hence () gives

$$\eta |e_k|^\alpha \approx C |e_k| \eta^{-1/\alpha} |e_k|^{1/\alpha} \quad \Longrightarrow \quad C^{-1} \eta^{1+\frac{1}{\alpha}} \approx |e_k|^{1-\alpha+\frac{1}{\alpha}}.$$

Since $|e_k| \to 0$ as $k \to \infty$, and $C^{-1}\eta^{1+\frac{1}{\alpha}}$ is a nonzero constant,

$$1 - \alpha + \frac{1}{\alpha} = 0 \quad \Longrightarrow \quad \alpha = \frac{1+\sqrt{5}}{2} \approx 1.62.$$

This result implies that $C^{-1}\eta^{1+\frac{1}{\alpha}} \to 1$ and

$$\eta \to C^{\frac{\alpha}{1+\alpha}} = \left(\frac{f''(x_*)}{2f'(x_*)} \right)^{0.62}.$$

In summary, we have shown that

$$|e_{k+1}| = \eta |e_k|^\alpha, \quad \alpha \approx 1.62, \tag{5.23}$$

that is, the convergence rate is superlinear.

The rapidity of convergence of the secant method is not as good as the Newton's method, but is better than the bisection method. However, each iteration of the secant method requires only one function evaluation while the Newton's method requires two, namely, $f(x_k)$ and $f''(x_k)$. Therefore, we may regard two steps of secant method are comparable to one step of Newton's method. Thus

$$|e_{k+2}| \approx \eta |e_{k+1}|^\alpha \approx \eta^{1+\alpha} |e_k^{\frac{3+\sqrt{5}}{2}}.$$

Since $\frac{3+\sqrt{5}}{2} \approx 2.62$, this shows that secant method is more efficient than Newton's method. Of course, two steps of secant method would require a little more work than one step of Newton's method.

Next we will extend secant method to solving system of nonlinear equations. Recall that in one dimensional case, one uses the linear model

$$\ell_k(x) = f(x_k) + a_k(x - x_k) \tag{5.24}$$

to approximate the function $f(x)$ at x_k. That is, $\ell_k(x_k) = f(x_k)$ for any $a_k \in \mathbb{R}$. If we further require that $\ell'(x_k) = f'(x_k)$, then $a_k = f'(x_k)$. The zero of $\ell_k(x)$ is used to give a new approximate for the zero of $f(x)$, that is,

$$x_{k+1} = x_k - \frac{1}{f'(x_k)} f(x_k)$$

which yields Newton's method.

If $f'(x_k)$ is not available, one instead asks the linear model to satisfy

$$\ell_k(x_k) = f(x_k) \quad \text{and} \quad \ell_k(x_{k-1}) = f(x_{k-1}). \tag{5.25}$$

In doing this, the identity

$$f(x_{k-1}) = \ell_k(x_{k-1}) = f(x_k) + a_k(x_{k-1} - x_k)$$

gives

$$a_k = \frac{f(x_k - f(x_{k-1})}{x_k - x_{k-1}}. \tag{5.26}$$

Solving $\ell_k(x) = 0$ yields the secant iteration

$$x_{k+1} = x_k - \frac{x_k - x_{k-1}}{f(x_k - f(x_{k-1})} f(x_k). \tag{5.27}$$

In multiple dimension, the analogue affine modle becomes

$$M_k(x) = F(x_k) + A_k(x - x_k), \tag{5.28}$$

where $x, x_k \in \mathbb{R}^n$ and $A_k \in \mathbb{R}^{n\times n}$, and satisfies

$$M_k(x_k) = F(x_k), \tag{5.29}$$

for any A_k. The zero of $M_k(x)$ is then used to give a new approximate for the zero of $F(x)$, that is,

$$x_{k+1} = x_k - A_k^{-1} F(x_k). \tag{5.30}$$

The Newton's method chooses

$$A_k = F'(x_k) \equiv J(x_k) = \text{the Jacobian matrix}. \tag{5.31}$$

and yields the iteration

$$x_{k+1} = x_k - \left(F'(x_k)\right)^{-1} F(x_k). \tag{5.32}$$

When the Jacobian matrix $J(x_k) \equiv F'(x_k)$ is not available, one can require

$$M_k(x_{k-1}) = F(x_{k-1}). \tag{5.33}$$

Then

$$F(x_{k-1}) = M_k(x_{k-1}) = F(x_k) + A_k(x_{k-1} - x_k),$$

which gives

$$A_k(x_k - x_{k-1}) = F(x_k) - F(x_{k-1}) \tag{5.34}$$

and this is the so-called secant equation. Let

$$s_k = x_k - x_{k-1} \quad \text{and} \quad y_k = F(x_k) - F(x_{k-1}). \tag{5.35}$$

The secant equation becomes

$$A_k s_k = y_k. \tag{5.36}$$

However, this secant equation can not uniquely determine A_k. One way of choosing A_k is to minimize $M_k - M_{k-1}$ subject to the secant equation. Note

$$\begin{aligned} M_k(x) - M_{k-1}(x) &= F(x_k) + A_k(x - x_k) - F(x_{k-1}) - A_{k-1}(x - x_{k-1}) \\ &= (F(x_k - F(x_{k-1})) + A_k(x - x_k) - A_{k-1}(x - x_{k-1}) \\ &= A_k(x_k - x_{k-1}) + A_k(x - x_k) - A_{k-1}(x - x_{k-1}) \\ &= A_k(x - x_{k-1}) - A_{k-1}(x - x_{k-1}) \\ &= (A_k - A_{k-1})(x - x_{k-1}). \end{aligned}$$

For any $x \in \mathbb{R}^n$, we express

$$x - x_{k-1} = \alpha s_k + t_k, \tag{5.37}$$

for some $\alpha \in \mathbb{R}$, $t_k \in \mathbb{R}^n$, and $s_k^T t_k = 0$. Then

$$M_k - M_{k-1} = (A_k - A_{k-1})(\alpha s_k + t_k) = \alpha(A_k - A_{k-1})s_k + (A_k - A_{k-1})t_k.$$

Since

$$(A_k - A_{k-1})s_k = A_k s_k - A_{k-1}s_k = y_k - A_{k-1}s_k,$$

both y_k and $A_{k-1}s_k$ are old values, we have no control over the first part $(A_k - A_{k-1})s_k$. In order to minimize $M_k(x) - M_{k-1}(x)$, we try to choose A_k so that

$$(A_k - A_{k-1})t_k = 0 \tag{5.38}$$

for all $t_k \in \mathbb{R}^n$, $s_k^T t_k = 0$. This requires that $A_k - A_{k-1}$ to be a rank-one matrix of the form

$$A_k - A_{k-1} = u_k s_k^T \tag{5.39}$$

for some $u_k \in \mathbb{R}^n$. Then

$$u_k s_k^T s_k = (A_k - A_{k-1})s_k = y_k - A_{k-1}s_k$$

which gives

$$u_k = \frac{y_k - A_{k-1}s_k}{s_k^T s_k}. \tag{5.40}$$

Therefore,

$$A_k = A_{k-1} + \frac{(y_k - A_{k-1}s_k)s_k^T}{s_k^T s_k} \tag{5.41}$$

After A_k is determined, the new iterate x_{k+1} is derived from solving $M_k(x) = 0$. It can be done by first noting that

$$s_{k+1} = x_{k+1} - x_k \quad \Longrightarrow \quad x_{k+1} = x_k + s_{k+1} \tag{5.42}$$

and

$$M_k(x_{k+1}) = 0 \quad \Longrightarrow \quad F(x_k) + A_k(x_{k+1} - x_k) = 0 \quad \Longrightarrow \quad A_k s_{k+1} = -F(x_k) \tag{5.43}$$

These formulations give the Broyden's method.

Algorithm 5.3 (Broyden's Method) *Given a n-variable nonlinear function $F : \mathbb{R}^n \to \mathbb{R}^n$, an initial iterate x_0 and initial Jacobian matrix $A_0 \in \mathbb{R}^{n\times n}$ (e.g., $A_0 = I$), this algorithm finds the solution for $F(x) = 0$.*

for $k = 0, 1, \cdots$, **do**
  Solve $A_k s_{k+1} = -F(x_k)$ for s_{k+1}
  Update $x_{k+1} = x_k + s_{k+1}$
  Compute $y_{k+1} = F(x_{k+1}) - F(x_k)$
  Update $A_{k+1} = A_k + \dfrac{(y_{k+1} - A_k s_{k+1})s_{k+1}^T}{s_{k+1}^T s_{k+1}} = A_k + \dfrac{(y_{k+1} + F(x_k)s_{k+1}^T}{s_{k+1}^T s_{k+1}}$
end for

Since a new matrix A_k is used to approximate the Jacobian matrix in each iteration of the Broyden's method, LU-factorization has to be performed in order to solve the linear system $A_k s_{k+1} = -F(x_k)$ for s_{k+1}. One LU-factorization would cost $frac23n^3 + O(n^2)$ floating-point operations. But if we apply the Shermann-Morrison-Woodbury formula to (), we have

$$A_k^{-1} = A_{k-1}^{-1} + \frac{(s_k - A_k^{-1}y_k)y_k^T}{y_k^T y_k}. \tag{5.44}$$

With a given A_0^{-1}, we can then update A_k^{-1} in each iteration using this formula and s_{k+1} can be easily computed by matrix-vector multiplication $s_{k+1} = -A_k^{-1}F(x_k)$ in $O(n^2)$ operations.

5.5 Fixed Point and Functional Iteration

In this section, we consider the problem of finding a fixed point with functional iteration method for a given fuction $f : D \to \mathbb{R}$, where $D \subseteq \mathbb{R}$ is a closed set.

Definition 5.4 *x is called a fixed point of a given function f if $f(x) = x$.*

If $f : \mathbb{R} \to \mathbb{R}$ is a function, then the fixed point of f can be viewed as the intersection of the curve $y = f(x)$ and the straight line $y = x$.

The problem of finding a fixed point for a given function is equivalent to the problem of finding a zero for a nonlinear fucntion. For instance, given a function $f(x)$ and a root x_* such that $f(x_*) = 0$, let $g(x) = x - f(x)$. Then $g(x_*) = x_* - f(x_*) = x_*$. That is, x_* is a fixed point for $g(x)$. Conversely, suppose that $g(x)$ has a fixed point at x_*. Then one may define $f(x) = x - g(x)$ so that $f(x_*) = x_* - g(x_*) = x_* - x_* = 0$, i.e., x_* is a zero of $f(x)$.

Two questions arise: "When does a function have a fixed point?" and "How to find a fixed point?".

5.5.1 Functional Iteration

To determine a fixed point x_* for a continuous function f, we choose an initial point x_0 and generate a sequence of points $\{x_k\}_{k\geq 0}$ by

$$x_{k+1} = f(x_k), \quad k \geq 0. \tag{5.45}$$

This algorithm is called fixed-point iteation or functional iteration. This formulation may generate sequences that do not converge, e.g., $f(x) = 3x$. However, wehn the sequence converges, say,

$$\lim_{k\to\infty} x_k = x_*,$$

then, since f is continuous,

$$f(x_*) = f(\lim_{k\to\infty} x_k) = \lim_{k\to\infty} f(x_k) = \lim_{k\to\infty} x_{k+1} = x_*.$$

That is, x_* is a fixed point of f.

Note that Newton's method for solving $g(x) = 0$

$$x_{k+1} = x_k - \frac{g(x_k)}{g'(x_k)}$$

is just a special case of functional iteration in which

$$f(x) = x - \frac{g(x)}{g'(x)}.$$

Definition 5.5 *A function (mapping) f is said to be contractive if there exists a constant $0 \leq \lambda < 1$ such that*

$$|f(x) - f(y)| \leq \lambda |x - y| \tag{5.46}$$

for all x, y in the domain of f.

Theorem 5.4 (Contractive Mapping Theorem) *Suppose $f : D \to D$, where $D \subseteq \mathbb{R}$ is a closed set, is a contractive mapping. Then f has a unique fixed point in D. Moreover, this fixed point is the limit of every sequence obtained by*

$$x_{k+1} = f(x_k)$$

with any initial point x_0.

Proof: We first show that $\lim\limits_{k\to\infty} x_k$ exists. Since

$$x_k = x_0 + (x_1 - x_0) + (x_2 - x_1) + \cdots + (x_k - x_{k-1}) = x_0 + \sum_{i=1}^{k}(x_i - x_{i-1}),$$

$\{x_k\}_{k\geq 0}$ converges if and only if $\sum_{i=1}^{\infty}(x_i - x_{i-1})$ converges and it is sufficient to show $\sum_{i=1}^{\infty}|x_i - x_{i-1}|$ converges.

Since f is contractive, we have

$$\begin{aligned}
|x_i - x_{i-1}| &= |f(x_{i-1}) - f(x_{i-2})| \\
&\leq \lambda|x_{i-1} - x_{i-2}| \\
&\leq \lambda^2|x_{i-2} - x_{i-3}| \\
&\vdots \\
&\leq \lambda^{i-1}|x_1 - x_0|.
\end{aligned}$$

Then we have

$$\begin{aligned}
\sum_{i=1}^{\infty}|x_i - x_{i-1}| &\leq \sum_{i=1}^{\infty}\lambda^{i-1}|x_1 - x_0| \\
&= |x_1 - x_0|\sum_{i=1}^{\infty}\lambda^{i-1} \\
&= \frac{1}{1-\lambda}|x_1 - x_0|
\end{aligned}$$

since $0 \leq \lambda < 1$. This show that $\sum_{i=1}^{\infty}|x_i - x_{i-1}|$ is bounded, hence it converges. This shows that the sequence $\{x_k\}_{k\geq 0}$ converges for any initial point x_0.

Let $\lim\limits_{k\to\infty} x_k = x_*$. Then we have showed that x_* would be a fixed point of f. (note: contractivenedd implies continuity)

To prove the uniqueness, let x and y both be fixed points of f. Then

$$|x - y| = |f(x) - f(y)| \leq \lambda|x - y|.$$

Since $\lambda < 1$, this forces $|x - y| = 0$. That is, $x = y$. ■

Theorem 5.5 *If $f \in C[a,b]$ such that $a \leq f(x) \leq b$ for all $x \in [a,b]$, then f has a fixed point in $[a,b]$. Suppose, in addition, that $f'(x)$ exists in (a,b) and there exists a positive constant $M < 1$ such that $|f'(x)| \leq M < 1$ for all $x \in (a,b)$. Then the fixed point is unique.*

Proof: If $f(a) = a$ or $f(b) = b$, then a or b is a fixed point of f and we are done. Otherwise, it must be $g(a) > a$ and $g(b) < b$. Let $h(x) = f(x) - x$. Then $h \in C[a, b]$ since $f \in C[a, b]$, and $h(a) = f(a) - a > 0$ and $h(b) = f(b) - b < 0$. By the itermediate value theorem, there exists $x_* \in [a, b]$ such that $h(x_*) = 0$. This implies that $f(x_*) - x_* = 0$ and $f(x_*) = x_*$. Hence f has a fixed point x_* in $[a, b]$.

Suppose that $p \neq q$ are both fixed points of f in $[a, b]$. By the Mean-Value theorem, there exists ξ between p and q such that

$$f'(\xi) = \frac{f(p) - f(q)}{p - q} = \frac{p - q}{p - q} = 1.$$

However, this contradicts to the assumption that $f'(x) \leq M < 1$ for all x in $[a, b]$. Therefore the fixed point of f is unique. ∎

5.5.2 Convergence Analysis

For the underlined function $f : D \to D$, we make the following assumptions:

1. f has a fixed point x_*, and the sequence $\{x_k\}_{k \geq 0}$ is generated by the iteration
$$x_{k+1} = f(x_k), \quad k = 0, 1, \ldots,$$
with an arbitrary initial point x_0;
2. f' exists, is continuous, and $|f'(x)| < 1$ for all x in the domain of f;
3. m is a positive integer such that $f^{(i)}(x_*) = 0$ for $i = 1, \ldots, m$, but $f^{(m)}(x_*) \neq 0$.

Let $e_k = x_k - x_*$ be the error at the k-th iteration. Then by the Mean-Value theorem

$$e_{k+1} = x_{k+1} - x_* = f(x_k) - f(x_*) = f'(\xi_k)(x_k - x_*) = f'(\xi_k)e_k, \tag{5.47}$$

where ξ_k is between x_k and x_*. Since $|f'(\xi_k)| < 1$ by assumption, it ensures the error decreases in magnitude.

When x_k is close to x_*, i.e., e_k is small, ξ_k will be close to x_*, and $f'(\xi_k)$ will be close to $f'(x_*)$ since f' is continuous. Hence () can be written as

$$e_{k+1} \approx f'(x_*)e_k,$$

and we can expect rapid convergence if $f'(x_*)$ is small.

Using Taylor's theorem, we have

$$\begin{aligned}
e_{k+1} &= x_{k+1} - x_* \\
&= f(x_k) - f(x_*) \\
&= f(x_* + e_k) - f(x_*) \\
&= \left(f(x_*) + f'(x_*)e_k + \frac{1}{2}f''(x_*)e_k^2 + \cdots + \frac{1}{(m-1)!}f^{(m-1)}(x_*)e_k^{m-1} + \frac{1}{m!}f^{(m)}(\eta_k)e_k^m \right) - f(x_*) \\
&= \frac{1}{m!}f^{(m)}(\eta_k)e_k^m,
\end{aligned}$$

where η_k is between x_* and x_k. Since $\lim_{k\to\infty} x_k = x_*$,

$$\eta_k \to x_* \Longrightarrow f^{(m)}(\eta_k) \to f^{(m)}(x_*).$$

Therefore we have

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^m} = \frac{1}{m!} f^{(m)}(x_*). \tag{5.48}$$

This shows that the convergence rate of the functional iteration is m.

Since Newton's method is a special case of functional iteration, we can also show the quadratic convergence of Newton's method using this analysis. Suppose Newton's method is applied to solving $g(x) = 0$. Let $f(x) = x - \frac{g(x)}{g'(x)}$. Then

$$f'(x_*) = 1 - \frac{[g'(x_*)]^2 - g(x_*)g''(x_*)}{[g'(x_*)]^2} = \frac{g(x_*)g''(x_*)}{[g'(x_*)]^2} = 0$$

since the fixed point of f is a zero of g. The second derivative of f at x_*

$$f''(x_*) = \frac{1}{[g'(x_*)]^4}\left([g'(x_*)]^2(g(x_*)g'''(x_*) + g''(x_*)g'(x_*)) - 2g(x_*)g'(x_*)[g''(x_*)]^2\right)$$

is usually not zero. Hence it shows that Newton's method has quadratic convergence rate.

Theorem 5.6 *Suppose that $f \in C[a,b]$ such that $a \le f(x) \le b$ for all $x \in [a,b]$, and that $f'(x)$ exists on (a,b) with $f'(x) \le M < 1$ for some constant M. If $x_0 \in [a,b]$ is arbitrary, then the sequence $\{x_k\}_{k\ge 0}$ generated by the functional iteration $x_{k+1} = f(x_k)$ converges to the unique fixed point x_* of f.*

Proof: By Theorem (), the fixed point x_* of f exists and is unique.

Since $a \le f(x) \le b$ for all $x \in [a,b]$,

$$a \le x_k = f(x_{k-1}) \le b, \quad \text{for all } k = 1, 2, \ldots.$$

By the Mean-Value theorem, there exists $\xi_k \in (a,b)$ such that

$$f'(\xi_k) = \frac{f(x_k) - f(x_*)}{x_k - x_*}.$$

Thus

$$\begin{aligned} |e_k| = |x_k - x_*| &= |f'(\xi_k)||f(x_k) - f(x_*)| \\ &= |f'(\xi_k)||x_{k-1} - x_*| \\ &\le M|e_{k-1}| \le M^2|e_{k-2}| \le \cdots \le M^k|e_0| = M^k|x_0 - x_*|. \end{aligned}$$

Since $M < 1$, $M^k \to 0$ as $k \to \infty$. Therefore

$$\lim_{k\to\infty} |x_k - x_*| \le \lim_{k\to\infty} M^k|x_0 - x_*| = 0 \quad \Longrightarrow \quad \lim_{k\to\infty} x_k = x_*.$$

■

Corollary 5.1 *Suppose f satisfies the conditions of the previous theorem and the sequence $\{x_k\}_{k\geq 0}$ is produced by the functional iteration $x_{k+1} = g(x_k)$ with an arbitrary initial point x_0. Then*

$$|e_k| = |x_k - x_*| \leq M^k \max\{x_0 - a, b - x_0\} \tag{5.49}$$

and

$$|e_k| = |x_k - x_*| \leq \frac{M^k}{1-M}|x_0 - x_1| = \frac{M^k}{1-M}|e_0|. \tag{5.50}$$

Proof: From the proof of the previous theorem,

$$|e_k| = |x_k - x_*| \leq M^k|x_0 - x_*| \leq M^k \max\{x_0 - a, b - x_0\}.$$

Since

$$\begin{aligned}
|x_{k+1} - x_k| &= |f(x_k) - f(x_{k-1})| \\
&= |f'(\xi_k)||x_k - x_{k-1}| \\
&\leq M|x_k - x_{k-1}| \\
&\leq M^2|x_{k-1} - x_{k-2}| \\
&\vdots \\
&\leq M^k|x_1 - x_0|,
\end{aligned}$$

for $m > k \leq 1$,

$$\begin{aligned}
|x_m - x_k| &= |x_m - x_{m-1} + x_{m-1} - x_{m-2} + \cdots + x_{k+1} - x_k| \\
&\leq |x_m - x_{m-1}| + |x_{m-1} - x_{m-2}| + \cdots + |x_{k+1} - x_k| \\
&\leq M^{m-1}|x_1 - x_0| + M^{m-2}|x_1 - x_0| + \cdots + M^k|x_1 - x_0| \\
&= M^k(M^{m-1-k} + M^{m-2-k} + \cdots + M + 1)|x_1 - x_0|.
\end{aligned}$$

Since $\lim_{m\to\infty} x_m = x_*$,

$$\begin{aligned}
|x_k - x_*| &= \lim_{m\to\infty} |x_k - x_m| \\
&\leq M^k|x_1 - x_0| \lim_{m\to\infty}(M^{m-1-k} + M^{m-2-k} + \cdots + M + 1) \\
&= M^k|x_1 - x_0| \sum_{i=1}^{\infty} M^i \\
&= \frac{M^k}{1-M}|x_1 - x_0|.
\end{aligned}$$

■

Chapter 6

Interpolation

In this chapter, we consider the interpolation problem: Suppose we do not know the function f, but a few information (data) about f, now we try to compute a function g that approximates f.

