Lecture Notes on Numerical Analysis

Author: Windson
Course: Numerical Analysis (I)

Department of Applied Mathematics
National Chung Hsing University

145 Xingda Road, South District, Taichung City 40227, Taiwan (R.O.C.)
e-mail: info AT windson.cc

License
This document is licensed under the Creative Commons Attribution 4.0 International License (CC BY-SA 4.0).

June 15, 2026

Contents

1 About Error **1**

2 Root Finding **1**

2.1 Bisection Method 1

2.1.1 Algorithms 1

2.1.2 Theoretical Foundations 2

2.2 Fixed-Point Iteration 3

2.2.1 Definition & Properties 3

2.2.2 Algorithms: Find Fixed Point 4

2.2.3 Convergence Theorem & Error Analysis 4

2.3 Newton's Method 5

2.3.1 Algorithms 6

2.3.2 Convergence of Newton's Method 6

2.3.3 Error Analysis 7

3 Interpolation and Polynomial approximation **7**

3.1 Lagrange Polynomial 8

3.2 Error Analysis of Lagrange Interpolation 9

3.3 Newton's Divided Difference 10

3.4 Cubic Spline Interpolation 12

4 Numerical Differentiation and Integration **14**

4.1 Numerical Differentiation 14

4.2 Numerical Integration 16

4.3 Adaptive Quadrature Methods 18

4.4 Gaussian Quadrature 19

4.5 Improper Integrals 21

5 Solve the Linear System of Equations **21**

5.1 Matrix Norms and Condition Number 22

5.2 Direct Methods 23

5.2.1 Gaussian Elimination 23

5.2.2 LU Decomposition 26

5.3 Iterative Methods 28

5.3.1 Jacobi Method 28

5.3.2 Gauss-Seidel Method 29

5.3.3 Conjugate Gradient Method (CG Method) 31

6 Approximation Theory **32**
6.1 Discrete Least Squares Approximation 33
6.2 Continuous Least Squares Approximation 34
6.3 Orthogonal Polynomials 35

7 Approximating Eigenvalues **36**
7.1 Singular Value Decomposition (SVD) 36
7.2 Orthogonal Projection 37

8 Multivariable Calculus **39**
8.1 F-Differentiable (Fréchet Derivative) 39
8.2 G-Differentiable (Gâteaux Derivative) 39

9 Systems of Nonlinear Equations **40**
9.1 Fixed-Point Iteration for Systems 40
9.2 Newton's Method for Systems 40
9.3 Inexact Newton's Method and GMRES 41

10 Numerical Solutions for Boundary Value Problems **42**
10.1 Finite Difference Method (FDM) 42
10.2 Finite Element Method (FEM) 42

1 About Error

Definition 1.1 (Order of Convergence)**.**
If a sequence $\{x_n\}_{n=1}^{\infty}$ converges to X^*, we say $\{x_n\}_{n=1}^{\infty}$ converges to X^* of order α with asymptotic error constant λ if:

$$\lim_{n\to\infty} \frac{|x_{n+1} - X^*|}{|x_n - X^*|^\alpha} = \lambda$$

- When $\alpha = 1$, it is called **linearly convergent**.
- When $\alpha = 2$, it is called **quadratically convergent**.

Definition 1.2 (Error Types)**.**
Let P be an approximation for P^*:

- **Absolute error**: $|P - P^*|$
- **Relative error**: $\frac{|P-P^*|}{|P^*|}$, if $|P^*| \neq 0$

2 Root Finding

2.1 Bisection Method

Necessary Condition: f is a continuous function $f \in C[a, b]$ and $f(a)f(b) < 0$.

2.1.1 Algorithms

Given $f(x)$, $f(a)$, $f(b)$ with tolerance ϵ_1, ϵ_2:

```
def bisection(f, a, b, tolerance):
    if f(a) * f(b) > 0:
        return None, None

    if a > b:
        a, b = b, a

    mid = (a+b)/2
    iteration = 0
    error = [(b-a)/2]

    while (abs(f(mid)) > tolerance and b - a > tolerance):
        mid = (a + b)/2

        if f(mid) == 0:
```

```
            return mid, None
        elif f(mid)*f(a) > 0:
            a = mid
        else:
            b = mid

        iteration += 1
        error.append((b-a)/2)
        ratio = error[iteration]/error[iteration-1]
        print(f"x_{iteration}: {mid:.6f} ,error: {error[iteration]:.6f} ,e_{iteration}/e_{iteration-1}: {ratio:.6f}")

    return (a+b)/2 ,(b-a)/2

def f(x):
    return x**3

res, err = bisection(f ,-100.565664, 333.5456,1e-10)
print(f"root find:{res}, with error: {err}")
```