6.1 Polynomial Interpolation

The polynomial interpolation problem, also called Lagrange interpolation, can be described as follows: Given $n+1$ data points (x_i, y_i), $i = 0, 1, \ldots, n$, find a polynomial P of lowest possible degree such that

$$y_i = P(x_i), \qquad i = 0, 1, \ldots, n. \tag{6.1}$$

Such a polynomial is said to interpolate the data. Here y_i may be the value of some unknown function f at x_i, i.e., $y_i = f(x_i)$.

One reason for considering the class of polynomials in approximation of functions is that they uniformly approximate continuous function.

Theorem 6.1 (Weierstrass Approximation Theorem) *Suppose that f is defined and continuous on $[a, b]$. For any $\varepsilon > 0$, there exists a polynomial $P(x)$, defined on $[a, b]$, with the property that*

$$|f(x) - P(x)| < \varepsilon, \qquad \textit{for all } x \textit{ in } [a, b].$$

Another important reason for considering the class of polynomials in approximation of functions is that the derivatives and indefinite integral of a polynomial are easy to determine and are also polynomials. For these reasons, polynomials are often used for approximating continuous functions.

6.1.1 Existence And Uniqueness

The theorem that governs the polynomial interpolation problem stats

Theorem 6.2 *If* (x_i, y_i), $x_i, y_i \in \mathbb{R}$, $i = 0, 1, \ldots, n$, *are* $n+1$ *distinct pairs of data point, then there is a unique polynomial* P_n *of degree at most* n *such that*

$$P_n(x_i) = y_i, \quad (0 \le i \le n). \tag{6.2}$$

Proof: Existence: Proof by mathematical induction. The theorem clearly holds for $n = 0$ (only one data point (x_0, y_0)) since one may choose the constant polynomial $P_0(x) = y_0$ for all x. Assume that the theorem holds for $n \le k$, that is, there is a polynomial P_k, $\deg(P_k) \le k$, such that $y_i = P_k(x_i)$, for $0 \le i \le k$. Next we try to construct a polynomial of degree at most $k+1$ to interpolate (x_i, y_i), $0 \le i \le k+1$. Let

$$P_{k+1}(x) = P_k(x) + c(x - x_0)(x - x_1)\cdots(x - x_k),$$

where

$$c = \frac{y_{k+1} - P_k(x_{k+1})}{(x_{k+1} - x_0)(x_{k+1} - x_1)\cdots(x_{k+1} - x_k)}.$$

Since x_i are distinct, the polynomial $P_{k+1}(x)$ is well-defined and $\deg(P_{k+1}) \le k+1$. It is easy to verify that

$$P_{k+1}(x_i) = y_i, \quad 0 \le i \le k+1.$$

Uniqueness: Suppose there are two such polynomials P_n and Q_n satisfying (6.2). Define

$$S_n(x) = P_n(x) - Q_n(x).$$

Since both $\deg(P_n) \le n$ and $\deg(Q_n) \le n$, $\deg(S_n) \le n$. Moreover

$$S_n(x_i) = P_n(x_i) - Q_n(x_i) = y_i - y_i = 0,$$

for $0 \le i \le n$. This means that S_n has at least $n+1$ zeros, it therefore must be $S_n = 0$. Hence $P_n = Q_n$. ∎

6.1.2 Naive Approach for Polynomial Interpolation

Suppose we require the interpolating polynomial to be expressed in power of x as

$$P_n(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_n x^n. \tag{6.3}$$

The straightforward approach for solving the polynomial interpolation is try to determine the coefficients $c_0, c_1, \ldots c_n$ using the conditions

$$P_n(x_i) = y_i, \qquad i = 0, 1, 2, \ldots, n.$$

These lead to

$$\begin{aligned}
c_0 + c_1 x_1 + c_2 x_1^2 + \cdots + c_n x_1^n &= y_0, \\
c_0 + c_1 x_2 + c_2 x_2^2 + \cdots + c_n x_2^n &= y_1, \\
&\vdots \\
c_0 + c_1 x_n + c_2 x_n^2 + \cdots + c_n x_n^n &= y_n,
\end{aligned}$$

which is equivalent to the system of linear equations:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}. \tag{6.4}$$

The coefficient matrix is called Vandermonde matrix. It can be proved that

$$\det \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & & & & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix} = \prod_{0 \le j < k \le n} (x_k - x_j). \tag{6.5}$$

Since x_i are distinct, the Vandermonde system is nonsingular, and solving this system gives the coefficients $c_0, c_1, \ldots, c_n$ which solves the polynomial interpolation problem (6.1). However, a Vandermonde matrix is often very ill-conditioned, thus the computed solutions c_i will be inaccurate. Moreover, the amount of computational cost, including forming the matrix, factorization, and triangular substitutions, is excessive. Therefore, this approach is not recommended.

6.1.3 Lagrange Form and Neville's Method

Next we present an alternative form for the interpolating polynomial $P(x)$ associated with the $n+1$ distinct data points (x_i, y_i) for $0 \le i \le n$. It is important to remember that there is one and only one polynomial of degree less than or equal to n that solves the problem (6.1), various approaches produce the same polynomial in different forms.

Polynomial interpolation in Lagrange form expresses the polynomial as

$$P_n(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x) = \sum_{k=0}^{n} y_k L_k(x), \tag{6.6}$$

where

$$L_k(x) = \frac{(x - x_0)\cdots(x - x_{k-1})(x - x_{k+1})\cdots(x - x_n)}{(x_k - x_0)\cdots(x_k - x_{k-1})(x_k - x_{k+1})\cdots(x_k - x_n)} = \prod_{\substack{i=0 \\ i \ne k}}^{n} \frac{x - x_i}{x_k - x_i}, \tag{6.7}$$

are known as cardinal polynomials.

One advantage of using the Lagrange form of the interpolating polynomial is that it can be written down at once, since the coefficients in the Lagrange formula are the given y_i. This fact will be useful in numerical integration when quadrature formulas are constructed.

Example 6.1 *Given the following 4 data points,*

x_i	*0*	*1*	*3*	*5*
y_i	*1*	*2*	*6*	*7*

find a polynomial in Lagrange form to interpolate these data.

Sol: The cardinal functions are

$$
\begin{aligned}
L_0(x) &= \frac{(x-1)(x-3)(x-5)}{(0-1)(0-3)(0-5)} = -\frac{1}{15}(x-1)(x-3)(x-5),\\
L_1(x) &= \frac{(x-0)(x-3)(x-5)}{(1-0)(1-3)(1-5)} = \frac{1}{8}x(x-3)(x-5),\\
L_2(x) &= \frac{(x-0)(x-1)(x-5)}{(3-0)(3-1)(3-5)} = -\frac{1}{12}x(x-1)(x-5),\\
L_3(x) &= \frac{(x-0)(x-1)(x-3)}{(5-0)(5-1)(5-3)} = \frac{1}{40}x(x-1)(x-3).
\end{aligned}
$$

The interpolating polynomial in the Lagrange form is

$$P_3(x) = L_0(x) + 2L_1(x) + 6L_2(x) + 7L_3(x).$$

■

One difficulty for the interpolating polynomial in the Lagrange form is that if more data points are added to the interpolation problem, all the cardinal functions have to be recalculated. We will now derive the interpolating polynomials in a manner that uses the previous calculations to greater advantage.

Definition 6.1 *Let (x_i, y_i), $0 \le i \le n$, where $y_i = f(x_i)$ for some unknown function f, be $(n+1)$ given distinct data points. Suppose that $m_1, m_2, \ldots, m_k$ are k distinct integers with $0 \le m_i \le n$ for each i. The Lagrange polynomial that interpolates f at the k points $x_{m_1}, x_{m_2}, \ldots, x_{m_k}$ is denoted $P_{m_1,m_2,\ldots,m_k}(x)$.*

Theorem 6.3 *Let f be defined at distinct points $x_0, x_1, \ldots, x_k$, and $0 \le, i, j \le k$, $i \ne j$. Then*

$$P_{0,1,\ldots,k}(x) = \frac{(x-x_j)}{(x_i-x_j)}P_{0,1,\ldots,j-1,j+1,\ldots,k}(x) - \frac{(x-x_i)}{(x_i-x_j)}P_{0,1,\ldots,i-1,i+1,\ldots,k}(x) \tag{6.8}$$

describes the k-th Lagrange polynomial that interpolates f at the $k+1$ points $x_0, x_1, \ldots, x_k$.

Proof: Since $\deg(P_{0,1,\ldots,j-1,j+1,\ldots,k}) \le k-1$ and $\deg(P_{0,1,\ldots,i-1,i+1,\ldots,k}) \le k-1$, by definition $\deg(P_{0,1,\ldots,k}) \le k$. If $r \ne i, j$, then

$$
\begin{aligned}
P_{0,1,\ldots,k}(x_r) &= \frac{(x_r-x_j)}{(x_i-x_j)}P_{0,1,\ldots,j-1,j+1,\ldots,k}(x_r) - \frac{(x_r-x_i)}{(x_i-x_j)}P_{0,1,\ldots,i-1,i+1,\ldots,k}(x_r)\\
&= \frac{(x_r-x_j)}{(x_i-x_j)}f(x_r) - \frac{(x_r-x_i)}{(x_i-x_j)}f(x_r) = f(x_r).
\end{aligned}
$$

Moreover

$$P_{0,1,\ldots,k}(x_i) = \frac{(x_i-x_j)}{(x_i-x_j)}P_{0,1,\ldots,j-1,j+1,\ldots,k}(x_i) - \frac{(x_i-x_i)}{(x_i-x_j)}P_{0,1,\ldots,i-1,i+1,\ldots,k}(x_i) = f(x_i)$$

and

$$P_{0,1,\ldots,k}(x_j) = \frac{(x_j - x_j)}{(x_i - x_j)} P_{0,1,\ldots,j-1,j+1,\ldots,k}(x_j) - \frac{(x_j - x_i)}{(x_i - x_j)} P_{0,1,\ldots,i-1,i+1,\ldots,k}(x_j) = f(x_j).$$

Therefore $P_{0,1,\ldots,k}(x)$ agrees with f at all points $x_0, x_1, \ldots, x_k$. By the uniqueness theorem, $P_{0,1,\ldots,k}(x)$ is the k-th Lagrange polynomial that interpolates f at the $k+1$ points $x_0, x_1, \ldots, x_k$. ∎

The theorem implies that the Lagrange interpolating polynomial can be generated recursively. The procedure is called the **Neville's method**. For example, the polynomials can be computed in a manner as shown in the following table.

x_0	$P_0 = Q_{0,0}$				
x_1	$P_1 = Q_{1,0}$	$P_{0,1} = Q_{1,1}$			
x_2	$P_2 = Q_{2,0}$	$P_{1,2} = Q_{2,1}$	$P_{0,1,2} = Q_{2,2}$		
x_3	$P_3 = Q_{3,0}$	$P_{2,3} = Q_{3,1}$	$P_{1,2,3} = Q_{3,2}$	$P_{0,1,2,3} = Q_{3,3}$	
x_4	$P_4 = Q_{4,0}$	$P_{3,4} = Q_{4,1}$	$P_{2,3,4} = Q_{4,2}$	$P_{1,2,3,4} = Q_{4,3}$	$P_{0,1,2,3,4} = Q_{4,4}$

For ease of notation, we denote, in the table

$$Q_{i,j} = P_{i-j,i-j+1,\ldots,i-1,i}.$$

Hence $Q_{i,j}$, $0 \le j \le i$, denotes the interpolating polynomial of degree j on the $j+1$ points $x_{i-j}, x_{i-j+1}, \ldots, x_{i-1}, x_i$.

Note that the Neville's method is usually used to generated successively higher degree polynomial approximations at a specified point, but not to generate the polynomials themselves.

6.1.4 Newton's Form of Polynomial Interpolation

Since there is one and only one polynomial of degree $\le n$ that takes prescribed values at given $n+1$ distinct points, various algorithms will produce the same polynomial but in different forms. For example, the polynomial can be constructed as a linear combination of the basic polynomials $1, x, x^2, \ldots, x^n$. For numerical work, it is preferable to use the following basis

$$\begin{aligned}
q_0(x) &= 1, \\
q_1(x) &= (x - x_0), \\
q_2(x) &= (x - x_0)(x - x_1), \\
&\vdots \\
q_n(x) &= (x - x_0)(x - x_1)\cdots(x - x_{n-1}),
\end{aligned}$$

and express $P(x)$ as

$$\begin{aligned}
P_n(x) &= c_0 + c_1(x - x_0) + c_2(x - x_0)(x - x_1) + \cdots + c_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}). \\
&= \sum_{k=0}^{n} c_k q_k(x) \qquad (6.9)
\end{aligned}$$

This polynomial is called the interpolating polynomial in Newton's form.

The interpolation conditions

$$P_n(x_i) = \sum_{k=0}^{n} c_k q_k(x_i) = y_i, \qquad i = 0, 1, \ldots, n,$$

give rise to $n+1$ equations with $n+1$ parameters $c_0, c_1, \ldots, c_n$ to be determined

$$\begin{aligned} c_0 &= y_0, \\ c_0 + c_1(x_1 - x_0) &= y_1, \\ c_0 + c_1(x_2 - x_0) + c_2(x_2 - x_0)(x_2 - x_1) &= y_2, \\ &\vdots \\ c_0 + c_1(x_n - x_0) + c_2(x_n - x_0)(x_n - x_1) + \cdots + c_n(x_n - x_0)\cdots(x_n - x_{n-1}) &= y_n. \end{aligned}$$

In matrix equation, we have a triangular linear system

$$\begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 1 & x_1 - x_0 & 0 & \cdots & 0 \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n - x_0 & (x_n - x_0)(x_n - x_1) & \cdots & (x_n - x_0)\cdots(x_n - x_{n-1}) \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}. \tag{6.10}$$

Use forward substitution, we can solve for $c_0, c_1, \ldots, c_n$. Computational work involved in this approach includes forming and solving the system.

Example 6.2 *Given the following* 4 *points* $(n = 3)$

x_i	*0*	*1*	*3*	*5*
y_i	*1*	*2*	*6*	*7*

find a polynomial of degree 3 *in Newton's form to interpolate these data.*

Sol: Write

$$\begin{aligned} P(x) &= c_0 + c_1(x-0) + c_2(x-0)(x-1) + c_3(x-0)(x-1)(x-3) \\ &= c_0 + c_1 x + c_2 x(x-1) + c_3 x(x-1)(x-3). \end{aligned}$$

Plug in the interpolation conditions,

$$\begin{aligned} P(0) &= c_0 = 1. \\ P(1) &= c_0 + c_1 = 2. \\ P(3) &= c_0 + 3c_1 + 6c_2 = 6. \\ P(5) &= c_0 + 5c_1 + 20c_2 + 40c_3 = 7, \end{aligned}$$

which lead to

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 3 & 6 & 0 \\ 1 & 5 & 20 & 40 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 6 \\ 7 \end{bmatrix} \quad \Longrightarrow \quad \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ \frac{1}{3} \\ -\frac{17}{120} \end{bmatrix}.$$

Thus

$$P(x) = 1 + x + \frac{1}{3}x(x-1) - \frac{17}{120}x(x-1)(x-3).$$

■

6.1.5 Divided Differences Scheme

With proper arrangement, the triangular linear system in the previous subsection can be solved with the formulations

$$\begin{aligned} c_0 &= f(x_0) \\ c_1 &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} \\ c_2 &= \frac{1}{x_2 - x_0}\left[\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0}\right] \\ &\vdots \end{aligned}$$

and so on. An efficient procedure, called the divided-difference method, can be derived for computing the $n+1$ coefficients $c_0, c_1, \cdots, c_n$ in the Newton's form interpolating polynomial.

First we introduce the divided-difference notation. The zero divided difference of the function f with respect to x_i,

$$f[x_i] = f(x_i), \tag{6.11}$$

is simply the value of f at x_i. The first divided difference of f with respect to x_i and x_{i+1} is denoted and defined as

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}. \tag{6.12}$$

The second divided difference of f is defined as

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}. \tag{6.13}$$

In general, the k-th divided difference relative to $x_i, x_{i+1}, \ldots, x_{i+k}$ is given by

$$f[x_i, x_{i+1}, \ldots, x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i}. \tag{6.14}$$

As might be expected from the evaluation of $c_0, c_1, \ldots, c_n$ in the Newton form interpolating polynomial, the required coefficients are given by

$$c_k = f[x_0, x_1, \ldots, x_k], \qquad k = 0, 1, \ldots, n. \tag{6.15}$$

Therefore the interpolating polynomial in Newton's form can be expressed as

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0)(x - x_1)\cdots(x - x_{k-1}). \tag{6.16}$$

This formula is known as the Newton's interpolatory divided-difference formula. The determination of the divided differences is best described by using tabular form as shown in the following table as an example.

x_i	$k=0$	$k=1$	$k=2$	$k=3$
x_0	$f[x_0]$			
		$> f[x_0,x_1]$		
x_1	$f[x_1]$		$> f[x_0,x_1,x_2]$	
		$> f[x_1,x_2]$		$> f[x_0,x_1,x_2,x_3]$
x_2	$f[x_2]$		$> f[x_1,x_2,x_3]$	
		$> f[x_2,x_3]$		
x_3	$f[x_3]$			

Example 6.3 *Given the following* 4 *points* $(n = 3)$

x_i	*0*	*1*	*3*	*5*
y_i	*1*	*2*	*6*	*7*

find a polynomial of degree 3 *in Newton's form to interpolate these data.*

Sol:

x_i	$k=0$	$k=1$	$k=2$	$k=3$
0	1			
		> 1		
1	2		$> \frac{1}{3}$	
		> 2		$> -\frac{17}{120}$
3	6		$> -\frac{3}{8}$	
		$> \frac{1}{2}$		
5	7			

So,

$$P(x) = 1 + x + \frac{1}{3}x(x-1) - \frac{17}{120}x(x-1)(x-3).$$

■

Note that x_i can be reordered, but must be distinct. When the order of x_i are changed, one obtains the same polynomial but in different form.

Example 6.4 *Given the following* 4 *points* $(n = 3)$

x_i	3	1	5	0
y_i	6	2	7	1

find a polynomial of degree 3 *in Newton's form to interpolate these data.*

Sol:

$$\begin{array}{c|cccc}
x_i & k=0 & k=1 & k=2 & k=3 \\ \hline
3 & 6 & & & \\
 & & > 2 & & \\
1 & 2 & & > -\frac{3}{8} & \\
 & & > \frac{5}{4} & & > -\frac{17}{120} \\
5 & 7 & & > \frac{1}{20} & \\
 & & > \frac{6}{5} & & \\
0 & 1 & & &
\end{array}$$

So,

$$P(x) = 6 + 2(x-3) - \frac{3}{8}(x-3)(x-1) - \frac{17}{120}(x-3)(x-1)(x-5).$$

■

Note that $c_0, c_1, \ldots, c_{n-1}$ are changed, but c_n is unchanged, and the polynomial is expressed in different form.

Theorem 6.4 *Suppose* $f \in C^n[a,b]$ *and* $x_0, x_1, \ldots, x_n$ *are distinct numbers in* $[a,b]$. *Then there exists* $\xi \in (a,b)$ *such that*

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

Sol: Let

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x-x_0)(x-x_1)\cdots(x-x_{k-1})$$

be the interpolating polynomial of f in Newton's form. Define

$$g(x) = f(x) - P_n(x).$$

Since $P_n(x_i) = f(x_i)$ for $i = 0, 1, \ldots, n$, the function g has $n+1$ distinct zeros in $[a,b]$. By the generalized Rolle's Theorem, there exists $\xi \in (a,b)$ such that

$$g^{(n)}(\xi) = f^{(n)}(\xi) - P_n^{(n)}(\xi) = 0.$$

Note that

$$P_n^{(n)}(x) = n! f[x_0, x_1, \ldots, x_n].$$

As a consequence

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}.$$

■

An algorithm, to be introduced, for computing the divided-difference table is very efficient and is recommended as the best scheme for producing an interpolating polynomial in Newton's form. First we change the notation to

$$\begin{aligned} a_{i,k} &= f[x_i, x_{i+1}, \ldots, x_{i+k}] \\ &= \frac{f[x_{i+1}, x_{i+2}, \ldots, x_{i+k}] - f[x_i, x_{i+1}, \ldots, x_{i+k-1}]}{x_{i+k} - x_i} = \frac{a_{i+1,k-1} - a_{i,k-1}}{x_{i+k} - x_i} \end{aligned}$$

so that the divided-difference table has the entries as shown in the following.

$$\begin{array}{c|ccccccccc}
x_i & k=0 & & k=1 & & k=2 & & & & k=n \\ \hline
x_0 & a_{00} & & & & & & & & \\
 & & \rangle & a_{01} & & & & & & \\
x_1 & a_{10} & & & \rangle & a_{02} & & & & \\
 & & \rangle & a_{12} & & & \ddots & & & \\
x_2 & a_{20} & & & \rangle & a_{12} & & \ddots & & \\
 & & \rangle & a_{21} & & & & & \cdots & a_{0,n} \\
x_3 & a_{30} & & & \vdots & & & \ddots & & \\
 & & \vdots & \vdots & & & \ddots & & & \\
\vdots & \vdots & & & & \ddots & & & & \\
 & & \rangle & a_{n-1,1} & & & & & & \\
x_n & a_{n,0} & & & & & & & &
\end{array}$$

The interpolating polynomial, of course, is

$$P_n(x) = a_{00} + \sum_{k=1}^{n} a_{0,k} \prod_{j=0}^{k-1} (x - x_j). \tag{6.17}$$

Algorithm 6.1 (Divided-Difference Table) *Given $n+1$ distinct data points (x_i, y_i), $i = 0, 1, \ldots, n$, this algorithm computes the coefficients of the interpolating polynomial in Newton's form by constructing the divided-difference table.*

> **for** $i = 0, 1, \ldots n$ **do**
> $a(i,0) = y(i) = f(x(i))$
> **end for**
> **for** $k = 1, \ldots, n$ **do**
>   **for** $i = 0, 1, \ldots, n-k$ **do**
> $a(i,k) = \dfrac{a(i+1,k-1) - a(i,k-1)}{x(i+k) - x(i)}$
>   **end for**
> **end for**

This algorithm uses one 2-d array of $(n+1)\times(n+1)$ for the storage of all a_{ij}. The inner i-loop can be performed in parallel. If only the coefficients in the Newton's interpolating polynomial are required for one set of data, an improved algorithm which uses only one 1-d array of size $n+1$ can be derived. The modified algorithm, however, can not be executed in parallel.

Algorithm 6.2 (Divided-Difference Algorithm) *Given $n+1$ distinct points (x_i, y_i), $i = 0, 1, \ldots, n$, this algorithm computes the coefficients of the interpolating polynomial in Newton's form using an 1-d array of size $n+1$..*

> **for** $i = 0, 1, \ldots, n$ **do**
> $\quad c(i) = y(i) = f(x(i))$
> **end for**
> **for** $k = 1, \ldots, n$ **do**
> $\quad$ **for** $i = n, n-1, \ldots, k$ **do**
> $$c(i) = \frac{c(i) - c(i-1)}{x(i) - x(i-k)}$$
> $\quad$ **end for**
> **end for**

Newton's interpolatory divided-difference formula can be expressed in a simplified form when $x_0, x_1, \ldots, x_n$ are arranged consecutively with equal spacing. Let

$$h = \frac{x_n - x_0}{n} = x_{i+1} - x_i, \qquad i = 0, 1, \ldots, n-1.$$

Then each $x_i = x_0 + i * h$, $i = 0, 1, \ldots, n$. For any $x \in [a, b]$, write

$$x = x_0 + s * h, \qquad s \in \mathbb{R}.$$

Hence $x - x_i = (s - i) * h$ and

$$\begin{aligned} P_n(x) &= f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](x - x_0)(x - x_1)\cdots(x - x_{k-1}) \\ &= f(x_0) + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k](s-0) * h * (s-1) * h * \cdots (s-k+1) * h \\ &= f(x_0) + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k] s(s-1)\cdots(s-k+1)h^k \\ &= f(x_0) + \sum_{k=1}^{n} f[x_0, x_1, \ldots, x_k] k! \binom{s}{k} h^k, \end{aligned} \tag{6.18}$$

where the binomial formula

$$\binom{s}{k} = \frac{s(s-1)\cdots(s-k+1)}{k!}$$

is used. This formula is called the Newton forward divided-difference formula.

Next we introduce the forward difference notation $\triangle$

$$\triangle f(x_i) = f(x_{i+1}) - f(x_i) \tag{6.19}$$

and

$$\triangle^k f(x_i) = \triangle^{k-1} f(x_{i+1}) - \triangle^{k-1} f(x_i) = \triangle\left(\triangle^{k-1} f(x_i)\right), \tag{6.20}$$

for $i = 0, 1, \ldots, n-1$. With this notation,

$$\begin{aligned} f[x_0, x_1] &= \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{1}{h}\triangle f(x_0) \\ f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{\frac{1}{h}\triangle f(x_1) - \frac{1}{h}\triangle f(x_0)}{2h} = \frac{1}{2h^2}\triangle^2 f(x_0), \end{aligned}$$

and, in general

$$f[x_0, x_1, \ldots, x_k] = \frac{1}{k!\, h^k}\triangle^k f(x_0). \tag{6.21}$$

Using the forward difference $\triangle$ notation, the Newton's forward divided-difference formula can be expressed as

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} \binom{s}{k} \triangle^k f(x_0). \tag{6.22}$$

This formula is called the Newton forward-difference formula.

If the interpolation nodes are arranged as $x_n, x_{n-1}, \ldots, x_0$, a formula for the interpolating polynomial similar to results

$$P_n(x) = f[x_n] + \sum_{k=1}^{n} f[x_n, x_{n-1}, \ldots, x_{n-k}](x - x_n)(x - x_{n-1}) \cdots (x - x_{n-k+1}). \tag{6.23}$$

If the nodes are equally spaced with

$$h = \frac{x_n - x_0}{n}, \qquad x_i = x_n - (n-i) * h, \qquad x = x_n + s * h,$$

then

$$\begin{aligned} P_n(x) &= f[x_n] + \sum_{k=1}^{n} f[x_n, x_{n-1}, \ldots, x_{n-k}] s * h * (s+1) * h * \cdots * (s+k-1) * h \\ &= f(x_n) + \sum_{k=1}^{n} f[x_n, x_{n-1}, \ldots, x_{n-k}](-1)^k k! \binom{-s}{k} h^k \end{aligned} \tag{6.24}$$

where the binomial formula is extended to include all real value s

$$\binom{-s}{k} = \frac{-s(-s-1)\cdots(-s-k+1)}{k!} = (-1)^k \frac{s(s+1)\cdots(s+k-1)}{k!}.$$

This form is called the Newton backward divided-difference formula. Similarly, we can introduce the backward-difference notation

$$\nabla^k f(x_i) = \nabla^{k-1} f(x_i) - \nabla^{k-1} f(x_{i-1}) = \nabla\left(\nabla^{k-1} f(x_i)\right), \tag{6.25}$$

then

$$f[x_n, x_{n-1}] = \frac{1}{h}\nabla f(x_n), \qquad f[x_n, x_{n-1}, x_{n-2}] = \frac{1}{2h^2}\nabla^2 f(x_n),$$

and, in general,

$$f[x_n, x_{n-1}, \ldots, x_{n-k}] = \frac{1}{k!\,h^k}\nabla^k f(x_n). \tag{6.26}$$

Using this backward-difference ∇ notation, the Newton backward divided-difference formula can be written as

$$P_n(x) = f(x_0) + \sum_{k=1}^{n} (-1)^k \binom{-s}{k} \nabla^k f(x_k). \tag{6.27}$$

This is called the Newton backward-difference formula.

6.1.6 Error Analysis for Polynomial Interpolation

Theorem 6.5 *Suppose $f(x) \in C^{n+1}[a,b]$, and $P(x)$ is a polynomial of degree $\leq n$ that interpolates f at $n+1$ given distinct points $x_0, x_1 \ldots, x_n$ in $[a,b]$. Then for each $x \in [a,b]$, there exists a point $\xi_x \in (a,b)$ such that*

$$f(x) - P(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{k=0}^{n} (x - x_k). \tag{6.28}$$

Proof: Note first that if $x = x_i$ for some $0 \leq i \leq n$, then $P(x) = f(x)$, the theorem holds for any arbitrary $\xi_x \in (a,b)$. Suppose that $x \in [a,b]$ but $x \neq x_i$ for all $i = 0, 1, \ldots, n$. Define a new function

$$g(t) = f(t) - P(t) - [f(x) - P(x)] \prod_{k=0}^{n} \frac{(t - x_k)}{(x - x_k)}, \qquad t \in [a,b].$$

Since $f \in C^{n+1}[a,b]$, $P \in C^{\infty}[a,b]$, and $x \neq x_i,\ \forall\, i$, it follows that $g(t)$ is well-defined for all $t \in [a,b]$, and $g \in C^{n+1}[a,b]$. Then

$$g(x_i) = f(x_i) - P(x_i) - [f(x) - P(x)] \prod_{k=0}^{n} \frac{(x_i - x_k)}{(x - x_k)} = 0, \qquad i = 0, 1, \ldots, n.$$

Moreover

$$g(x) = f(x) - P(x) - [f(x) - P(x)] \prod_{k=0}^{n} \frac{(x - x_k)}{(x - x_k)} = 0.$$

Thus $g \in C^{n+1}[a,b]$ and g has $n+2$ zeros in $[a,b]$. By the generalized Rolle's theorem, there exists $\xi_x \in (a,b)$ for which $g^{(n+1)}(\xi_x) = 0$. Hence

$$\begin{aligned} 0 = g^{(n+1)}(\xi_x) &= f^{(n+1)}(\xi_x) - P^{(n+1)}(\xi_x) - [f(x) - P(x)] \left[\frac{d^{n+1}}{dt^{n+1}} \prod_{k=0}^{n} \frac{(t-x_k)}{(x-x_k)} \right] \Bigg|_{t=\xi_x} \\ &= f^{(n+1)}(\xi_x) - 0 - [f(x) - P(x)] \cdot \frac{(n+1)!}{\prod (x-x_k)}. \end{aligned}$$

Therefore

$$f(x) - P(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{k=0}^{n} (x - x_k).$$

■

Example 6.5 *Suppose $f(x) = \sin x$ is approximated by an interpolating polynomial $p(x)$ of degree 9 in $[0,1]$. Estimate $|f(x) - p(x)|$, for all $x \in [0,1]$.*

Sol: With $n = 9$, and since $f(x) = \sin x$, $\left|f^{(10)}(\xi)\right| \le 1$, and $x \in [0,1]$,

$$|x - x_i| \le 1, \quad \Longrightarrow \quad \left| \prod_{i=0}^{10} (x - x_i) \right| \le 1,$$

we have

$$|f(x) - p(x)| = \frac{1}{10!} \left|f^{(10)}(\xi)\right| \left| \prod_{i=0}^{10} (x - x_i) \right| \le \frac{1}{10!}.$$

■

6.2 Hermite Interpolation

If values of a function f and some of its derivatives are to be interpolated by a polynomial, the topic is known as Birkhoff interpolation. The general problem of this type may have some intriguing difficulties associated with it because the linear system of equations from which we expect to compute the coefficients in the polynomial may be singular.

Example 6.6 *Find a polynomial p that assumes theses values:*

$$p(0) = 0, \quad p(1) = 1, \quad p'\left(\frac{1}{2}\right) = 2.$$

Sol: Since there are three conditions, we try a quadratic polynomial

$$p(x) = a + bx + cx^2.$$

Then $p'(x) = b + 2cx$. Plug in the given conditions, we have

$$\begin{aligned} a &= 0 \\ a + b + c &= 1 \\ b + c &= 2 \end{aligned}$$

However the coefficient matrix of this linear system of equations

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

is singular. There is no solution for coefficients a, b, and c. Thus there is no polynomial of degree at most 2 that solves this problem. ■

The problems in a restricted class are the ones generally known as Hermite interpolation. Consider the given $n+1$ data points $x_0 < x_1 < \cdots < x_n$, and real numbers $y_i^{(k)}$, $k = 0, 1, \ldots, m_i - 1$, $i = 0, 1, \ldots, n$, where

$$\begin{array}{cccc} y_0^{(0)} = f(x_0) & y_1^{(0)} = f(x_1) & \cdots & y_n^{(0)} = f(x_n) \\ y_0^{(1)} = f'(x_0) & y_1^{(1)} = f'(x_1) & \cdots & y_n^{(1)} = f'(x_n) \\ \vdots & \vdots & & \vdots \\ y_0^{(m_0-1)} = f^{(m_0-1)}(x_0) & y_1^{(m_1-1)} = f^{(m_1-1)}(x_1) & \cdots & y_n^{(m_n-1)} = f^{(m_n-1)}(x_n) \\ \downarrow & \downarrow & & \downarrow \\ m_0 \text{ conditions} & m_1 \text{ conditions} & \cdots & m_n \text{ conditions} \end{array}$$

for some function f. The Hermite interpolation problem for these data consists of determining a polynomial P of degree at most N, where

$$N = \left(\sum_{i=0}^{n} m_i \right) - 1. \tag{6.29}$$

satisfies the following interpolation conditions

$$P^{(k)}(x_i) = y_i^{(k)}, \qquad j = 0, 1, \ldots (m_i - 1), \quad i = 0, 1, \ldots, n. \tag{6.30}$$

In a Hermite interpolation problem, it is assumed that whenever a derivative $p^{(j)}(x_i)$ is to be prescribed at a node x_i, then $p^{(j-1)}(x_i)$, $p^{(j-2)}(x_i), \ldots p'(x_i)$, and $p(x_i)$ will also prescribed. The polynomial interpolation of the previous section is the special case where $n_i = 1$ for all $i = 0, 1, \ldots, n$.

6.2.1 Existence and Uniqueness

Theorem 6.6 *For arbitrary numbers $x_0 < x_1 < \cdots < x_n$, and $y_i^{(k)}$, $k = 0, 1, \ldots, (m_i - 1)$, $i = 0, 1, \ldots, n$, there exists a unique polynomial P of degree at most N, where $N = (\sum_{i=0}^{n} m_i) - 1$, which solves the Hermite interpolation problem (6.30).*

Proof: Since the polynomial P to be sought has degree at most N, write

$$P(x) = a_0 + a_1 x + \cdots + a_N x^N,$$

There are $N+1$ coefficents, $a_0, a_1, \ldots, a_N$, to be determined and there are exactly $N+1$ conditions given, thus we have a square system of $N+1$ linear equations in $N+1$ unknowns to solve, and we wish to show that the coefficient matrix A of this system is nonsingular.

To prove A is nonsingular, it suffices to prove that the homogeneous system $Au = 0$ has only the trivial solution $u = 0$. In the Hermite interpolation problem under discussion, the homogeneous problem is to find a polynomial P of degree at most N such that

$$P^{(k)}(x_i) = 0, \qquad k = 0, 1, \ldots, m_i - 1, \quad i = 0, 1, \ldots, n.$$

Such a polynomial has a zero of multiplicity m_i at x_i and must therefore be a multiple of the polynomial given by

$$q(x) = \prod_{i=0}^{n} (x - x_i)^{m_i}.$$

However, $\deg(q) = \sum_{i=0}^{n} m_i = N+1$ whereas P has degree at most N. We therefore conclude that $P(x) = q(x) = 0$. That is, A is nonsingular, and the Hermite interpolation problem has a unique solution. ∎

6.2.2 Lagrange Form for Hermite Interpolation

Hermite interpolating polynomial can be given explicitly in a form analogous to the polynomial interpolation formula in Lagrange form. The Hermite interpolating polynomial in Lagrange form is given by

$$P(x) = \sum_{i=0}^{n} \sum_{k=0}^{m_i - 1} y_i^{(k)} L_{ik}(x), \tag{6.31}$$

where, for $i = 0, 1, \ldots, n$,

$$L_{ik}(x) = \begin{cases} \ell_{i,m_i-1}(x), & \text{for } k = m_i - 1, \\ \ell_{ik}(x) - \sum_{j=k+1}^{m_i-1} \ell_{ik}^{(j)}(x_i) L_{ij}(x), & \text{for } k = m_i - 2, m_i - 3, \ldots, 0, \end{cases} \tag{6.32}$$

with the auxiliary polynomial

$$\ell_{ik}(x) = \frac{(x - x_i)^k}{k!} \prod_{\substack{j=0 \\ j \neq i}}^{n} \left(\frac{x - x_j}{x_i - x_j} \right)^{m_j}, \qquad 0 \le k \le m_i - 1, \quad 0 \le i \le n. \tag{6.33}$$

By induction

$$L_{ik}^{(s)}(x_t) = \begin{cases} 1, & \text{if } i = t \text{ and } k = s; \\ 0, & \text{otherwise.} \end{cases}$$

Thus $P(x)$ is indeed the desired Hermite interpolating polynomial.

6.2.3 Divided Difference Method for Hermite Interpolation

Next we illustrate how the Newton divided difference method can be extended to solve Hermite interpolation problems. First we denote the $N+1$ conditions pairs

$$(x_0, y_0^{(0)}),\ (x_0, y_0^{(1)}),\ \dots\ (x_0, y_0^{(m_0-1)}),\ (x_1, y_1^{(0)}),\ (x_1, y_1^{(1)}),\ \dots\ (x_1, y_1^{(m_1-1)}),\ \dots\ (x_n, y_1^{(m_n-1)})$$

by consecutive coordinate pairs

$$(\underbrace{z_0}_{x_0}, y_0^{(0)}),\ (\underbrace{z_1}_{x_0}, y_0^{(1)}),\ \cdots,\ (\underbrace{z_{m_0-1}}_{x_0}, y_0^{(m_0-1)}),\ (\underbrace{z_{m_0}}_{x_1}, y_1^{(0)}),\ (\underbrace{z_{m_0+1}}_{x_1}, y_1^{(1)}),\ \cdots\ (\underbrace{z_N}_{x_n}, y_n^{(m_n-1)}).$$

Note that $z_0 \leq z_1 \leq \cdots \leq z_N$. The unique Hermite interpolating polynomial in Newton's form can then be written as

$$P(x) = f[z_0] + \sum_{k=1}^{N} f[z_0, z_1, \dots, z_k](x - z_0)(x - z_1)\cdots(x - z_{k-1}). \tag{6.34}$$

The coefficient $f[z_0, z_1, \dots, z_k]$ is again computed by the divided-difference method. If $z_i \neq z_{i+k}$, then

$$f[z_i, z_{i+1}, \dots, z_{i+k}] = \frac{f[z_{i+1}, z_{i+2}, \dots, z_{i+k}] - f[z_i, z_{i+1}, \dots, z_{i+k-1}]}{z_{i+k} - z_i}. \tag{6.35}$$

However the divided-difference formula has to be modified because there may be repetitions among z_i. The extension of the definition of divided-differences to the case of repeated arguments involves transition to a limit. When $z_i = z_{i+1} = \cdots = z_{i+k}$, let

$$\xi_i < \xi_{i+1} < \cdots < \xi_{i+k}$$

be distinct coordinates and $\xi_j \to z_j$, for $i \leq j \leq i+k$. Then, by applying Theorem 6.4,

$$f[z_i, z_{i+1}, \dots, z_{i+k}] = \lim_{\substack{\xi_j \to z_j \\ i \leq j \leq i+k}} f[\xi_i, \xi_{i+1}, \dots, \xi_{i+k}] = \frac{f^{(k)}(z_i)}{k!} \tag{6.36}$$

In summary, the following algorithm computes the coefficients of the polynomial which solves the Hermite interpolation problem.

Algorithm 6.3 (Divided-Difference Method for Hermite Interpolation) *Given arbitrary $x_0 < x_1 < \cdots < x_n$, and $y_i^{(k)}$, $k = 0, 1, \dots, (m_i - 1)$, $i = 0, 1, \dots, n$, this algorithm computes the coefficients of the polynomial $P(x)$ such that $P^{(k)}(x_i) = y_i^{(k)}$, $j = 0, 1, \dots (m_i - 1)$, $i = 0, 1, \dots, n$.*

if $z_i = z_{i+k}$ **then**

$$f[z_i, z_{i+1}, \dots, z_{i+k}] = \frac{f^{(k)}(z_i)}{k!}$$

else

$$f[z_i, z_{i+1}, \dots, z_{i+k}] = \frac{f[z_{i+1}, \dots, z_{i+k}] - f[z_{i+1}, \dots, z_{i+k-1}]}{z_{i+k} - z_i}$$

end if

Example 6.7 *Given*

$$\begin{array}{cc} x_0 = 0 & x_1 = 1 \\ f(0) = 1, & f(1) = 2, \\ f'(0) = 2, & f'(1) = -2, \\ & f''(1) = -3. \end{array}$$

Find a polynomial of degree 4 *which interpolates these data.*

Sol: Constructing the divided-difference table with the previous algorithm.

z_i	$f[z_i]$				
0	1				
		> 2			
0	1		> -1		
		> 1		> -2	
1	2		> -3		$> \frac{7}{2}$
		> -2		$> \frac{3}{2}$	
1	2		$> -\frac{3}{2}$		
		> -2			
1	2				

The polynomial sought is

$$p(x) = 1 + 2x + (-1)x^2 - 2x^2(x-1) + \frac{7}{2}x^2(x-1)^2.$$

■

6.2.4 Error Analysis for Hermite Interpolation

The interpolation error which is incurred by Hermite interpolation can be estimated in the same fashion as for the usual polynomial interpolation.

Theorem 6.7 *Suppose $x_0 < x_1 < \cdots < x_n$, $x_i \in [a, b]$. If the polynomial P is of degree at most N, where*

$$N = \left(\sum_{i=0}^{n} m_i\right) - 1,$$

and satisfies the interpolation conditions

$$P^{(k)}(x_i) = f^{(k)}(x_i), \qquad k = 0, 1, \ldots, m_i - 1, \quad i = 0, 1, \ldots, n,$$

and $f \in C^{N+1}[a, b]$, then for any $x \in [a, b]$ there exists $\xi_x \in [a, b]$ such that

$$f(x) - P(x) = \frac{1}{(N+1)!} f^{(N+1)}(\xi_x) \prod_{i=0}^{n} (x - x_i)^{m_i}. \tag{6.37}$$

6.3 Spline Interpolation

The previous sections concern the approximation of an arbitrary function on a closed interval by a polynomial. However, the oscillatory nature of high-degree polynomials restricts their use. An alternative approach is to divide the interval into a collection of subintervals and construct different approximation on each subinterval. Approximation of this type is called piecewise polynomial interpolation.