2.1.2 Theoretical Foundations

Theorem 2.1.
Suppose $f \in C[a,b]$ and $f(a)f(b) < 0$. The bisection method generates a sequence $\{P_n\}_{n=1}^{\infty}$ for approximating a zero P^* of $f(P^*) = 0$ with:

$$|P_n - P^*| \leq \frac{b-a}{2^n}, \quad \forall n \geq 1$$

Proof.
For any $n \geq 1$, there exists subintervals $[a_n, b_n]$ of $[a,b]$ such that $b_n - a_n = \frac{1}{2^n}(b-a)$ and $P^* \in (a_n, b_n)$.

$$P_n = \frac{1}{2}(a_n + b_n)$$

$$\therefore |P_n - P^*| \leq \frac{1}{2}(b_n - a_n) = \frac{1}{2} \cdot \frac{1}{2^{n-1}}(b-a) = \frac{1}{2^n}(b-a)$$

Hence $\{P_n\}$ converges to P^* with rate $\mathcal{O}\left(\frac{1}{2^n}\right)$. ■

Note: For a function f with accuracy ϵ in interval $[a,b]$, how many steps are required?

$$|P_n - P^*| \leq \frac{1}{2^n}(b-a) < \epsilon$$

Where n is the minimum steps required.

2.2 Fixed-Point Iteration

2.2.1 Definition & Properties

Definition 2.1.
A number X^* is called a **fixed point** of a function g if and only if $X^* = g(X^*)$.

Theorem 2.2 (Uniqueness and existence of fixed point)**.**

1. If $g \in C[a,b]$ and $g(x) \in [a,b]$ for all $x \in [a,b]$, then $\exists X^* \in [a,b]$.
2. If $g'(x)$ exists on (a,b) and $|g'(x)| \leq k < 1, \forall x \in (a,b)$, then $\exists! X^* \in [a,b]$.

Explanation:

1. Imagine there is a square with the range $x \in [a,b]$, $y \in [a,b]$. Then, $g(x)$ is a continuous curve extending from $x = a$ to $x = b$. Because the value of $g(x)$ is always between $[a,b]$, this line must cross the diagonal $y = x$ at least once. That intersection point is the fixed point X^*.
2. $|g'(x)|$ is the slope of the function. The restriction $|g'(x)| \leq k < 1$ means that the function's change is relatively "gradual," even flatter than the diagonal $y = x$. If you cross the diagonal once and want to cross it again, your slope must become steeper than 1. Since the slope is restricted to below 1, once the function crosses the diagonal, it can never return.

Proof.
Existence:

Case 1: $g(a) = a$ or $g(b) = b$.

Then $g(x)$ trivially has a fixed point.

Case 2: $g(a) > a$ and $g(b) < b$. Let $h(x) := g(x) - x$, which is continuous on $[a,b]$ and $h(a) > 0$, $h(b) < 0$. By the Intermediate Value Theorem, $\exists X^* \in (a,b)$ such that $h(X^*) = 0$, i.e. $g(X^*) = X^*$. Hence, the fixed point exists.

Uniqueness:

Suppose $\exists\ y^*, y^* \neq X^*$ such that $g(y^*) = y^*$. By the Mean Value Theorem, $\exists\ \xi \in (X^*, y^*)$ such that:

$$g'(\xi) = \frac{g(X^*) - g(y^*)}{X^* - y^*}$$

$$\begin{aligned} \therefore |g(X^*) - g(y^*)| &= |g'(\xi)||X^* - y^*| \\ \Rightarrow |X^* - y^*| &= |g'(\xi)||X^* - y^*| \\ \Rightarrow |g'(\xi)| &= 1 \quad (\Rightarrow\Leftarrow) \end{aligned}$$

Contradiction! Hence, $X^* = y^*$, which means the fixed point is unique. ■

Extension: Why can't $g'(x)$ be less than -1?
One might think that even if $g'(x) < -1$, the fixed point is still unique. **THIS IS FALSE**, because it would never satisfy the self-mapping condition $g(x) \in [a, b]$. By the Mean Value Theorem, $\exists\, c \in (a, b)$ such that:

$$\begin{aligned} g(b) - g(a) = g'(c)(b-a), \quad & \text{where } c \in (a,b) \text{ and } g'(x) < -1 \\ \Rightarrow\ & g(b) - g(a) < -1 \cdot (b-a) \\ \Rightarrow\ & g(a) - g(b) > b - a \\ \Rightarrow\ & |g(a) - g(b)| > |a - b| \\ \therefore\ & g([a,b]) \not\subset [a,b] \end{aligned}$$

2.2.2 Algorithms: Find Fixed Point

```python
def fixed_point(g, x_0, tolerance = 1e-10, max_iteration = 100):
    x = [x_0]
    for k in range(1, max_iteration+1):
        x.append(g(x[k-1]))
        if abs(x[k] - x[k-1]) < tolerance:
            return x[k]
    return x[-1]
```

2.2.3 Convergence Theorem & Error Analysis

Theorem 2.3.
If $f \in C[a, b], g(x) \in [a, b], \forall x \in (a, b)$, and $|g'(x)| \leq k < 1$ for some $k \in (0, 1)$, then for any $x_0 \in (a, b)$, the sequence $\{x_n\}_{n=1}^{\infty}$ generated by $x_n = g(x_{n-1}), n \geq 1$, converges to the unique fixed point in $[a, b]$.