The simplest piecewise polynomial approximation is piecewise linear interpolation which consists of joining a set of data points by a series of straight line segments. A disadvantage of linear function approximation is that there is no assurance of differentiability at each of the endpoints of the subintervals, which, in a geometrical context, means that the interpolating function is not smooth at these points. It is often required that the approximating function is continuously differentiable.

An alternative procedure is to use a piecewise polynomial of Hermite type. However, to use Hermite piecewise polynomials for general interpolation, we need to know the derivatives of the function being approximated, which is frequently not available.

In this section, we consider approximation using spline function that require no derivative information, except perhaps at the endpoints of the intervals on which the function is being approximated. A spline function consists of polynomial pieces on subintervals joined together with certain continuity conditions.

Definition 6.2 *Suppose the interval* $[a, b]$ *is partitioned by* $a = x_0 < x_1 < \cdots < x_n = b$. *A spline function* S_Δ *of degree* k *defined on* $[a, b]$ *is a function such that*

- S_Δ *is a polynomial of degree at most* k *on each subinterval* $[x_i, x_{i+1})$, *namely,*

$$S_\Delta = \begin{cases} S_0(x), & x \in [x_0, x_1), \\ S_1(x), & x \in [x_1, x_2), \\ \vdots & \vdots \\ S_{n-1}(x), & x \in [x_{n-1}, x_n], \end{cases}$$

  and $\deg(S_i) \le k$, $i = 0, 1, \ldots, n-1$;

- S_Δ *is* $k-1$ *continuous differentiable on* $[x_0, x_n]$, *that is,* $S_\Delta \in C^{k-1}[x_0, x_n]$.

The spline function $S_\Delta(x)$ can be viewed as a collection of n piecewise continuous polynomials of degree at most k having continuous derivatives of all order up to $k-1$. Spline functions yield smooth interpolating curves which are less likely to exhibit the large oscillations characteristics of high-degree polynomials.

Given $n+1$ distinct points $x_0 < x_1 < \cdots < x_n$ and the values, $y_i = f(x_i)$, $i = 0, 1, \ldots, n$, of function f, we try to find a spline function S_Δ to interpolate $f(x)$.

6.3.1 Cubic Spline

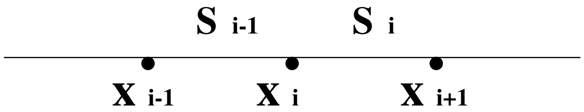
A spline of degree 3 is called a cubic spline.

- S_Δ consists of cubic polynomials ($k = 3$) on each of the subinterval $[x_i, x_{i+1})$ glued together;
- S_Δ is twice continuous differentiable on $[x_0, x_n]$. i.e., $S_\Delta \in C^2[x_0, x_n]$.

$$S_\Delta = \begin{cases} S_0(x), & x \in [x_0, x_1], \\ S_1(x), & x \in [x_1, x_2], \\ S_2(x), & x \in [x_2, x_3], \\ \vdots & \vdots \\ S_{n-1}(x), & x \in [x_{n-1}, x_n]. \end{cases}$$

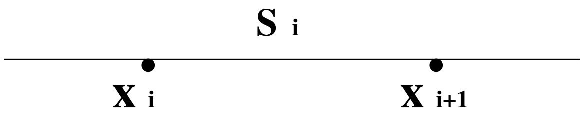
with

$$S_{i-1}(x_i) = y_i = S_i(x_i), \qquad i = 1, 2, \ldots, n-1.$$



Each $S_i(x)$ is a polynomial of degree 3, therefore, there are 4 parameters to be determined for each $S_i(x)$. Hence, $4n$ parameters in total need to be computed. On each subinterval,

$$S_i(x_i) = y_i, \qquad S_i(x_{i+1}) = y_{i+1}, \qquad i = 0, 1, 2, \ldots, n-1. \tag{6.38}$$



These give $2n$ conditions. The twice conditions differentiable conditions lead to

$$S'_{i-1}(x_i) = S'_i(x_i), \qquad S''_{i-1}(x_i) = S''_i(x_i), \qquad i = 1, 2, \ldots, n-1, \tag{6.39}$$

and give $2n-2$ conditions. Together, we have $4n-2$ conditions and $4n$ parameters to be determined. This leaves 2 degree of freedom.

Since $S_i(x)$ is a polynomial of degree 3, hence $S''_i(x)$ is a polynomial of degree 1. We may write

$$S''_i(x) = A_i x + B_i.$$

Assume

$$\begin{cases} S''_i(x_i) & = z_i, \\ S''_i(x_{i+1}) & = z_{i+1}, \end{cases} \qquad i = 0, 1, 2, \ldots, n-1. \tag{6.40}$$

Then

$$\begin{cases} A_i x_i + B_i & = z_i, \\ A_i x_{i+1} + B_i & = z_{i+1}. \end{cases}$$

Let $h_i = x_{i+1} - x_i$. Then

$$A_i = \frac{z_{i+1} - z_i}{x_{i+1} - x_i} = \frac{z_{i+1} - z_i}{h_i}.$$

and

$$B_i = z_i - A_i x_i = z_i - \frac{z_{i+1} - z_i}{h_i} x_i.$$

Hence,

$$\begin{aligned}
S_i''(x) &= \frac{z_{i+1} - z_i}{h_i} x + z_i - \frac{z_{i+1} - z_i}{h_i} x_i \\
&= \frac{z_{i+1}}{h_i} x - \frac{z_i}{h_i} x + z_i - \frac{z_i}{h_i} x_i + \frac{z_{i+1}}{h_i} x_i \\
&= \frac{z_{i+1}}{h_i}(x - x_i) - \frac{z_i}{h_i} x + z_i + \frac{z_i}{h_i}(x_{i+1} - h_i) \\
&= \frac{z_{i+1}}{h_i}(x - x_i) - \frac{z_i}{h_i} x + z_i + \frac{z_i}{h_i} x_{i+1} - z_i \\
&= \frac{z_{i+1}}{h_i}(x - x_i) + \frac{z_i}{h_i}(x_{i+1} - x) \\
&= \frac{z_{i+1}}{h_i}(x - x_i) - \frac{z_i}{h_i}(x - x_{i+1}), \qquad i = 1, 2, \ldots, n-1,
\end{aligned}$$

and

$$\begin{aligned}
S_i'(x) &= \frac{z_{i+1}}{h_i} \int (x - x_i)\, dx + \frac{z_i}{h_i} \int (x_{i+1} - x)\, dx \\
&= \frac{1}{2} \frac{z_{i+1}}{h_i} (x - x_i)^2 - \frac{1}{2} \frac{z_i}{h_i} (x_{i+1} - x)^2 + F_i,
\end{aligned}$$

Therefore

$$S_i(x) = \frac{1}{6} \frac{z_{i+1}}{h_i} (x - x_i)^3 - \frac{1}{6} \frac{z_i}{h_i} (x - x_{i+1})^3 + F_i(x - x_i) + G_i,$$

Using the conditions $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$, we can determine F_i and G_i.

$$y_i = -\frac{1}{6} \frac{z_i}{h_i} (x_i - x_{i+1})^3 + G_i, \quad \Longrightarrow \quad G_i = y_i - \frac{1}{6} z_i h_i^2.$$

and

$$y_{i+1} = \frac{1}{6} \frac{z_{i+1}}{h_i} (x_{i+1} - x_i)^3 + F_i(x_{i+1} - x_i) + G_i \quad \Longrightarrow \quad F_i = \frac{y_{i+1} - y_i}{h_i} + \frac{1}{6} h_i (z_i - z_{i+1}).$$

Hence,

$$\begin{aligned}
S_i(x) &= \frac{1}{6} \frac{z_{i+1}}{h_i} (x - x_i)^3 - \frac{1}{6} \frac{z_i}{h_i} (x - x_{i+1})^3 + \left[\frac{1}{h_i} (y_{i+1} - y_i) + \frac{1}{6} h_i (z_i - z_{i+1}) \right] (x - x_i) \\
&\quad + (y_i - \frac{1}{6} z_i h_i^2), \qquad i = 0, 1, 2, \ldots n-1,
\end{aligned}$$

and

$$S_i'(x) = \frac{1}{2}\frac{z_{i+1}}{h_i}(x - x_i)^2 - \frac{1}{2}\frac{z_i}{h_i}(x - x_{i+1})^2 + \frac{1}{h_i}(y_{i+1} - y_i) + \frac{1}{6}h_i(z_i - z_{i+1}).$$

What left is to determine $z_i = S_i''(x_i)$, for $i = 0, 1, \dots, n-1$. Since $S_{i-1}'(x_i) = S_i'(x_i)$, for $i = 1, 2, \dots, n-1$,

$$\frac{1}{6}h_{i-1}z_{i-1} + \frac{1}{3}h_{i-1}z_i + \frac{1}{h_{i-1}}(y_i - y_{i-1}) = -\frac{1}{3}h_i z_i - \frac{1}{6}h_i z_{i+1} + \frac{1}{h_i}(y_{i+1} - y_i),$$

which leads to

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = \frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}).$$

If we define

$$c_i = \frac{6}{h_i}(y_{i+1} - y_i),$$

then

$$\frac{6}{h_i}(y_{i+1} - y_i) - \frac{6}{h_{i-1}}(y_i - y_{i-1}) = c_i - c_{i-1}.$$

Let $b_i = c_i - c_{i-1}$. Then

$$h_{i-1}z_{i-1} + 2(h_{i-1} + h_i)z_i + h_i z_{i+1} = b_i, \tag{6.41}$$

for $i = 1, 2, \dots, n-1$. There are $n-1$ equations with $n+1$ unknowns $z_0, z_1, \dots, z_n$. Recall, we have 2 degrees of freedom. The widely used choice is to let $z_0 = z_n = 0$, i.e.,

$$S_\Delta''(x_0) = S_\Delta''(x_n) = 0$$

This is called the natural spline. With this convention, we have $n-1$ equations and want to solve $n-1$ unknowns $z_1, z_2, \dots, z_{n-1}$. equation () leads to a linear system of following form.

$$\begin{bmatrix} 2(h_0+h_1) & h_1 & & & & \\ h_1 & 2(h_1+h_2) & h_2 & & & \\ & h_2 & 2(h_2+h_3) & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & h_{n-2} \\ & & & & h_{n-2} & 2(h_{n-2}+h_{n-1}) \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ \vdots \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ \vdots \\ b_{n-1} \end{bmatrix}$$

The coefficient matrix is tridiagonal, symmetric, and diagonally dominant. Hence, Gaussian elimination without pivoting can be used to solve the system for $z_1, \dots, z_n$. Once z_i are computed, $S_i(x)$ can be determined.

Chapter 7

Numerical Differentiation and Integration

7.1 Numerical Differentiation

7.1.1 Finite Difference Method

Suppose a given function f has continuous first derivative and f'' exists. From Taylor's theorem

$$f(x+h) = f(x) + f'(x)h + \frac{1}{2}f''(\xi)h^2, \quad , h > 0,$$

where ξ is between x and $x+h$, one has

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

Hence it is reasonable to use the approximation

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{7.1}$$

difference, and the error involved is

$$|e| = \frac{h}{2}|f''(\xi)| \leq \max_{t\in(x,x+h)} |f''(t)|. \tag{7.2}$$

This kind of error is called the truncation error and is unavoidable when using finite difference approach because the Taylor series is truncated. A only special case is when f is a linear function in which $e = 0$.

Similarly one can derive the backward finite difference approximation

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{7.3}$$

which has the same order of truncation error as the forward finite difference scheme.

In theory, the truncation error $e \to 0$ as $h \to 0$. However, the roundoff error will kick in when h is small. This is because when h is small

$$x \approx x + h \quad \Longrightarrow \quad f(x) \approx f(x+h)$$

since f is continuous. This results cancellation of significance in evaluating $f(x+h) - f(x)$. Higher precision operation should be used to prevent such subtractive cancellation.

The forward difference is an $O(h)$ scheme. An $O(h^2)$ scheme can also be derived from the Taylor's theorem

$$\begin{aligned} f(x+h) &= f(x) + f'(x)h + \frac{1}{2}f''(x)h^2 + \frac{1}{6}f'''(\xi_1)h^3 \\ f(x-h) &= f(x) - f'(x)h + \frac{1}{2}f''(x)h^2 - \frac{1}{6}f'''(\xi_2)h^3, \end{aligned}$$

where ξ_1 is between x and $x+h$ and ξ_2 is between x and $x-h$. Hence

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{1}{6}[f'''(\xi_1) + f'''(\xi_2)]h^3$$

and

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{12}[f'''(\xi_1) + f'''(\xi_2)]h^2$$

Let

$$M = \max_{z \in [x-h, x+h]} f'''(z) \quad \text{and} \quad m = \min_{z \in [x-h, x+h]} f'''(z).$$

If f''' is continuous on $[x-h, x+h]$, then by the intermediate value theorem, there exists $\xi \in [x-h, x+h]$ such that $f(\xi) = \frac{1}{2}[f'''(\xi_1) + f'''(\xi_2)]$. Hence

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{1}{6}f'''(\xi)h^2 = \frac{f(x+h) - f(x-h)}{2h} + O(h^2). \tag{7.4}$$

This is called center difference approximation and the truncation error is

$$|e| = \frac{h^2}{6}f'''(\xi)$$

Similarly, we can derive an $O(h^2)$ scheme from Taylor's theorem for $f''(x)$

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{1}{12}f^{(4)}(\xi)h^2, \tag{7.5}$$

where ξ is between $x - h$ and $x + h$.

7.1.2 Polynomial Interpolation Method

A general approach for numerical differentiation is to use polynomial interpolation. For a given function f (or a set of function values), we find a polynomial p such that $p \approx f$ and expect $p' \approx f'$.

Suppose that $n+1$ points, $x_0, x_1, \ldots, x_n$, and values of f, $f(x_0), f(x_1), \ldots, f(x_n)$, have been given or evaluated, we apply the Lagrange polynomial interpolation scheme to derive

$$p(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x), \tag{7.6}$$

where

$$\ell_i(x) = \prod_{j=0, j\neq i}^{n} \frac{x - x_j}{x_i - x_j}. \tag{7.7}$$

Since $f(x)$ can be written as

$$f(x) = \sum_{i=0}^{n} f(x_i)\ell_i(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) w(x), \tag{7.8}$$

where

$$w(x) = \prod_{j=0}^{n} (x - x_j), \tag{7.9}$$

we have, by taking derivative,

$$f'(x) = \sum_{i=0} n f(x_i)\ell_i'(x) + \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) w'(x) + \frac{1}{(n+1)!} w(x) \frac{d}{dx} f^{(n+1)}(\xi_x). \tag{7.10}$$

Note that

$$w'(x) = \sum_{j=0}^{n} \prod_{i=0, i\neq j}^{n} (x - x_i). \tag{7.11}$$

Hence a reasonable approximation for the first derivative of f is

$$f'(x) \approx \sum_{i=0}^{n} f(x_i)\ell_i'(x). \tag{7.12}$$

When $x = x_k$ for some $0 \leq k \leq n$,

$$w(x_k) = 0 \quad \text{and} \quad w'(x_k) = \prod_{i=0, i\neq k}^{n} (x_k - x_i).$$

Hence

$$f'(x_k) = \sum_{i=0}^{n} f(x_i)\ell_i'(x_k) + \frac{1}{(n+1)!} f^{(n+1)}(\xi_x) \prod_{i=0, i\neq k}^{n} (x_k - x_i). \tag{7.13}$$

7.1.3 Richardson Extrapolation Method

Again from the Taylor's theorem

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + \frac{h^5}{5!}f^{(5)}(x) + \frac{h^6}{6!}f^{(6)}(x) + \cdots \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - \frac{h^5}{5!}f^{(5)}(x) + \frac{h^6}{6!}f^{(6)}(x) + \cdots \end{aligned}$$

we have

$$f(x+h) - f(x-h) = 2hf'(x) + \frac{2h^3}{3!}f'''(x) + \frac{2h^5}{5!}f^{(5)}(x) + \cdots ,$$

and, consequently,

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \left[\frac{h^2}{3!}f'''(x) + \frac{h^4}{5!}f^{(5)}(x) + \cdots\right]. \tag{7.14}$$

Denote

$$L \equiv f'(x) \quad \text{and} \quad \phi(h) = \frac{f(x+h) - f(x-h)}{2h}. \tag{7.15}$$

Then equation () can be expressed as

$$L = \phi(h) + a_2h^2 + a_4h^4 + a_6h^6 + \cdots . \tag{7.16}$$

This yields the $O(h^2)$ central difference scheme.

Now replace h with $\frac{h}{2}$ in the Taylor's expansion, one has instead

$$f'(x) = \frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{h} - \left[\frac{1}{3!}\left(\frac{h}{2}\right)^2 f'''(x) + \frac{1}{5!}\left(\frac{h}{2}\right)^4 f^{(5)}(x) + \cdots\right]. \tag{7.17}$$

This gives

$$L = \phi(\frac{h}{2}) + \frac{1}{4}a_2h^2 + \frac{1}{16}a_4h^4 + \frac{1}{64}a_6h^6 + \cdots , \tag{7.18}$$

or, equivalently,

$$4L = 4\phi(\frac{h}{2}) + a_2h^2 + \frac{1}{4}a_4h^4 + \frac{1}{16}a_6h^6 + \cdots . \tag{7.19}$$

Subtract () from ()

$$3L = 4\phi(\frac{h}{2}) - \phi(h) - \frac{3}{4}a_4h^4 - \frac{15}{16}a_6h^6 + \cdots .$$

This yields an $O(h^4)$ approximation scheme

$$f'(x) = L = \frac{4}{3}\phi(\frac{h}{2}) - \frac{1}{3}\phi(h) - \frac{1}{4}a_4h^4 - \frac{5}{16}a_6h^6 + \cdots . \tag{7.20}$$

We may repeat this idea by letting

$$\psi(h) = \frac{4}{3}\phi(\frac{h}{2}) - \frac{1}{3}\phi(h). \tag{7.21}$$

Then

$$L = \psi(h) + b_4 h^4 + b^6 h^6 + \cdots \tag{7.22}$$

and also

$$16L = 16\psi(\frac{h}{2}) + b_4 h^4 + \frac{1}{4} b_6 h^6 + \cdots . \tag{7.23}$$

Subtract () from () to give

$$15L = 16\psi(\frac{h}{2}) - \psi(h) - \frac{3}{4} b_6 h^6 + \cdots .$$

It leads to an $O(h^6)$ approximation

$$f'(x) = L = \frac{16}{15}\psi(\frac{h}{2}) - \frac{1}{15}\psi(h) - \frac{1}{20} b_6 h^6 + \cdots . \tag{7.24}$$

The process can be repeated again and again, and is called the Richardson extrapolation. For practical implementation, we denote

7.2 Numerical Integration

Calculating the definite integral of a given function $f(x)$ over an interval $[a, b]$,

$$\int_a^b f(x)\, dx, \tag{7.25}$$

is a classic problem. By the fundamental theorem of Calculus, this problem is solved by finding an anti-derivative F of f, that is, $F'(x) = f(x)$, and then

$$\int_a^b f(x)\, dx = F(b) - F(a).$$

But finding an anti-derivative is not an easy task in general. Many elementary functions do not have simple anti-derivatives. Hence it is certainly not a good approach for numerical computation.

Numerical integration is the process of producing a numerical value for the definite integral problem (7.25). As a rule, definite integrals are computed using discretization methods which approximate the integral by finite sums corresponding to some partition of the interval of integration $[a, b]$. One powerful stratagem for computing the integral (7.25) numerically is to replace f by another function g that approximates f well and is easy to integrate. Hopefully, when $f \approx g$, we can expect

$$\int_a^b f(x)\,dx \approx \int_a^b g(x)\,dx.$$

It should be no surprise that polynomials are good candidates for the function g, and indeed g can be a polynomial that interpolates f at a certain set of nodes. With this observation, we would expect that the interpolation techniques play an important role in numerical integration.

7.2.1 Elements of Numerical Integration

The basic method involved in approximating the integration (7.25) is called numerical quadrature and uses a sum of the type

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} c_i f(x_i). \tag{7.26}$$

The method of quadrature in this section is based on the polynomial interpolation. We first select a set of distinct nodes $\{x_0, x_1, \dots, x_n\}$ from the interval $[a, b]$. Then the Lagrange polynomial

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

is used to approximate $f(x)$. With the error term we have

$$f(x) = P_n(x) + E_n(x) = \sum_{i=0}^{n} f(x_i)L_i(x) + \frac{f^{(n+1)}(\zeta_x)}{(n+1)!}\prod_{i=0}^{n}(x - x_i),$$

where $\zeta_x \in [a, b]$ and depends on x, and

$$\begin{aligned}
\int_a^b f(x)\,dx &= \int_a^b P_n(x)\,dx + \int_a^b E_n(x)\,dx \\
&= \sum_{i=0}^{n} f(x_i)\int_a^b L_i(x)\,dx + \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\zeta_x)\prod_{i=0}^{n}(x - x_i)\,dx. \qquad (7.27)
\end{aligned}$$

The quadrature formula is, therefore,

$$\int_a^b f(x)\,dx \approx \int_a^b P_n(x)\,dx = \sum_{i=0}^{n} f(x_i)\int_a^b L_i(x)\,dx \equiv \sum_{i=0}^{n} c_i f(x_i), \tag{7.28}$$

where

$$c_i = \int_a^b L_i(x)\,dx = \int_a^b \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}\,dx. \tag{7.29}$$

Moreover, the error in the quadrature formula is given by

$$E = \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\zeta_x)\prod_{i=0}^{n}(x-x_i)\,dx, \tag{7.30}$$

for some $\zeta_x \in [a,b]$. If $|f^{(n+1)}(x)| \leq M$ on $[a,b]$, then

$$\left|\int_a^b f(x)\,dx - \sum_{i=0}^{n} c_i f(x_i)\right| \leq \frac{M}{(n+1)!}\int_a^b \prod_{i=0}^{n}(x-x_i)\,dx. \tag{7.31}$$

The choice of nodes that makes the right-hand side of this error bound as small as possible is know to be

$$x_i = \frac{a+b}{2} + \frac{b-a}{2}\cos\left[\frac{(i+1)\pi}{n+2}\right], \qquad i = 0,1,\dots,n. \tag{7.32}$$

Of course, a polynomial interpolation to f can be obtained in other ways, for example, polynomial in Newton's form using divided-difference method,

$$P_n(x) = f(x_0) + \sum_{i=1}^{n} f[x_0,x_1,\dots,x_i]\prod_{j=0}^{i-1}(x-x_j)$$

where $f[x_0,x_1,\dots,x_i]$ are evaluated with the divided difference algorithm. Then

$$\int_a^b f(x)\,dx \approx f(x_0)(b-a) + \sum_{i=1}^{n} f[x_0,x_1,\dots,x_i]\int_a^b \prod_{j=0}^{i-1}(x-x_j)\,dx. \tag{7.33}$$

The standard derivation of quadrature error formulas is based on determining the class of polynomials for which theses formulas produce exact results. The next definition is used to facilitate the discussion of this derivation.

Definition 7.1 *The degree of accuracy, or precision, of a quadrature formula is the largest positive integer n such that the formula is exact for x^k, when $k = 0,1,\dots,n$.*

The definition implies that the degree of accuracy of a quadrature formula is n if and only if the error $E = 0$ for all polynomials $P(x)$ of degree less than or equal to n, but $E \neq 0$ for some polynomials of degree greater than n.

7.2.2 Newton-Cotes Formulas

A quadrature formula of the form (7.26) is called a Newton-Cotes formula if the nodes $\{x_0,x_1,\dots,x_n\}$ are equally spaced. Consider a uniform partition of the closed interval $[a,b]$ by

$$x_i = a + ih, \quad i = 0,1,\dots,n, \quad h = \frac{b-a}{n},$$

where n is a positive integer and h is called the step length. By introduction a new variable t such that $x = a + ht$, the fundamental Lagrange polynomial becomes

$$L_i(x) = \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{a + ht - a - jh}{a + ih - a - jh} = \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{t - j}{i - j} \equiv \varphi_i(t).$$

Therefore, the integration (7.29) gives

$$c_i = \int_a^b L_i(x)\,dx = \int_0^n \varphi_i(t)h\,dt = h\int_0^n \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{t - j}{i - j}\,dt, \tag{7.34}$$

and the general Newton-Cotes formula has the form

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n} f(x_i)\int_0^n \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{t - j}{i - j}\,dt + \frac{1}{(n+1)!}\int_a^b f^{(n+1)}(\zeta_x)\prod_{i=0}^{n}(x - x_i)\,dx. \tag{7.35}$$

The simplest case is to choose $n = 1$, $x_0 = a$, $x_1 = b$, $h = b - a$, and use the linear Lagrange polynomial

$$P_1(x) = f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0} = f(a)\frac{x - b}{a - b} + f(b)\frac{x - a}{b - a}.$$

to interpolate $f(x)$. Then

$$c_0 = h\int_0^1 \frac{t - 1}{0 - 1}\,dt = \frac{h}{2}, \qquad c_1 = h\int_0^1 \frac{t - 0}{1 - 0}\,dt = \frac{h}{2},$$

and

$$\int_a^b P_1(x)\,dx = c_0 f(x_0) + c_1 f(x_1) = \frac{h}{2}\left[f(a) + f(b)\right].$$

Since $(x - x_0)(x - x_1) = (x - a)(x - b)$ does not change sign on $[a, b]$, by the Weighted Mean-Value Theorem for integrals, there exists some $\xi \in (a, b)$ such that

$$\begin{aligned}
\int_a^b f''(\zeta_x)(x - x_0)(x - x_1)\,dx &= f''(\xi)\int_a^b (x - x_0)(x - x_1)\,dx \\
&= f''(\xi)\int_a^b (x - a)(x - b)\,dx \\
&= f''(\xi)\left[\frac{1}{3}x^3 - \frac{1}{2}(a + b)x^2 + abx\right]\Bigg|_a^b \\
&= -\frac{1}{6}f''(\xi)(b - a)^3 = -\frac{1}{6}f''(\xi)h^3.
\end{aligned}$$

Consequently,

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + f(b)\right] - \frac{h^3}{12}f''(\xi).$$

This gives the so-called Trapezoidal rule.

Trapezoidal Rule:

$$\int_a^b f(x)\,dx = \frac{1}{2}(b-a)\,[f(a)+f(b)] - \frac{h^3}{12}f''(\xi), \tag{7.36}$$

where $h = b-a$ and $\xi \in (a,b)$.

It is evident that the error term of the Trapezoidal rule is $O(h^3)$. Since the rule involves f'', it gives the exact result when applied to any function whose second derivative is identically zero, e.g., any polynomial of degree 1 or less. Hence the degree of accuracy of Trapezoidal rule is one.

If we choose $n=2$, $x_0=a$, $x_1=\frac{1}{2}(a+b)$, $x_2=b$, $h=(b-a)/2$, and the second order Lagrange polynomial

$$P_2(x) = f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

to interpolate $f(x)$, then

$$\begin{aligned}
c_0 &= h\int_0^2 \frac{t-1}{0-1}\cdot\frac{t-2}{0-2}\,dt = \frac{h}{2}\int_0^2 (t^2-3t+2)\,dt = \frac{h}{3},\\
c_1 &= h\int_0^2 \frac{t-0}{1-0}\cdot\frac{t-2}{1-2}\,dt = -h\int_0^2 (t^2-2t)\,dt = \frac{4h}{3},\\
c_2 &= h\int_0^2 \frac{t-0}{2-0}\cdot\frac{t-1}{2-1}\,dt = \frac{h}{2}\int_0^2 (t^2-t)\,dt = \frac{h}{3},
\end{aligned}$$

and

$$\int_a^b P_2(x)\,dx = c_0 f(x_0) + c_1 f(x_1) + c_2 f(x_2) = h\left[\frac{1}{3}f(a) + \frac{4}{3}f(\frac{a+b}{2}) + \frac{1}{3}f(b)\right]$$

gives the so-called Simpson's rule. Deriving the formulation this way, however, the error term

$$\frac{1}{6}\int_a^b f^{(3)}(\zeta_x)(x-x_0)(x-x_1)(x-x_2)\,dx$$

provides only an $O(h^4)$ formulation involving $f^{(3)}$. A higher order error analysis can be derived by expanding f in the third Taylor's formula about x_1. Then for each $x \in [a,b]$, there exists $\zeta_x \in (a,b)$ such that

$$f(x) = f(x_1) + f'(x_1)(x-x_1) + \frac{f''(x_1)}{2}(x-x_1)^2 + \frac{f'''(x_1)}{6}(x-x_1)^3 + \frac{f^{(4)}(\zeta_x)}{24}(x-x_1)^4.$$

Then

$$\begin{aligned}
\int_a^b f(x)\,dx = {} & \left[f(x_1)(x-x_1) + \frac{f'(x_1)}{2}(x-x_1)^2 + \frac{f''(x_1)}{6}(x-x_1)^3 + \frac{f'''(x_1)}{24}(x-x_1)^4\right]\Bigg|_a^b \\
& + \frac{1}{24}\int_a^b f^{(4)}(\zeta_x)(x-x_1)^4\,dx.
\end{aligned}$$

Note that $(b-x_1)=h$, $(a-x_1)=-h$, and since $(x-x_1)^4$ does not change sign in $[a,b]$, by the Weighted Mean-Value Theorem for Integral, there exists $\xi_1 \in (a,b)$ such that

$$\int_a^b f^{(4)}(\zeta_x)(x-x_1)^4\,dx = f^{(4)}(\xi_1)\int_a^b (x-x_1)^4\,dx = \frac{2f^{(4)}(\xi_1)}{5}h^5.$$

Consequently,

$$\int_a^b f(x)\,dx = 2f(x_1)h + \frac{f''(x_1)}{3}h^3 + \frac{f^{(4)}(\xi_1)}{60}h^5.$$

Finally we replace $f''(x_1)$ by the central finite difference formulation

$$f''(x_1) = \frac{f(x_0)-2f(x_1)+f(x_2)}{h^2} - \frac{f^{(4)}(\xi_2)}{12}h^2,$$

for some $\xi_2 \in (a,b)$, to obtain

$$\begin{aligned}\int_a^b f(x)\,dx &= 2hf(x_1) + \frac{h}{3}\left(f(x_0)-2f(x_1)+f(x_2)\right) - \frac{f^{(4)}(\xi_2)}{36}h^5 + \frac{f^{(4)}(\xi_1)}{60}h^5 \\ &= h\left[\frac{1}{3}f(x_0)+\frac{4}{3}f(x_1)+\frac{1}{3}f(x_2)\right] + \frac{1}{90}\left[\frac{3}{2}f^{(4)}(\xi_1) - \frac{5}{2}f^{(4)}(\xi_2)\right]h^5.\end{aligned}$$

By letting $f(x)=x^4$, one can show that there exists $\xi \in (a,b)$ such that

$$\int_a^b f(x)\,dx = h\left[\frac{1}{3}f(x_0)+\frac{4}{3}f(x_1)+\frac{1}{3}f(x_2)\right] + \frac{f^{(4)}(\xi)}{90}h^5.$$

This gives the Simpson's rule formulation.

Simpson's Rule:

$$\int_a^b f(x)\,dx = \left(\frac{b-a}{2}\right)\left[\frac{1}{3}f(a)+\frac{4}{3}f(\frac{a+b}{2})+\frac{1}{3}f(b)\right] + \frac{f^{(4)}(\xi)}{90}h^5, \tag{7.37}$$

for some $\xi \in (a,b)$. The Simpson's rule is an $O(h^5)$ scheme and the degree of accuracy is three.

The Trapezoidal and Simpson's rules are examples of a class of methods known as closed Newton-Cotes formula. The $(n+1)$-point closed Newton-Cotes method uses nodes $x_i = a+ih$, for $i=0,1,\ldots,n$, where $h=(b-a)/n$. Note that both endpoints, $a=x_0$ and $b=x_n$, of the closed interval $[a,b]$ are included as nodes. The following theorem details the Newton-Cotes formulas and the associated error analysis.

Theorem 7.1 (Closed Newton-Cotes Formulas) *For a given function $f(x)$ and closed interval $[a,b]$, the $(n+1)$-point closed Newton-Cotes method uses nodes*

$$x_i = a+ih, \quad i=0,1,\ldots,n, \quad h=\frac{b-a}{n}.$$

If n is even and $f \in C^{n+2}[a,b]$, then

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n} \alpha_i f(x_i) + \frac{h^{n+3} f^{(n+2)}(\xi)}{(n+2)!}\int_0^n t^2(t-1)\cdots(t-n)\,dt, \tag{7.38}$$

and if n is odd and $f \in C^{n+1}[a,b]$, then

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n} \alpha_i f(x_i) + \frac{h^{n+2} f^{(n+1)}(\xi)}{(n+1)!}\int_0^n t(t-1)\cdots(t-n)\,dt, \tag{7.39}$$

where $\xi \in (a,b)$ and

$$\alpha_i = \int_0^n \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-j}{i-j}\,dt, \qquad i = 0,1,\ldots,n. \tag{7.40}$$

Consequently, the degree of accuracy is $n+1$ when n is an even integer, and n when n is an odd integer.

The weights α_i in the Newton-Cotes formula has the property

$$\sum_{i=0}^{n} \alpha_i = n. \tag{7.41}$$

This can be shown by applying the formula to $f(x) = 1$ with interpolating polynomial $P_n(x) = 1$. Let s be the common denominator of α_i, that is,

$$\alpha_i = \frac{\sigma_i}{s} \qquad (\Rightarrow \sigma_i = s\alpha_i)$$

such that σ_i are integers, then the formulation for approximating the definite integral can be expressed as

$$\int_a^b f(x)\,dx \approx h\sum_{i=0}^{n} \alpha_i f(x_i) = \frac{h}{s}\sum_{i=0}^{n} \sigma_i f(x_i). \tag{7.42}$$

Some of the most common closed Newton-Cotes formulas with their error terms are listed in the following table.

Name	n	s	σ_i	Error
Trapezoidal rule	1	2	$1, 1$	$-\frac{1}{12} f^{(2)}(\xi)h^3$
Simpson's rule	2	3	$1, 4, 1$	$-\frac{1}{90} f^{(4)}(\xi)h^5$
3/8-rule	3	$\frac{8}{3}$	$1, 3, 3, 1$	$-\frac{3}{80} f^{(4)}(\xi)h^5$
Milne's rule	4	$\frac{45}{2}$	$7, 32, 12, 32, 7$	$-\frac{8}{945} f^{(6)}(\xi)h^7$
	5	$\frac{288}{5}$	$19, 75, 50, 50, 75, 19$	$-\frac{275}{12096} f^{(6)}(\xi)h^7$
Weddle's rule	6	140	$41, 216, 27, 272, 27, 216, 41$	$-\frac{9}{1400} f^{(8)}(\xi)h^9$

Another class of Newton-Cotes formulas is the open Newton-Cotes formulas in which the nodes

$$x_i = x_0 + ih, \quad i = 0, 1, \ldots, n, \quad \text{where } x_0 = a + h \text{ and } h = \frac{b-a}{n+2},$$

are used. This implies that $x_n = b - h$, and the endpoints, a and b, are not used. Hence we label $a = x_{-1}$ and $b = x_{n+1}$. The following theorem summarizes the open Newton-Cotes formulas.

Theorem 7.2 (Open Newton-Cotes Formulas) *For a given function $f(x)$ and closed interval $[a, b]$, the $(n+1)$-point open Newton-Cotes method uses nodes*

$$x_i = x_0 + ih, \quad i = 0, 1, \ldots, n, \quad \textit{where } x_0 = a + h \textit{ and } h = \frac{b-a}{n+2}.$$

If n is even and $f \in C^{n+2}[a, b]$, then

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n} \alpha_i f(x_i) + \frac{h^{n+3} f^{(n+2)}(\xi)}{(n+2)!} \int_{-1}^{n+1} t^2(t-1)\cdots(t-n)\,dt, \tag{7.43}$$

and if n is odd and $f \in C^{n+1}[a, b]$, then

$$\int_a^b f(x)\,dx = h\sum_{i=0}^{n} \alpha_i f(x_i) + \frac{h^{n+2} f^{(n+1)}(\xi)}{(n+1)!} \int_{-1}^{n+1} t(t-1)\cdots(t-n)\,dt, \tag{7.44}$$

where $\xi \in (a, b)$ and

$$\alpha_i = \int_{-1}^{n+1} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t-j}{i-j}\,dt, \qquad i = 0, 1, \ldots, n. \tag{7.45}$$

Consequently, the degree of accuracy is $n+1$ when n is an even integer, and n when n is an odd integer.

The simplest open Newton-Cotes formula is choosing $n = 0$ and only using the midpoint $x_0 = \frac{a+b}{2}$. Then the coefficient and the error term can be computed easily as

$$\alpha_0 = \int_1^{-1} dt = 2, \quad \text{and} \quad \frac{h^3 f''(\xi)}{2!} \int_{-1}^{1} t^2\,dt = \frac{1}{3} f''(\xi) h^3.$$

These gives the so-called Midpoint rule or Rectangular rule.

Midpoint Rule:

$$\int_a^b f(x)\,dx = 2hf(x_0) + \frac{1}{3}f''(\xi)h^3 = (b-a)f(\frac{a+b}{2}) + \frac{1}{3}f''(\xi)h^3, \tag{7.46}$$

for some $\xi \in (a,b)$.

Analogous to the closed Newton-Cotes formulas, we list some of the commonly used open Newton-Cotes formulas in the following table.

Name	n	s	σ_i	Error
Midpoint rule	0	1	2	$\frac{1}{3}f^{(2)}(\xi)h^3$
	1	2	$3,3$	$\frac{3}{4}f^{(2)}(\xi)h^3$
	2	3	$8,-4,8$	$\frac{14}{45}f^{(4)}(\xi)h^5$
	3	24	$55,5,5,55$	$\frac{95}{144}f^{(4)}(\xi)h^5$

7.2.3 Composite Newton-Cotes Forumlas

It is obvious that the Newton-Cotes formulas are generally not suitable for numerical integration over large interval. Higher degree formulas would be required, and the coefficients in these formulas are difficult to obtain. Also the Newton-Cotes formulas which are based on polynomial interpolation would be inaccurate over a large interval because of the oscillatory nature of high-degree polynomials. Now we discuss a piecewise approach, called composite rule, to numerical integration over large interval that uses the low-order Newton-Cotes formulas.

A composite rule is one obtained by applying an integration formula for a single interval to each subinterval of a partitioned interval. To illustrate the procedure, we choose an even integer n and partition the interval $[a,b]$ into n subintervals by nodes $x_0 < x_1 < \cdots < x_n = b$, and apply Simpson's rule on each consecutive pair of subintervals. With

$$h = \frac{b-a}{n} \quad \text{and} \quad x_j = a + jh, \quad j = 0, 1, \ldots, n,$$

we have on each interval $[x_{2j-2}, x_{2j}]$,

$$\int_{x_{2j-2}}^{x_{2j}} f(x)\,dx = \frac{h}{3}\left[f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j})\right] - \frac{h^5}{90}f^{(4)}(\xi_j),$$

for some $\xi_j \in (x_{2j-2}, x_{2j})$, provided that $f \in C^4[a,b]$. The composite rule is obtained by summing up over the entire interval, that is,

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{j=1}^{n/2} \int_{x_{2j-2}}^{x_{2j}} f(x)\,dx \\
&= \sum_{j=1}^{n/2} \left[\frac{h}{3} \left(f(x_{2j-2}) + 4f(x_{2j-1}) + f(x_{2j}) \right) - \frac{h^5}{90} f^{(4)}(\xi_j) \right] \\
&= \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2) + f(x_2) + 4f(x_3) + f(x_4) + f(x_4) + 4f(x_5) \\
&\quad + \cdots + f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] - \frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \\
&= \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + 4f(x_5) \\
&\quad + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)] - \frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \\
&= \frac{h}{3} \left[f(x_0) + 4 \sum_{j=1}^{n/2} f(x_{2j-1}) + 2 \sum_{j=1}^{(n/2)-1} f(x_{2j}) + f(x_n) \right] - \frac{h^5}{90} \sum_{j=1}^{n/2} f^{(4)}(\xi_j).
\end{aligned}$$

To estimate the error associated with approximation, since $f \in C^4[a,b]$, we have, by the Extreme Value Theorem,

$$\min_{x\in[a,b]} f^{(4)}(x) \le f^{(4)}(\xi_j) \le \max_{x\in[a,b]} f^{(4)}(x),$$

for each $\xi_j \in (x_{2j-2}, x_{2j})$. Hence

$$\frac{n}{2} \min_{x\in[a,b]} f^{(4)}(x) \le \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le \frac{n}{2} \max_{x\in[a,b]} f^{(4)}(x),$$

and

$$\min_{x\in[a,b]} f^{(4)}(x) \le \frac{2}{n} \sum_{j=1}^{n/2} f^{(4)}(\xi_j) \le \max_{x\in[a,b]} f^{(4)}(x).$$

By the Intermediate Value Theorem, there exists $\mu \in (a,b)$ such that

$$f^{(4)}(\mu) = \frac{2}{n} \sum_{j=1}^{n/2} f^{(4)}(\xi_j).$$

Thus, by replacing $n = (b-a)/h$,

$$\sum_{j=1}^{n/2} f^{(4)}(\xi_j) = \frac{n}{2} f^{(4)}(\mu) = \frac{b-a}{2h} f^{(4)}(\mu).$$

Consequently, the composite Simpson's rule is derived.

Composite Simpson's Rule:

$$\int_a^b f(x)\,dx = \frac{h}{3}\left[f(a) + 4\sum_{j=1}^{n/2} f(x_{2j-1}) + 2\sum_{j=1}^{(n/2)-1} f(x_{2j}) + f(b)\right] - \frac{b-a}{180}f^{(4)}(\mu)h^4, \quad (7.47)$$

where n is an even integer, $h = (b-a)/n$, $x_j = a+jh$, for $j = 0, 1, \ldots, n$, and some $\mu \in (a, b)$.

The composite Midpoint rule can be derived in a similar way, except the midpoint rule is applied on each subinterval $[x_{2j-1}, x_{2j}]$ instead. That is,

$$\int_{x_{2j-2}}^{x_{2j}} f(x)\,dx = 2hf(x_{2j-1}) + \frac{h^3}{3}f''(\xi_j), \qquad j = 1, 2, \ldots, \frac{n}{2}.$$

Note that n must again be even. Consequently,

$$\int_a^b f(x)\,dx = 2h\sum_{j=1}^{n/2} f(x_{2j-1}) + \frac{h^3}{3}\sum_{j=1}^{n/2} f''(\xi_j).$$

The error term can be written as

$$\sum_{j=1}^{n/2} f''(\xi_j) = \frac{n}{2}f''(\mu) = \frac{b-a}{2h}f''(\mu),$$

for some $\mu \in (a, b)$. Therefore, the composite Midpoint rule has the following formulation.

Composite Midpoint Rule:

$$\int_a^b f(x)\,dx = 2h\sum_{j=1}^{n/2} f(x_{2j-1}) - \frac{b-a}{6}f''(\mu)h^2, \quad (7.48)$$

where n is an even integer, $h = (b-a)/n$, $x_j = a+jh$, for $j = 0, 1, \ldots, n$, and some $\mu \in (a, b)$.

To derive the composite Trapezoidal rule, we partition the interval $[a, b]$ by n equally spaced nodes $a = x_0 < x_1 < \cdots < x_n = b$, where n can be either odd or even. We then apply the trapezoidal rule on each subinterval $[x_{j-1}, x_j]$ and sum them up to obtain

$$\begin{aligned}
\int_a^b f(x)\,dx &= \sum_{j=1}^{n} \int_{x_{j-1}}^{x_j} f(x)\,dx \\
&= \sum_{j=1}^{n} \left[\frac{h}{2}\left(f(x_{j-1}) + f(x_j)\right) - \frac{h^3}{12} f''(\xi_j)\right] \\
&= \frac{h}{2}\left[f(x_0) + f(x_1) + f(x_1) + f(x_2) + \cdots + f(x_{n-1}) + f(x_n)\right] - \frac{h^3}{12}\sum_{j=1}^{n} f''(\xi_j) \\
&= \frac{h}{2}\left[f(x_0) + 2f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-1}) + f(x_n)\right] - \frac{h^3}{12}\sum_{j=1}^{n} f''(\xi_j) \\
&= \frac{h}{2}\left[f(a) + \sum_{j=1}^{n-1} f(x_j) + f(b)\right] - \frac{h^3}{12}\sum_{j=1}^{n} f''(\xi_j) \\
&= \frac{h}{2}\left[f(a) + \sum_{j=1}^{n-1} f(x_j) + f(b)\right] - \frac{b-a}{12} f''(\mu) h^2,
\end{aligned}$$

where each $\xi_j \in (x_{j-1}, x_j)$ and $\mu \in (a, b)$.

Composite Trapezoidal Rule

$$\int_a^b f(x)\,dx = \frac{h}{2}\left[f(a) + \sum_{j=1}^{n-1} f(x_j) + f(b)\right] - \frac{b-a}{12} f''(\mu) h^2, \tag{7.49}$$

where n is an integer, $h = (b-a)/n$, $x_j = a + jh$, for $j = 0, 1, \ldots, n$, and some $\mu \in (a, b)$.

7.3 Gaussian Quadrature

In the preceding section, the quadrature formulas of the type

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} c_i f(x_i) \tag{7.50}$$

are based on polynomial interpolation, where the choice of nodes $x_0, x_1, \ldots, x_n$ was made *a priori*. All Newton-Cotes formulas use values of the function at equally spaced points. Once the nodes were fixed, the coefficients were determined, e.g., by integrating the fundamental Lagrange polynomials of degree n. Furthermore, these formulas are exact for polynomials of degree $\leq n$.

This approach is convenient when the formulas are combined to form the composite rules, but the restriction may decrease the accuracy of the approximation. It is nature to ask whether some choices of nodes are better than others in formula (7.50). In this section, we consider the Gaussian quadrature which chooses the points for evaluation in an optimal, rather than pre-fixed or equally-spaced, way. The theory can be formulated for quadrature rule of a slightly more general form, namely,

$$\int_a^b f(x)w(x)\,dx \approx \sum_{i=0}^{n} c_i f(x_i), \tag{7.51}$$

where $w(x)$ is a fixed positive weight function. The case when $w(x) \equiv 1$ is, naturally, of special importance. The nodes $x_0, x_1, \dots, x_n$ in the interval $[a, b]$ and the coefficients $c_0, c_1, \dots, c_n$ are chosen to minimize the expected error obtained in performing the approximation (7.51), and to produce the exact result for the largest class of polynomials, that is, the choice which gives the greatest degree of precision.

The coefficients $c_0, c_1, \dots, c_n$ in the approximation formula (7.51) are arbitrary, and the nodes $x_0, x_1, \dots, x_n$ are restricted only by the specification that they lie in $[a, b]$, the interval of integration. These give $2n+2$ degrees of freedom. Thus we can expect that the quadrature formula of the form (7.51) can be discovered that will be exact for polynomials of degree $\leq 2n+1$.

7.3.1 Orthogonal Polynomials and Quadrature Rule

Recall that a real inner-product space E is a linear vector space in which an inner product $< \cdot, \cdot >$ and a norm $\|\cdot\|$ have been defined with the following properties:

1. $\langle f, g\rangle = \langle g, f\rangle$
2. $\langle f, \alpha g + \beta h\rangle = \alpha\langle f, g\rangle + \beta\langle f, h\rangle$
3. $\langle f, f\rangle > 0$ if $f \neq 0$, and $\langle f, f\rangle = 0$ if and only if $f = 0$
4. $\|f\| = \sqrt{\langle f, f\rangle}$

where $f, g, h \in E$ and $\alpha, \beta \in \mathbb{R}$.

In a inner-product space, we say f is orthogonal to g, and write $f \perp g$ if $\langle f, g\rangle = 0$. We write $f \perp G$ if $f \perp g$ for all $g \in G$. We say that a finite or infinite sequence of vectors $f_1, f_2, \dots$ in an inner-product space is orthogonal if $\langle f_i, f_j\rangle = 0$ for all $i \neq j$, and orthonormal if $\langle f_i, f_j\rangle = \delta_{ij}$.

Lemma 7.1 *In an inner-product space, we have*

1. $\langle \sum_{i=1}^n a_i f_i, g\rangle = \sum_{i=1}^n a_i \langle f_i, g\rangle$
2. $\|f+g\|^2 = \|f\|^2 + 2\langle f, g\rangle + \|g\|^2$
3. *If* $f \perp g$, *then* $\|f+g\|^2 = \|f\|^2 + \|g\|^2$
4. $|\langle f, g\rangle| \leq \|f\|\|g\|$
5. $\|f+g\|^2 + \|f-g\|^2 = 2\|f\|^2 + 2\|g\|^2$

The space of continuous functions on $[a, b]$, $C[a, b]$, with inner-product defined as

$$\langle f, g\rangle = \int_a^b f(x)g(x)w(x)\,dx, \tag{7.52}$$

where $w(x)$ is a fixed continuous positive function called weight function, is an inner-product space.

Definition 7.2 *$\{\phi_0, \phi_1, \ldots, \phi_n\}$, where $\phi_i \in C[a, b]$ for all $i = 0, 1, \ldots, n$, is said to be an orthogonal set of functions for the interval $[a, b]$ with respect to the weight function w if*

$$\langle \phi_i, \phi_j\rangle = \int_a^b \phi_i(x)\phi_j(x)w(x)\,dx = \begin{cases} 0, & \text{when } i \neq j, \\ \alpha_i > 0, & \text{when } i = j. \end{cases}$$

If, in addition, $\alpha_i = 1$ for all $i = 0, 1, \ldots, n$, then the set is said to be orthonormal.

Theorem 7.3 *The set of polynomials $\{p_0, p_1, \ldots, p_n\}$ defined inductively as follows is orthogonal:*

$$p_n(x) = (x - a_n)p_{n-1}(x) - b_n p_{n-2}(x) \qquad (n \geq 2)$$

with $p_0(x) = 1$, $p_1(x) = x - a_1$, and

$$a_n = \frac{\langle xp_{n-1}, p_{n-1}\rangle}{\langle p_{n-1}, p_{n-1}\rangle}, \qquad b_n = \frac{\langle xp_{n-1}, p_{n-2}\rangle}{\langle p_{n-2}, p_{n-2}\rangle}.$$

Proof: It is clear from the formulation that each p_n is a monic polynomial of degree n and is therefore not zero. Hence, the denominators in a_n and b_n are not zero. Now we prove the theorem by induction on n that $\langle p_n, p_i\rangle = 0$ for $i = 0, 1, \ldots, n-1$.

For $n = 1$, it can be shown directly from the definition that

$$\langle p_1, p_0\rangle = \langle (x - a_1)p_0, p_0\rangle = \langle xp_0, p_0\rangle - a_1\langle p_0, p_0\rangle = \langle xp_0, p_0\rangle - \frac{\langle xp_0, p_0\rangle}{\langle p_0, p_0\rangle}\langle p_0, p_0\rangle = 0.$$

Now suppose that the assertion is true for an index $n - 1$, where $n \geq 2$. That is

$$\langle p_{n-1}, p_i\rangle = 0, \qquad i = 0, 1, \ldots, n-2, \quad n \geq 2.$$

Now we can verify that

$$\begin{aligned}
\langle p_n, p_{n-1}\rangle &= \langle (x - a_n)p_{n-1} - b_n p_{n-2}, p_{n-1}\rangle \\
&= \langle xp_{n-1}, p_{n-1}\rangle - a_n\langle p_{n-1}, p_{n-1}\rangle - b_n\langle p_{n-2}, p_{n-1}\rangle \\
&= \langle xp_{n-1}, p_{n-1}\rangle - \frac{\langle xp_{n-1}, p_{n-1}\rangle}{\langle p_{n-1}, p_{n-1}\rangle}\langle p_{n-1}, p_{n-1}\rangle \\
&= 0, \\
\langle p_n, p_{n-2}\rangle &= \langle (x - a_n)p_{n-1} - b_n p_{n-2}, p_{n-2}\rangle \\
&= \langle xp_{n-1}, p_{n-2}\rangle - a_n\langle p_{n-1}, p_{n-2}\rangle - b_n\langle p_{n-2}, p_{n-2}\rangle \\
&= \langle xp_{n-1}, p_{n-2}\rangle - \frac{\langle xp_{n-1}, p_{n-2}\rangle}{\langle p_{n-2}, p_{n-2}\rangle}\langle p_{n-2}, p_{n-2}\rangle \\
&= 0,
\end{aligned}$$

and, for $i = 1, \ldots, n-3$,

$$\begin{aligned}
\langle p_n, p_i \rangle &= \langle (x-a_n)p_{n-1} - b_n p_{n-2}, p_i \rangle \\
&= \langle x p_{n-1}, p_i \rangle - a_n \langle p_{n-1}, p_i \rangle - b_n \langle p_{n-2}, p_i \rangle \\
&= \langle x p_{n-1}, p_i \rangle \\
&= \langle p_{n-1}, x p_i \rangle \\
&= \langle p_{n-1}, x p_i - a_{i+1} p_i - b_{i+1} p_{i-1} + a_{i+1} p_i + b_{i+1} p_{i-1} \rangle \\
&= \langle p_{n-1}, p_{i+1} + a_{i+1} p_i + b_{i+1} p_{i-1} \rangle \\
&= \langle p_{n-1}, p_{i+1} \rangle + a_{i+1} \langle p_{n-1}, p_i \rangle + b_{i+1} \langle p_{n-1}, p_{i-1} \rangle \\
&= 0.
\end{aligned}$$

Finally, when $i = 0$,

$$\langle p_n, p_0 \rangle = \langle (x-a_n)p_{n-1} - b_n p_{n-2}, p_0 \rangle = \langle x p_{n-1}, p_0 \rangle = \langle p_{n-1}, x p_0 \rangle = \langle p_{n-1}, p_1 + a_1 p_0 \rangle = 0.$$

Hence the theorem is proved. ■

Corollary 7.1 *For any $n > 0$, the set of polynomials $\{p_0, p_1, \ldots, p_n\}$ given in the previous theorem is linearly independent and*

$$\langle q, p_n \rangle = \int_a^b q(x) p_n(x) w(x)\, dx = 0$$

for any polynomial $q(x)$ with $\deg(q(x)) < n$.

Proof: Suppose $\alpha_0, \alpha_1, \ldots, \alpha_n$ are real numbers such that

$$P(x) = \alpha_0 p_0(x) + \alpha_1 p_1(x) + \cdots + \alpha_n p_n(x) = 0.$$

This means that the coefficients of the terms $1, x, x^2, \ldots x^n$ are all zero. Since $\deg(p_j) = j$ and $\alpha_n p_n$ is the only term in $P(x)$ which contains the x^n term, this implies $\alpha_n = 0$. Hence $P(x)$ can be reduced to

$$P(x) = \sum_{j=0}^{n-1} \alpha_j p_j(x) = 0.$$

Similarly, we have $\alpha_{n-1} = 0, \alpha_{n-2} = 0, \ldots, \alpha_0 = 0$. Therefore, $\{p_0, p_1, \ldots, p_n\}$ is linearly independent.

Suppose $\deg(q) = k < n$. Write

$$q(x) = a_0 p_0(x) + a_1 p_1(x) + \cdots + a_k p_k(x) = \sum_{j=0}^{k} a_j p_j(x),$$

for some real numbers $a_0, a_1, \dots, a_k$. Then

$$\begin{aligned}
\langle q, p_n \rangle &= \int_a^b q(x) p_n(x) w(x)\, dx \\
&= \int_a^b \left(\sum_{j=0}^{k} a_j p_j(x) \right) p_n(x) w(x)\, dx = \sum_{j=0}^{k} \int_a^b p_j(x) p_n(x) w(x)\, dx = 0.
\end{aligned}$$

■

Let Π_n denote the set of polynomials of degree at most n, that is,

$$\Pi_n = \{p(x) \,|\, p(x) \text{ is a polynomial and } \deg(p) \le n\}.$$

Definition 7.3 *Let w be a positive weight function and $f(x)$ be any nonzero continuous function. We say f is w-orthogonal to Π_n, and denote $f \perp_w \Pi_n$, if*

$$\langle f, p \rangle = \int_a^b f(x) p(x) w(x)\, dx = 0,$$

for any $p(x) \in \Pi_n$.

Theorem 7.4 *Let $q(x)$ be any nonzero polynomial of degree $n+1$, and $q(x) \perp_w \Pi_n$. If $x_0, x_1, \dots, x_n$ are the roots of $q(x)$ in $[a, b]$, and*

$$c_i = \int_a^b w(x) \prod_{\substack{j=0 \\ j \ne i}}^{n} \frac{x - x_j}{x_i - x_j}\, dx,$$

then

$$\int_a^b f(x) w(x)\, dx = \sum_{i=0}^{n} c_i f(x_i),$$

for any $f \in \Pi_{2n+1}$. That is, the quadrature rule is exact for any polynomial of degree $\le 2n+1$.

Proof: For any polynomial $f \in \Pi_{2n+1}$, we can write

$$f(x) = q(x) p(x) + r(x),$$

where $p(x), r(x) \in \Pi_n$. Since $x_0, x_1, \dots, x_n$ are roots of $q(x)$, we have

$$f(x_i) = q(x_i) p(x_i) + r(x_i) = r(x_i), \qquad i = 0, 1, \dots, n.$$

By assumption, $q \perp_w \Pi_n$, we have

$$\langle q, p \rangle = \int_a^b q(x) p(x) w(x)\, dx = 0.$$

Since $r(x) \in \Pi_n$, it can be expressed exactly in the Lagrange form

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

Hence

$$\begin{aligned}
\int_a^b f(x)w(x)\,dx &= \int_a^b q(x)p(x)w(x)\,dx + \int_a^b r(x)w(x)\,dx \\
&= \int_a^b r(x)w(x)\,dx = \int_a^b w(x) \sum_{i=0}^{n} r(x_i) \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}\,dx \\
&= \sum_{i=0}^{n} r(x_i) \int_a^b w(x) \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} = \sum_{i=0}^{n} f(x_i) \int_a^b w(x) \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} \\
&= \sum_{i=0}^{n} c_i f(x_i).
\end{aligned}$$

■

Next theorem shows that the roots of $q(x)$ are simple roots and lie in the interior of the interval $[a, b]$.

Theorem 7.5 *Let $w \in C[a, b]$ be a positive weight function, and $f \in C[a, b]$ be a nonzero function such that $f \perp_w \Pi_n$. Then f changes sign at least $n + 1$ times on (a, b).*

Proof: Since $1 \in \Pi_n$ and $f \perp_w \Pi_n$, we have

$$\int_a^b f(x)w(x)\,dx = 0,$$

and this shows that f changes sign at least once in $[a, b]$.

Suppose that f only changes sign m times and $m \leq n$. Let $t_0 = a < t_1 < \cdots < t_m < t_{m+1} = b$ such that f remains the same sign in each interval $(t_0, t_1), (t_1, t_2), \ldots, (t_m, t_{m+1})$. Define an m-th degree polynomial as follows

$$p(x) = (x - t_1)(x - t_2)\cdots(x - t_m) = \prod_{i=1}^{m} (x - t_i).$$

Then $p(x)$ has the same sign property as f. This implies that

$$\int_a^b f(x)p(x)w(x)\,dx \neq 0.$$

However, $\deg(p) = m \leq n$, hence $p \in \Pi_n$. The above conclusion contradicts to the assumption that

$$f \perp_w \Pi_n \quad \Longrightarrow \quad \int_a^b f(x)p(x)w(x)\,dx = 0.$$

Therefore f changes sign at least $n+1$ times in (a, b). ■

These theorems show that the quadrature rule (7.51) is exact for polynomials of degree $\leq 2n+1$ if the nodes $x_0, x_1, \ldots, x_n$ are chosen to be the distinct roots of a polynomial of degree $n+1$ which is w-orthogonal to Π_n.

7.3.2 Gaussian Quadrature Rule

If the positive weight function $w(x) \equiv 1$ and the interval is $[-1, 1]$, then we can use Theorem with inner product

$$\langle f, g \rangle = \int_{-1}^{1} f(x) g(x)\, dx$$

to obtain a set of orthogonal polynomials called the Lengendre polynomials. The first few Lengendre polynomials are

$$\begin{aligned}
p_0(x) &= 1 \\
p_1(x) &= x \\
p_2(x) &= x^2 - \frac{1}{3} \\
p_3(x) &= x^3 - \frac{3}{5}x \\
p_4(x) &= x^4 - \frac{6}{7}x^2 + \frac{3}{35} \\
p_5(x) &= x^5 - \frac{10}{9}x^3 + \frac{5}{21}x
\end{aligned}$$

For a given integer n, the Lengendre polynomial p_{n+1} is a monic polynomial of degree $n+1$ and is orthogonal to the set of first n Lengendre polynomials $\{p_0, p_1, \ldots, p_n\}$. By theorem, $\{p_0, p_1, \ldots, p_n\}$ is linearly independent, and, hence, is a basis for the vector space Π_n. This implies that $p_{n+1} \perp \Pi_n$. Thus, by theorem, the quadrature rule

$$\int_{-1}^{1} f(x)\, dx \approx \sum_{i=0}^{n} c_i f(x_i)$$

will be exact for $f \in \Pi_{2n+1}$ if the nodes $x_0, x_1, \ldots, x_n$ are chosen to be the roots of p_{n+1}. That is, we proved the following theorem.

Theorem 7.6 *Suppose that $x_0, x_1, \ldots, x_n$ are the roots of the $(n+1)$-st Lengendre polynomial p_{n+1}, and that for each $i = 0, 1, \ldots, n$,*

$$c_i = \int_{-1}^{1} \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}\, dx.$$

If $f(x)$ is any polynomial of degree $\leq 2n+1$, then

$$\int_{-1}^{1} f(x)\, dx = \sum_{i=0}^{n} c_i f(x_i).$$

The discussion above gives the Gaussian quadrature rule.

Gaussian Quadrature Rule: For a given function $f(x) \in C[-1,1]$ and integer n,

$$\int_{-1}^{1} f(x)\,dx \approx \sum_{i=0}^{n} c_i f(x_I), \tag{7.53}$$

where $x_0, x_1, \ldots, x_n$ are the roots of the $(n+1)$-st Lengendre polynomial p_{n+1}, and

$$c_i = \int_{-1}^{1} \prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}\,dx. \qquad i = 0, 1, \ldots, n.$$

The following table give some frequently used values for x_i and c_i.

n	x_i	c_i
0	$x_0 = 0$	$c_0 = 2$
1	$x_0 = -0.5773502692$	$c_0 = c_1 = 1$
	$x_1 = 0.5773502692$	
2	$x_0 = -0.7745966692$	$c_0 = \frac{5}{9}$
	$x_1 = 0$	$c_1 = \frac{8}{9}$
	$x_2 = 0.7745966692$	$c_2 = \frac{5}{9}$
3	$x_0 = -0.8611363116$	$c_0 = 0.3478548451$
	$x_1 = -0.3399810436$	$c_1 = 0.6521451549$
	$x_2 = 0.3399810436$	$c_2 = 0.6521451549$
	$x_3 = 0.8611363116$	$c_3 = 0.3478548451$
4	$x_0 = -0.9061798459$	$c_0 = 0.2369268851$
	$x_1 = -0.5384693101$	$c_1 = 0.4786286705$
	$x_2 = 0$	$c_2 = \frac{128}{225} = 0.568888889$
	$x_3 = 0.5384693101$	$c_3 = 0.4786286705$
	$x_4 = 0.9061798459$	$c_4 = 0.2369268851$

A definite integral over an arbitrary interval $[a,b]$ can be transformed into an integral over $[-1,1]$ by using a simple change of variable technique:

$$t = \frac{2x - a - b}{b - a} \qquad \Longleftrightarrow \qquad x = \frac{1}{2}\left[(b-a)t + a + b\right]. \tag{7.54}$$

The idea of Gaussian quadrature rule is not restricted to the use of the Lengendre polynomials. For a given weight function $w(x)$ and interval $[a,b]$, the nodes $x_0, x_1, \ldots, x_n$ can be chosen to be the roots of any monic polynomial q_{n+1} of degree $n+1$ which is w-orthogonal to Π_n in the sense that

$$\int_a^b q_{n+1}(x)p(x)w(x)\,dx = 0, \qquad \text{for all } p \in \Pi_n.$$

The computation of the coefficients c_i in a Gaussian quadrature formula proceeds in the manner already indicated for non-Gaussian formulas once the nodes x_i have been determined. That is,

$$c_i = \int_a^b w(x) \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}\, dx, \tag{7.55}$$

and

$$\int_a^b f(x)w(x)\, dx \approx \sum_{i=0}^n c_i f(x_i). \tag{7.56}$$

An interesting characteristic of the Gaussian quadrature formulas can be verified immediately.

Lemma 7.2 *In a Gaussian quadrature formula, the coefficients c_i are positive and their sum is $\int_a^b w(x)\, dx$.*

Proof: For any $0 \leq k \leq n$, let

$$p(x) = \frac{q_{n+1}(x)}{x - x_k}.$$

Since the polynomial q_{n+1} is of degree $n+1$, the polynomial $p^2(x)$ is of degree at most $2n$. The Gaussian quadrature rule will be exact, that is,

$$0 < \int_a^b p^2(x)w(x)\, dx = \sum_{i=0}^n c_i p^2(x_i) = c_k p^2(x_k).$$

This shows that $c_k > 0$ for any $0 \leq k \leq n$. Since the Gaussian quadrature is of course exact for $f(x) = 1$,

$$\int_a^b f(x)w(x)\, dx = \int_a^b w(x)\, dx = \sum_{i=0}^n c_i f(x_i) = \sum_{i=0}^n c_i.$$

■

Note that in the Gaussian quadrature formulas the interval $[a, b]$ can be infinite, e.g., $[0, \infty)$ or $(-\infty, \infty)$. Some other important cases which lead to Gaussian integration rules are listed in the following table.

$[a, b]$	$w(x)$	Orthogonal polynomials
$[-1, 1]$	$(1 - x^2)^{-1/2}$	$T_n(x)$, Chebyshev polynomials
$[0, \infty)$	e^{-x}	$L_n(x)$, Laguerre polynomials
$(-\infty, \infty)$	e^{-x^2}	$H_n(x)$, Hermite polynomials

7.3.3 Error Analysis

Theorem 7.7 *Consider a Gaussian quadrature formula with error term*

$$\int_a^b f(x)w(x)\,dx = \sum_{i=0}^{n} c_i f(x_i) + E. \tag{7.57}$$

For any $f \in C^{2n+2}[a,b]$, we have

$$E = \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\int_a^b q^2(x)w(x)\,dx, \qquad \text{where } \xi \in (a,b) \text{ and } q(x) = \prod_{i=0}^{n}(x - x_i). \tag{7.58}$$

Proof: By Hermite interpolation, there is a unique polynomial $p(x)$ of degree at most $2n+1$ such that

$$p(x_i) = f(x_i), \qquad p'(x_i) = f'(x_i), \qquad i = 0, 1, \ldots, n.$$

The error for this interpolation is

$$f(x) - p(x) = \frac{1}{(2n+2)!} f^{(2n+2)}(\xi_x)\prod_{i=0}^{n}(x - x_i)^2 = \frac{1}{(2n+2)!} f^{(2n+2)}(\xi_x) q^2(x),$$

where $\xi_x \in (a,b)$ depends on x. It follows that

$$\int_a^b f(x)w(x)\,dx - \int_a^b p(x)w(x)\,dx = \frac{1}{(2n+2)!}\int_a^b f^{(2n+2)}(\xi_x)q^2(x)w(x)\,dx.$$

Since $\deg(p) \leq 2n+1$, the Guassian quadrature formula is exact for $p(x)$,

$$\int_a^b p(x)w(x)\,dx = \sum_{i=0}^{n} c_i p(x_i) = \sum_{i=0}^{n} c_i f(x_i).$$

Furthermore, $f \in C^{2n+2}[a,b]$, and $w(x)$ is a positive weight function, $q^2(x)w(x) > 0$ does not change sign in $[a,b]$, by the weighted Mean-Value Theorem for integral, there exist $\xi \in (a,b)$ such that

$$\int_a^b f^{(2n+2)}(\xi_x)q^2(x)w(x)\,dx = f^{(2n+2)}(\xi)\int_a^b q^2(x)w(x)\,dx.$$

Therefore,

$$\int_a^b f(x)w(x)\,dx = \sum_{i=0}^{n} c_i f(x_i) + \frac{f^{(2n+2)}(\xi)}{(2n+2)!}\int_a^b q^2(x)w(x)\,dx.$$

∎

7.4 Adaptive Quadrature

7.5 Romberg Integration

Recall the trapezoidal rule integral formulation

$$\begin{aligned}\int_a^b f(x)\,dx &\approx T(n)\\ &= \frac{h}{2}[f(a)+2f(a+h)+2f(a+2h)+\cdots+2f(a+(n-1)h)+f(a+nh)],\end{aligned}$$

where $h=\frac{b-a}{n}$. Observe in the following example that the interval $[a,b]$ is partitioned equally-spaced by $x_0=a, x_1, x_2, x_3$, and $x_4=b$. Let $n=2$ and $h=\frac{b-a}{2n}$. If we only consider the partitions by x_0, x_2, and x_4, and apply the Trapezoidal rule, we have the approximation

$$T(n)=\frac{2h}{2}[f(a)+2f(a+2h)+f(a+4h)]=h[f(a)+2f(a+2h)+f(a+4h)].$$

On the other hand, if we apply Trapezoidal rule to the partitions by all points x_0, x_1, x_2, x_3, and x_4, then we have

$$\begin{aligned}T(2n) &= \frac{h}{2}[f(a)+2f(a+h)+2f(a+2h)+2f(a+3h)+f(a+4h)]\\ &= \frac{1}{2}T(n)+h[f(a+h)+f(a+3h)].\end{aligned}$$

This observation shows that if we have computed $T(n)$ and the step size is halfed, we don't have to compute all the function values all over again, but just at those newly added points. In general, suppose $T(n)$ has been computed and the step size becomes $h=\frac{b-a}{2n}$, then

$$T(2n)=\frac{1}{2}T(n)+h\sum_{i=1}^{n}f(a+(2i-1)h). \tag{7.59}$$

With this idea in mind, we can apply the trapezoidal rule recursively, i.e., we partition the interval $[a,b]$ into 2^n subintervals with $n=1,2,3,\ldots$, and the integral formulation becomes

$$T(2^n)=\frac{1}{2}T(2^{n-1})+\frac{b-a}{2^n}\sum_{i=1}^{2^{n-1}}f(a+(2i-1)\frac{b-a}{2^n}) \tag{7.60}$$

This is called the Romberg algorithm.

Algorithm 7.1 (Romberg Integration Algorithm) *Use the Romberg algorithm to evaluate* $\int_a^b f(x)\,dx$.

$R(0,0) = \frac{1}{2}(b-a)[f(a)+f(b)]$
for $n = 1, 2, \ldots, M$ **do**
  $R(n,0) = \frac{1}{2}R(n-1,0) + \frac{b-a}{2^n}\sum_{i=1}^{2^{n-1}} f(a+(2i-1)\frac{b-a}{2^n})$
end for
for $k = 1, 2, \ldots, M$ **do**
  for $n = k, k+1, \ldots, M$ **do**
    $R(n,k) = R(n,k-1) + \frac{1}{4^k-1}[R(n,k-1) - R(n-1,k-1)]$
  end for
end for

Remarks 7.1 *1. The Romberg algorithm produces a table looks like the following.*

2. M is modest, not too large.

3. No duplicate function evaluations in $R(0,0), R(1,0), \ldots, R(M,0)$.

4. No function evaluation is needed im computing $R(m,k)$ when $k \geq 1$.

Theorem 7.8 *If $f \in C[a,b]$, then for each column k,*

$$\lim_{n\to\infty} R(n,k) = \int_a^b f(x)\,dx. \tag{7.61}$$

Moreover, if $f \in C^{2m}[a,b]$, then $R(n,m)$ converges to $\int_a^b f(x)\,dx$ with a rate of $O(h^{2m})$, where $h = \frac{b-a}{2^n}$.

Chapter 8

Numerical Solutions of Ordinary Differential Equations

In this chapter, we discuss numerical methods for solving ordinary differential equations of initial-value problems (IVP) of the form

$$\begin{cases} y' = f(x, y), & x \in \mathbb{R} \\ y(x_0) = y_0, \end{cases} \tag{8.1}$$

where y is a function of x, f is a function of y and x, x_0 is called the initial point, and y_0 the initial value. The numerical values of $y(x)$ on an interval containing x_0 are to be determined.

8.1 Existence and Uniqueness of Solutions

Theorem 8.1 *If $f(x, y)$ is continuous in a region Ω, where*

$$\Omega = \{(x, y) |\ |x - x_0| \le \alpha,\ |y - y_0| \le \beta\} \tag{8.2}$$

then the IVP (8.1) has a solution $y(x)$ for $|x - x_0| \le \min\{\alpha, \dfrac{\beta}{M}\}$, where $M = \max\limits_{(x,y)\in\Omega} |f(x, y)|$.

Theorem 8.2 *If f and $\frac{\partial f}{\partial x}$ are continuous in Ω, then the IVP (8.1) has a unique solution in the interval $|x - x_0| \le \min\{\alpha, \frac{\beta}{M}\}$.*

Theorem 8.3 *If f is continuous in $a \le x \le b$, $-\infty < y < \infty$ and*

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

for some positive constant L, (that is, f is Lipschitz continuous in y), then IVP (8.1) has a unique solution in the interval $[a, b]$.

8.2 Euler's Method

On of the simplest methods for solving the IVP (8.1) is the Euler's method. Note that the slope of the tangent line at x_0 is

$$y'(x_0) = y_0' = \frac{y_1 - y_0}{h},$$

hence

$$y_1 = y_0 + hy_0' = y_0 + hf(x_0, y_0)).$$

Repeat this idea at (x_1, y_1) and we get

$$y_2 = y_1 + hy_1' = y_1 + hf(x_1, y_1),$$

and so on. If x_i are equally spaced, and h is the increment then we have the formulation of Euler's mehtod

$$\begin{aligned} x_{k+1} &= x_k + h = x_0 + (k+1)h \\ y_{k+1} &= y_k + hf(x_k, y_k) \end{aligned}$$

Euler's method is a first-order ($O(h)$) method.

The Improved Euler's method

$$\begin{aligned} x_{k+1} &= x_k + h = x_0 + (k+1)h \\ y^*_{k+1} &= y_k + hf(x_k, y_k) \\ y_{k+1} &= y_k + \frac{h}{2}\left[f(x_k, y_k) + f(x_{k+1}, y^*_{k+1})\right] \end{aligned} \tag{8.3}$$

8.3 Runge-Kutta Methods

One of the most important methods for solving the IVP (8.1) is the Runge-Kutta method. Recall the Taylor's Theorem

$$\begin{aligned} y(x+h) &= y(x) + hy^{'}(x) + \frac{h^2}{2}y^{''} + \frac{h^3}{2}y^{'''} + \ldots \\ &= y(x) + hy^{'}(x) + \frac{h^2}{2}y^{''} + O(h^3) \end{aligned}$$

By differentiating $y(x)$, we have

$$\begin{aligned} y^{'} &= f(x,y) \equiv f \\ y^{''} &= \frac{d}{dx}f = f_x + f_y y^{'} = f_x + f_y f \\ y^{'''} &= f_{xx} + f\frac{d}{dx}f_y + f_y\frac{d}{dx}f \\ &= f_{xx} + f(\frac{f_y}{dy}\frac{dy}{dx} + f_{yx}) + f_y(f_x + f_y f) \\ &= f_{xx} + f(f_{yx} + f_{yy}f) + f_y(f_x + f_y f) \end{aligned}$$

plug in, we get

$$\begin{aligned}
y(x+h) &= y + hy' + \frac{h^2}{2}y'' + O(h^3) \\
&= y + hf + \frac{h^2}{2}(f_x + f_y f) + O(h^3) \\
&= y + \frac{h}{2}f + \frac{h}{2}f + \frac{h^2}{2}(f_x + f_y f) + O(h^3) \\
&= y + \frac{h}{2}f + \frac{h}{2}(f + hf_x + hf_y f) + O(h^3)
\end{aligned}$$

Apply Taylor's Theorem on f

$$\begin{aligned}
& f(x+h, y+hf) = f(x,y) + hf_x + hff_y + O(h^2) \\
\Rightarrow\ & f + hf_x + hff_y \approx f(x+h, y+hf) \\
\Rightarrow\ & y(x+h) = y + \frac{1}{2}hf + \frac{h}{2}f(x+h, y+hf) + O(h^3)
\end{aligned}$$

Let

$$F_1 = hf(x,y) \qquad F_2 = hf(x+h, y+F_1)$$

Then

$$y(x+h) = y + \frac{1}{2}(F_1 + F_2)$$

This is called the second-order Runge-Kutta method. Two function evaluations are required at each step.

Algorithm for second-order Runge-Kutta mehtod :

for $k = 0, 1, 2, \ldots$ **do**
  $x_{k+1} = x_k + h = x_0 + (k+1)h$
  $F_1 = hf(x_k, y_k)$
  $F_2 = hf(x_{k+1}, y_k + F_1)$
  $y_{k+1} = y_k + \frac{1}{2}(F_1 + F_2)$
end for

General form of second-order Runge-Kutta method :

$$y(x+h) = y + \omega_1 hf + \omega_2 hf(x + \alpha h, y + \beta hf) + O(h^3)$$

where $\omega_1, \omega_2, \alpha, \beta$ are constants to be defined, and

$$\begin{cases} \omega_1 + \omega_2 = 1 \\ \omega_2 \alpha = \frac{1}{2} \\ \omega_2 \beta = \frac{1}{2} \end{cases} .$$

By letting $\omega_1 = 0, \omega_2 = 1, \alpha = \beta = \frac{1}{2}$ leads to the modified Euler's mehtod.

Fourth-Order Runge-kutta method

$$y(x+h) = y + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) + O(h^5)$$

where

$$\begin{cases} F_1 = hf(x, y) \\ F_2 = hf(x + \frac{1}{2}h, y + \frac{1}{2}F_1) \\ F_3 = hf(x + \frac{1}{2}h, y + \frac{1}{2}F_2) \\ F_4 = hf(x + h, y + F_3) \end{cases}$$

Algorithm for fourth-order Runge-Kutta mehtod :

for $k = 0, 1, 2, \ldots$ **do**
  $x_{k+\frac{1}{2}} = x_k + \frac{1}{2}h$
  $x_{k+1} = x_k + h = x_0 + (k+1)h$
  $F_1 = hf(x_k, y_k)$
  $F_2 = hf(x_{k+\frac{1}{2}}, y_k + \frac{1}{2}F_1)$
  $F_3 = hf(x_{k+\frac{1}{2}}, y_k + \frac{1}{2}F_2)$
  $F_4 = hf(x_{k+1}, y_k + F_3)$
  $y_{k+1} = y_k + \frac{1}{6}(F_1 + 2F_2 + 2F_3 + F_4) + O(h^5)$
end for

Local truncation error in the Runge-Kutta mehtod is the error that arises in each step simply because of the truncated Taylor series. This error is inevitable. The fourth Runge-Kutta method involves a local truncation error of $O(h^5)$.

The Number of function evaluations in the second order method is two and the fourth order is four.

8.4 Systems and Higher-Order Ordinary Differential Equations

Consider a system of first-order ODE's.

$$\begin{cases} y_1' &= f_1(x, y_1, y_2, \ldots, y_n) \\ y_2' &= f_2(x, y_1, y_2, \ldots, y_n) \\ &\vdots \\ y_n' &= f_n(x, y_1, y_2, \ldots, y_n) \end{cases}$$

with initial conditions

$$\begin{cases} y_1(x_0) &= y_1^0 \\ y_2(x_0) &= y_2^0 \\ &\vdots \\ y_n(x_0) &= y_n^0 \end{cases}$$

Define

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \qquad F = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}, \qquad Y_0 = \begin{bmatrix} y_1^0 \\ y_2^0 \\ \vdots \\ y_n^0 \end{bmatrix}$$

and transform the system into vector form

$$\begin{cases} Y' & = F(x, Y) \\ Y(x_0) & = Y_0 \end{cases}$$

The Runge-Kutta methods can be easily extended to vector form.

Example 8.1

$$\begin{cases} y_1' & = y_1 + 4y_2 - e^x \\ y_2' & = y_1 + y_2 + 2e^x \end{cases} \qquad \textit{with initial conditions} \qquad \begin{cases} y_1(0) & = 4 \\ y_2(0) & = \frac{5}{4} \end{cases}$$

Sol: Transform to

$$\begin{aligned} Y' &= \begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_1 + 4y_2 - e^x \\ y_1 + y_2 + 2e^x \end{bmatrix} = F(x, Y) \\ Y_0 &= \begin{bmatrix} 4 \\ \frac{5}{4} \end{bmatrix}, \qquad x_0 = 0 \end{aligned}$$

■

Higher-Order ODE's. An higher order ODE can be converted into a system of first-order equations, hence higher-order ODEs can be solved in vector form.

Example 8.2

$$\begin{cases} (\sin x)y''' + \cos(xy) + \sin(x^2 + y'') + (y')^3 = \log x \\ y(2) = 7 \\ y'(2) = 3 \\ y''(2) = -4 \end{cases}$$

Sol: Let $u_1 = y, u_2 = y'$, and $u_3 = y''$

$$\begin{aligned} & (\sin x)y''' + \cos(xy) + \sin(x^2 + y'') + (y')^3 = \log x \\ \Rightarrow \quad & (\sin x)u_3' + \cos(xu_1) + \sin(x^2 + u_3) + (u_2)^3 = \log x \\ \Rightarrow \quad & u_3' = \left[\log x - \cos(xu_1) - \sin(x^2 + u_3) - (u_2)^3\right] / \sin x \end{aligned}$$

system:

$$\begin{cases} u_1' &= u_2 \\ u_2' &= u_3 \\ u_3' &= [\log x - \cos(xu_1) - \sin(x^2 + u_3) - u_2^3] / \sin x \end{cases} \qquad \begin{cases} u_1(2) &= 7 \\ u_2(2) &= 3 \\ u_3(2) &= -4 \end{cases}$$

let

$$U = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix}, \qquad F = \begin{bmatrix} u_2 \\ u_3 \\ [\log x - \cos(xu_1) - \sin(x^2 + u_3) - u_2^3] \,/ \sin x \end{bmatrix},$$

and

$$U_0 = \begin{bmatrix} 7 \\ 3 \\ -4 \end{bmatrix}, \qquad x_0 = 2.$$

The higher-order ODE becomes

$$\begin{cases} U' &= F(x, U) \\ U(x_0) &= U_0 \end{cases}$$

and can be solved by Runge-Kutta methods. ∎

Chapter 9

Boundary-Value Problems for Ordinary Differential Equations

Physical problems that are position-dependent rather than time dependent are often modeled in terms of differential equations with boundary conditions imposed at more than one point. The two-point boundary-value problems (BVP) considered in this chapter involve a second-order differential equation together with boundary condition in the following form:

$$\begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha, \qquad y(b) = \beta \end{cases} \tag{9.1}$$

The numerical procedures for finding approximate solutions to the initial-value problems can not be adapted to the solution of this problem since the specification of conditions involve two different points, $x = a$ and $x = b$. New techniques are introduced in this chapter for handling problems (9.1) in which the conditions imposed are of a boundary-value rather than an initial-value type.

9.1 Mathematical Theories

Before considering numerical methods, a few mathematical theories about the two-point boundary-value problem (9.1), such as the existence and uniqueness of solution, shall be discussed in this section. Existence theorems for solutions of (9.1) tend to be rather complicated. The following results give some insight into this aspect.

Theorem 9.1 *Suppose that the function f in the two-point boundary-value problem (9.1) is continuous on the set*

$$D = \{(x, y, y')|a \le x \le b, \, -\infty < y < \infty, \, -\infty < y' < \infty\},$$

and that $\dfrac{\partial f}{\partial y}$ and $\dfrac{\partial f}{\partial y'}$ are also continuous on D. If

1. $\dfrac{\partial f}{\partial y}(x, y, y') > 0$ *for all $(x, y, y') \in D$, and*

2. *a constant M exists, with* $\left|\dfrac{\partial f}{\partial y'}(x, y, y')\right| \leq M$, *for all* $(x, y, y') \in D$,

then the two-point boundary-value problem (9.1) has a unique solution.

When the function $f(x, y, y')$ has the special form

$$f(x, y, y') = p(x)y' + q(x)y + r(x), \tag{9.2}$$

the differential equation become a so-called linear problem. The previous theorem can be simplified for this case.

Corollary 9.1 *If the linear two-point boundary-value problem*

$$\begin{cases} y'' = p(x)y' + q(x)y + r(x) \\ y(a) = \alpha, \qquad y(b) = \beta \end{cases} \tag{9.3}$$

satisfies

1. $p(x), q(x)$, *and* $r(x)$ *are continuous on* $[a, b]$, *and*
2. $q(x) > 0$ *on* $[a, b]$,

then the problem has a unique solution.

Many theories and application models consider the boundary-value problem in a special form as follows.

$$\begin{cases} y'' = f(x, y) \\ y(0) = 0, \qquad y(1) = 0 \end{cases} \tag{9.4}$$

We will show that this simple form can be derived from the original problem by simple techniques such as changes of variables and linear transformation. To do this, we begin by changing the variable. Suppose that the original problem is

$$\begin{cases} y'' = f(x, y) \\ y(a) = \alpha, \qquad y(b) = \beta \end{cases} \tag{9.5}$$

where $y = y(x)$. Now let $\lambda = b - a$ and define a new variable

$$t = \frac{x - a}{b - a} = \frac{1}{\lambda}(x - a).$$

That is, $x = a + \lambda t$. Notice that $t = 0$ corresponds to $x = a$, and $t = 1$ corresponds to $x = b$. Then we define

$$z(t) = y(a + \lambda t) = y(x)$$

with $\lambda = b - a$. This gives

$$z'(t) = \frac{d}{dt}z(t) = \frac{d}{dt}y(a + \lambda t) = \left[\frac{d}{dx}y(x)\right]\left[\frac{d}{dt}(a + \lambda t)\right] = \lambda y'(x)$$

and, analogously,

$$z''(t) = \frac{d}{dt}z'(t) = \lambda^2 y''(x) = \lambda^2 f(x, y(x)) = \lambda^2 f(a + \lambda t, z(t)).$$

Likewise the boundary conditions are changed to

$$z(0) = y(a) = \alpha \quad \text{and} \quad z(1) = y(b) = \beta.$$

With all these together, the problem (9.5) is transformed into

$$\begin{cases} z''(t) = \lambda^2 f(a + \lambda t, z(t)) \\ z(0) = \alpha, \qquad z(1) = \beta \end{cases} \tag{9.6}$$

Thus, if $y(x)$ is a solution for (9.5), then $z(t) = y(a+\lambda t)$ is a solution for the boundary-value problem (9.6). Conversely, if $z(t)$ is a solution for (9.6), then $y(x) = z(\frac{1}{\lambda}(x-a))$ is a solution for (9.5).

Theorem 9.2 *Consider the two-point boundary-value problems*

$$\begin{cases} y''(x) = f(x, y) \\ y(a) = \alpha, \qquad y(b) = \beta \end{cases} \tag{9.7}$$

and

$$\begin{cases} z''(t) = g(t, z) \\ z(0) = \alpha, \qquad z(1) = \beta \end{cases} \tag{9.8}$$

in which

$$g(p, q) = (b - a)^2 f(a + (b - a)p, q).$$

Then

1. *if $z(t)$ is a solution of (9.8), then the function defined by $y(x) = z((x - a)/(b - a))$ is a solution of (9.7), and*

2. *if $y(x)$ is a solution of (9.7), then $z(t) = y(a + (b - a)t)$ is a solution of (9.8).*

Example 9.1 *Simplify the boundary conditions of the following equation by use of changing variables.*

$$\begin{cases} y'' = \sin(xy) + y^2 \\ y(1) = 3, \qquad y(4) = 7 \end{cases}$$

Sol: In this problem $a = 1, b = 4$, hence $\lambda = 3$. Now define the new variable $t = \frac{1}{3}(x-1)$, hence $x = 1 + 3t$, and let $z(t) = y(x) = y(1 + 3t)$. Then

$$\lambda^2 f(a + \lambda t, z) = 9\left[\sin(1 + 3t)z + z^2\right],$$

and the original equation is reduced to

$$\begin{cases} z''(t) = 9\sin((1+3t)z) + 9z^2 \\ z(0) = 3, \qquad z(1) = 7 \end{cases}$$

■

To reduce a two-point boundary-value problem

$$\begin{cases} z''(t) = g(t, z) \\ z(0) = \alpha, \qquad z(1) = \beta \end{cases}$$

to a homogeneous system, we simply subtract from z a linear function that takes the values α and β at 0 and 1. That is, let

$$u(t) = z(t) - [\alpha + (\beta - \alpha)t]$$

then $u''(t) = z''(t)$, and

$$u(0) = z(0) - \alpha = 0 \quad \text{and} \quad u(1) = z(1) - \beta = 0$$

Moreover,

$$g(t, z) = g(t, u + \alpha + (\beta - \alpha)t) \equiv h(t, u).$$

The system is now transformed into

$$\begin{cases} u''(t) = h(t, u) \\ u(0) = 0, \qquad u(1) = 0 \end{cases}$$

Here is the theorem.

Theorem 9.3 *Consider the two-point boundary-value problems*

$$\begin{cases} z''(t) = g(t, z) \\ z(0) = \alpha, \qquad z(1) = \beta \end{cases} \tag{9.9}$$

and

$$\begin{cases} u''(t) = h(t, u) \\ u(0) = 0, \qquad u(1) = 0 \end{cases} \tag{9.10}$$

in which

$$h(p, q) = g(p, q + \alpha + (\beta - \alpha)p).$$

Then

1. *if $u(t)$ solves (9.10), then the function $z(t) = u(t) + \alpha + (\beta - \alpha)t$ solves (9.9), and*

2. *if $z(t)$ solves (9.9), then $u(t) = z(t) - [\alpha + (\beta - \alpha)t]$ solves (9.10).*

Example 9.2 *Reduce the system*

$$\begin{cases} z'' = [5z - 10t + 35 + \sin(3z - 6t + 21)]e^t \\ z(0) = -7, \qquad z(1) = -5 \end{cases}$$

to a homogeneous problem by linear transformation technique.

Sol: Let

$$u(t) = z(t) - [-7 + (-5 + 7)t] = z(t) - 2t + 7.$$

Then $z(t) = u(t) + 2t - 7$, and

$$\begin{aligned} u'' = z'' &= [5z - 10t + 35 + \sin(3z - 6t + 21)]e^t \\ &= [5(u + 2t - 7) - 10t + 35 + \sin(3(u + 2t - 7) - 6t + 21)]e^t \\ &= [5u + \sin(3u)]e^t \end{aligned}$$

The system is transformed to

$$\begin{cases} u''(t) = [5u + \sin(3u)]e^t \\ u(0) = u(1) = 0 \end{cases}$$

■

Example 9.3 *Reduce the following two-point boundary-value problem*

$$\begin{cases} y'' = y^2 + 3 - x^2 + xy \\ y(3) = 7, \qquad y(5) = 9 \end{cases}$$

to a homogeneous system.

Sol: In the original system, $a = 3, b = 5, \alpha = 7, \beta = 9$. Let $\lambda = b - a = 2$ and define a new variable

$$t = \frac{1}{2}(x - 3) \quad \Longrightarrow \quad x = 2t + 3.$$

Let the function $z(t) = y(x) = y(2t + 3)$. Then

$$\begin{aligned} z''(t) &= \lambda^2 y''(2t + 3) = \lambda^2 f(2t + 3, u) \\ &= 4[z^2 + 3 - (2t + 3)^2 + (2t + 3)z] \\ &= 4[z^2 + 3z + 2tz - 4t^2 - 12t - 6] \end{aligned}$$

The original problem is first transformed into

$$\begin{cases} z''(t) = 4[z^2 + 3z + 2tz - 4t^2 - 12t - 6] \\ z(0) = 7, \qquad z(1) = 9 \end{cases}$$

Next let

$$u(t) = z(t) - [7 + 2t], \quad \text{or equivalently,} \quad z(t) = u(t) + 2t + 7.$$

Then

$$\begin{aligned} u''(t) &= 4[(z + 2t + 7)^2 + 3(u + 2t + 7) + 2t(u + 2t + 7) - 4t^2 - 12t - 6] \\ &= 4[u^2 + 6tu + 17u + 4t^2 + 36t + 64]. \end{aligned}$$

The original problem is transformed into the following homogeneous system

$$\begin{cases} u''(t) = 4[u^2 + 6tu + 17u + 4t^2 + 36t + 64] \\ u(0) = u(1) = 0 \end{cases}$$

■

When a two-point boundary-value problem is expressed in the homogeneous system from, we have the following elegant results.

Theorem 9.4 *The boundary-value problem*

$$\begin{cases} y'' = f(x, y) \\ y(0) = 0, \qquad y(1) = 0 \end{cases} \tag{9.11}$$

has a unique solution if $\dfrac{\partial f}{\partial y}$ *is continuous, non-negative, and bounded in the strip* $0 \le x \le 1$ *and* $-\infty < y < \infty$.

Theorem 9.5 *If* f *is a continuous function of* (s, t) *in the domain* $0 \le s \le 1$ *and* $-\infty < t < \infty$ *such that*

$$|f(s, t_1) - f(s, t_2)| \le K|t_1 - t_2|, \qquad (K < 8).$$

Then the boundary-value problem

$$\begin{cases} y'' = f(x, y) \\ y(0) = 0, \qquad y(1) = 0 \end{cases}$$

has a unique solution in $C[0, 1]$.

9.2 Finite Difference Method For Linear Problems

We consider finite difference method for solving the linear two-point boundary-value problem of the form

$$\begin{cases} y'' = p(x)y' + q(x)y + r(x) \\ y(a) = \alpha, \qquad y(b) = \beta. \end{cases} \tag{9.12}$$

Methods involving finite differences for solving boundary-value problems replace each of the derivatives in the differential equation by an appropriate difference-quotient approximation. The difference-quotient is chosen to maintain a specified order of truncation error. The linear second-order problem (9.12) requires that difference-quotient approximations be used to approximate both y' and y''.

9.2.1 The Finite Difference Formulation

First, we select an integer $n > 0$ and partition the interval $[a, b]$ into n equally-spaced subintervals by points $a = x_0 < x_1 < \ldots < x_n < x_n = b$. Each mesh point x_i can be computed by

$$x_i = a + i * h, \quad i = 0, 1, \ldots, n,$$

where

$$h = \frac{b-a}{n}$$

is called the mesh size. These mesh points need not be equally spaces. Indeed, if the points are not uniformly distributed, then more complicated formulations will be involved in the derivation that follows. For simplicity, they are usually equally spaced in practice.

At the interior mesh points, x_i, for $i = 1, 2, \ldots, n-1$, the differential equation to be approximated satisfies

$$y''(x_i) = p(x_i)y'(x_i) + q(x_i)y(x_i) + r(x_i). \tag{9.13}$$

The central finite difference formulae

$$y'(x_i) = \frac{y(x_{i+1}) - y(x_{i-1})}{2h} - \frac{h^2}{6}y^{(3)}(\eta_i), \tag{9.14}$$

for some η_i in the interval (x_{i-1}, x_{i+1}), and

$$y''(x_i) = \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2} - \frac{h^2}{12}y^{(4)}(\xi_i), \tag{9.15}$$

for some ξ_i in the interval (x_{i-1}, x_{i+1}), can be derived from Taylor's theorem by expanding y about x_i.

Let u_i denote the approximate value of $y_i = y(x_i)$. If $y \in C^4[a,b]$, then a finite difference method with truncation error of order $O(h^2)$ can be obtained by using the approximations

$$y'(x_i) \approx \frac{u_{i+1} - u_{i-1}}{2h} \quad \text{and} \quad y''(x_i) \approx \frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} \tag{9.16}$$

for $y'(x_i)$ and $y''(x_i)$, respectively. Furthermore, let

$$p_i = p(x_i), \quad q_i = q(x_i), \quad r_i = r(x_i).$$

The discrete version of equation (9.12) is then

$$\frac{u_{i+1} - 2u_i + u_{i-1}}{h^2} = p_i \frac{u_{i+1} - u_{i-1}}{2h} + q_i u_i + r_i, \qquad i = 1, 2, \ldots, n-1, \tag{9.17}$$

together with boundary conditions $u_0 = \alpha$ and $u_n = \beta$. Equation (9.17) can be written in the form

$$\left(1 + \frac{h}{2} p_i\right) u_{i-1} - \left(2 + h^2 q_i\right) u_i + \left(1 - \frac{h}{2} p_i\right) u_{i+1} = h^2 r_i, \qquad i = 1, 2, \ldots, n-1. \tag{9.18}$$

In equation (9.17), $u_1, u_2, \ldots, u_{n-1}$ are the unknown, and there are $n-1$ linear equations to be solved. The resulting system of linear equations can be expressed in the matrix form

$$Au = f, \tag{9.19}$$

where

$$A = \begin{bmatrix} -2 - h^2 q_1 & 1 - \frac{h}{2} p_1 & & & & \\ 1 + \frac{h}{2} p_2 & -2 - h^2 q_2 & 1 - \frac{h}{2} p_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & 1 + \frac{h}{2} p_{n-2} & -2 - h^2 q_{n-2} & 1 - \frac{h}{2} p_{n-2} \\ & & & & 1 + \frac{h}{2} p_{n-1} & -2 - h^2 q_{n-1} \end{bmatrix}, \tag{9.20}$$

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix}, \qquad \text{and} \qquad f = \begin{bmatrix} h^2 r_1 - \left(1 + \frac{h}{2} p_1\right) \alpha \\ h^2 r_2 \\ \vdots \\ h^2 r_{n-2} \\ h^2 r_{n-1} - \left(1 - \frac{h}{2} p_{n-1}\right) \beta \end{bmatrix} \tag{9.21}$$

Since the matrix A is tridiagonal, this system can be solved by a special Gaussian elimination in $O(n^2)$ flops. Note that if $q_i > 0$ for all i (Corollary 9.1), and h is small so that $\left|\frac{h}{2} p_i\right| < 1$ for all i, then A is diagonally dominant, since

$$\left|-2 - h^2 q_i\right| > 2 = \left|1 + \frac{h}{2} p_i\right| + \left|1 - \frac{h}{2} p_i\right|, \tag{9.22}$$

and the linear system (9.19) has a unique solution.

Theorem 9.6 *Suppose that $p(x), q(x)$, and $r(x)$ in the linear two-point boundary-value problem (9.12) are continuous on $[a,b]$, and $q(x) > 0$ on $[a,b]$. Then the tridiagonal linear system (9.19) has a unique solution provided that $h < 2/L$, where $L = \max_{a \le x \le b} |p(x)|$.*

9.2.2 Convergence Analysis

We shall analyze that when h converges to zero, the solution u_i of the discrete problem (9.17) converges to the solution y_i of the original continuous problem (9.13). Note that u_i depends on the mesh size h. We shall estimate $|y_i - u_i|$ and show that it converges to zero as $h \to 0$.

With the aid of formulas (9.14) and (9.15), we see that the exact solution y_i satisfies the following system of equations

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} - \frac{h^2}{12} y^{(4)}(\xi_i) = p_i \left(\frac{y_{i+1} - y_{i-1}}{2h} - \frac{h^2}{6} y^{(3)}(\eta_i) \right) + q_i y_i + r_i, \tag{9.23}$$

for $i = 1, 2, \ldots, n-1$. On the other hand, the discrete solution u_i satisfies the equations (9.17). If we subtract (9.17) from (9.23) and let $e_i = y_i - u_i$, the result is

$$\frac{e_{i+1} - 2e_i + e_{i-1}}{h^2} = p_i \frac{e_{i+1} - e_{i-1}}{2h} + q_i e_i + h^2 g_i, \qquad i = 1, 2, \ldots, n-1,$$

where

$$g_i = \frac{1}{12} y^{(4)}(\xi_i) - \frac{1}{6} p_i y^{(3)}(\eta_i).$$

After collecting terms and multiplying by h^2. We have equations similar to (9.18), namely,

$$\left(1 + \frac{h}{2} p_i\right) e_{i-1} - \left(2 + h^2 q_i\right) e_i + \left(1 - \frac{h}{2} p_i\right) e_{i+1} = h^4 g_i, \qquad i = 1, 2, \ldots, n-1. \tag{9.24}$$

Let $e = [e_1, e_2, \ldots, e_{n-1}]^T$ and k be the index such that $|e_k| = \|e\|_\infty$. Then from (9.24) we have

$$\left(2 + h^2 q_i\right) e_k = \left(1 + \frac{h}{2} p_i\right) e_{k-1} + \left(1 - \frac{h}{2} p_i\right) e_{k+1} - h^4 g_k,$$

and, hence

$$\begin{aligned}
\left|2 + h^2 q_k\right| |e_k| &\leq \left|1 + \frac{h}{2} p_k\right| |e_{k-1}| + \left|1 - \frac{h}{2} p_k\right| |e_{k+1}| + h^4 |g_k| \\
&\leq \left|1 + \frac{h}{2} p_k\right| \|e\|_\infty + \left|1 - \frac{h}{2} p_k\right| \|e\|_\infty + h^4 \|g\|_\infty
\end{aligned}$$

When $q(x) > 0$ for all $x \in [a, b]$ and h is chosen small enough so that $|\frac{h}{2} p_i| < 1$ for all i, then the condition (9.24) holds, and the above inequality induces

$$h^2 q_k \|e\|_\infty \leq h^4 \|g\|_\infty.$$

Therefore, we derive an upper bound for $\|e\|_\infty$

$$\|e\|_\infty \leq h^2 \left(\frac{\|g\|_\infty}{\inf q(x)} \right). \tag{9.25}$$

By the definition of g_i, we have

$$\|g\|_\infty \leq \frac{1}{12} \|y^{(4)}(x)\|_\infty + \frac{1}{6} \|p(x)\|_\infty \|y^{(3)}(x)\|_\infty.$$

Hence $\frac{\|g\|_\infty}{\inf q(x)}$ is a bound independent of h. Thus we can conclude that $\|e\|_\infty$ is $O(h^2)$ as $h \to 0$.

9.2.3 Higher Order Approximations

The finite difference method established in this section ensure that the truncation error has order $O(h^2)$ provided that $y \in C^4[a,b]$. To obtain a finite difference method with greater accuracy, we may proceed in deriving higher order approximations for $y'(x_i)$ and $y''(x_i)$ using high oder of Taylor series. However this requires more mesh points around x_i in the approximation formulas which lead to difficulties at the boundary points. Moreover, the resulting linear system is no longer in tridiagonal form, and the computational cost for the solution is much higher.

Instead of attempting to obtain a difference method with a higher-order truncation error, it is generally more satisfactory to consider a reduction in step size h. In addition, some extrapolation techniques can be applied effectively for this matter, provided that y is sufficiently differentiable.

9.3 Finite Difference Method For Nonlinear Problems

9.4 Shooting Methods

We consider solving the following 2-point boundary-value problem with shooting mehtod:

$$(1) \qquad \begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha, \quad y(b) = \beta \end{cases}$$

The idea of shooting method for the BVP (1) is to solve a related initial-value problem with a guess for $y'(a)$, say z. The corresponding IVP

$$(2) \qquad \begin{cases} y'' = f(x, y, y') \\ y(a) = \alpha, \quad y'(a) = z \end{cases}$$

can then be solved by, for example, Runge-Kutta method. We denote this approximate solution y_z and hope $y_z(b) = \beta$. If not, we use another guess for $y'(a)$, and try to solve an altered IVP (2) again. This process is repeated and can be done systematically.
The objective is to select z, so that $y_z(b) = \beta$.
Let

$$\phi(z) = y_z(b) - \beta.$$

Now our objective is simply to solve the equation $\phi(z) = 0$. Hence secant method can be used.
Note: $\phi(z)$ is very expensive to compute, since each value of $\phi(z)$ is obtained by numerically solving an IVP.
Suppose we have solutions y_{z_1}, y_{z_2} with guesses z_1, z_2 and obtain $\phi(z_1)$ and $\phi(z_2)$. If these guesses can not generate satisfactory solutions, we can obtain another guess z_3 by the secant method

$$z_3 = z_2 - \phi(z_2)\frac{z_2 - z_1}{\phi(z_2) - \phi(z_1)}.$$

In general

$$z_{k+1} = z_k - \phi(z_k)\frac{z_k - z_{k-1}}{\phi(z_k) - \phi(z_{k-1})}.$$

The shooting method can be quite costly in computational effort. One may solve the IVP with a large step size in the first few tries, since high precision is essentially wasted in the first few tries. Only when $\phi(z)$ is close to zero should a small step size be used.
When the BVP has the following special form

$$(3) \qquad \begin{cases} y^{''} = u(x) + v(x)y + w(x)y^{'} \\ y(a) = \alpha, \quad y(b) = \beta \end{cases}$$

where $u(x), v(x), w(x)$ are continuous in $[a, b]$. Then (3) can be solved in at most two tries (one step).
Suppose we have solved (3) twice with initial guesses z_1 and z_2, and obtain approximate solutions y_1 and y_2, hence

$$\begin{cases} y_1^{''} = u + vy_1 + wy_1^{'} \\ y_1(a) = \alpha, \quad y_1^{'}(b) = z_1 \end{cases}$$

and

$$\begin{cases} y_2^{''} = u + vy_2 + wy_2^{'} \\ y_2(a) = \alpha, \quad y_2^{'}(b) = z_2 \end{cases}$$

Now let

$$y(x) = \lambda y_1(x) + (1 - \lambda)y_2(x)$$

for some parameter λ, we can show

$$y^{''} = u + vy + wy^{'}$$

and

$$y(a) = \lambda y_1(a) + (1 - \lambda)y_2(a) = \alpha$$

We can select λ so that $y(b) = \beta$.

$$\begin{aligned} \beta &= y(b) = \lambda y_1(b) + (1 - \lambda)y_2(b) \\ &\qquad = \lambda(y_1(b) - y_2(b)) + y_2(b) \\ \Rightarrow \lambda &= \frac{\beta - y_2(b)}{(y_1(b) - y_2(b))} \end{aligned}$$

In practice, we can solve the following two IVPs (in parallel)

$$\begin{cases} y^{''} = u(x) + v(x)y + w(x)y^{'} \\ y(a) = \alpha, \quad y^{'}(a) = 0 \end{cases}$$

and

$$\begin{cases} y^{''} = u(x) + v(x)y + w(x)y^{'} \\ y(a) = \alpha, \quad y^{'}(a) = 1 \end{cases}$$

to obtain approximate solutions y_1 and y_2, then compute λ and form the solution y. Moreover, the problem can be transformed by solving a system of first-order ODE.
Let

$$\begin{cases} y_0(x) = x \\ y_3(x) = y_1^{'}(x) \\ y_4(x) = y_2^{'}(x) \end{cases}$$

Then

$$\begin{cases} y_0^{'} = 1 \\ y_1^{'} = y_3 \\ y_2^{'} = y_4 \\ y_3^{'} = y_1^{''} = u + vy_1 + wy_1^{'} = u + vy_1 + wy_3 \\ 4_4^{'} = u + vy_2 + wy_4 \end{cases}$$

with I.C.

$$\begin{cases} y_0(a) = a \\ y_1(a) = \alpha \\ y_2(a) = \alpha \\ y_3(a) = y_1^{'}(a) = 0 \\ 4_4(a) = y_2^{'}(a) = 1 \end{cases}$$