Proof.
For $n > 0$, and X^* as the fixed point:

$$\begin{aligned} |x_n - X^*| &= |g(x_{n-1}) - g(X^*)| \quad \text{(by MVT)} \\ &= |g'(\xi)(x_{n-1} - X^*)|, \quad \xi \in (x_{n-1}, X^*) \\ &\leq k|x_{n-1} - X^*| \leq k^2|x_{n-2} - X^*| \leq \cdots \leq k^n|x_0 - X^*| \end{aligned}$$

Since $k \in (0, 1)$, $\lim_{n\to\infty} k^n = 0 \Rightarrow \lim_{n\to\infty} |x_n - X^*| = 0$.
Hence, $\{x_n\}$ converges to X^*. ■

Corollary 2.3.1 (The Error Bound)**.**

$$|x_n - X^*| \le k^n \max(x_0 - a, b - x_0) \quad \text{and} \quad |x_n - X^*| \le \frac{k^n}{1-k}|x_1 - x_0|$$

Theorem 2.4 (Convergent Rate for Fixed Point Iteration)**.**
Let $g \in C[a,b]$, $g(x) \in [a,b]$, $\forall\ x \in [a,b]$. If $|g'(x)| \le k < 1$ and $g'(x) \neq 0$, then for any $x_0 \in [a,b]$, the fixed point sequence $\{x_n\}_{n=1}^{\infty}$ with $x_n = g(x_{n-1})$ will converge linearly to the unique X^*.

Proof.

$$\because x_{n+1} - X^* = g(x_n) - g(X^*) = g'(\xi)(x_n - X^*) \quad \text{(MVT)}$$
Since $g'(x)$ is continuous on (a,b), $\lim\limits_{n\to\infty} g'(\xi) = g'(X^*)$
Thus, $\lim\limits_{n\to\infty} \dfrac{|x_{n+1} - X^*|}{|x_n - X^*|} = \lim\limits_{n\to\infty} |g'(\xi)| = |g'(X^*)| \le k$
$\therefore \alpha = 1,\ \lambda = g'(X^*)$
Hence, $\{x_n\}$ converges linearly to X^*.

■

2.3 Newton's Method

If $f \in C^2[a,b], x_0 \in (a,b)$, the solution $X^* \in (a,b)$ and $f(X^*) = 0$. Assume x_0 is close to X^*. By Taylor's expansion:

$$f(X^*) = f(x_0) + f'(x_0)(X^* - x_0) + \frac{f''(\xi)}{2!}(X^* - x_0)^2, \quad \xi \in (x_0, X^*)$$

$\because |x_0 - X^*|$ is small, $\therefore |x_0 - X^*|^2 \to 0$ and
$\because f \in C^2[a,b]$, $\therefore |f''(\xi)|$ is bounded.

$$\begin{aligned} 0 &\approx f(x_0) + f'(x_0)(X^* - x_0) \\ \Rightarrow X^* &\approx x_0 - \frac{f(x_0)}{f'(x_0)} \\ \Rightarrow x_1 &= x_0 - \frac{f(x_0)}{f'(x_0)} \\ \Rightarrow x_n &= x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}, \quad n = 1, 2, \dots \end{aligned}$$

2.3.1 Algorithms

```python
import math

def d(f, x):
    h = 1e-6
    return (f(x + h) - f(x - h)) / (2 * h)

def f(x):
    return math.log(x) - 1

def newton_method(f, a, tolerance):
    res = [a]
    res.append(res[0] - f(res[0]) / d(f, res[0]))
    iteration = 1

    while (abs(res[iteration] - res[iteration-1]) >= tolerance or
        abs(f(res[iteration])) >= tolerance):
        current_x = res[iteration]
        next_x = current_x - f(current_x) / d(f, current_x)
        res.append(next_x)
        iteration += 1
        print(f"Iteration␣{iteration}:␣x␣=␣{res[iteration]:.6f},␣
           f(x)␣=␣{f(res[iteration]):.6e}")

    return res[iteration]

final_root = newton_method(f, 4, 1e-6)
print(f"Found␣root:␣{final_root:.6f}")
```

2.3.2 Convergence of Newton's Method

Theorem 2.5.
If $f \in C^2[a,b], \exists\ X^* \in [a,b]$ such that $f(X^*) = 0$ and $f'(X^*) \neq 0$, then $\exists \delta > 0$ such that for any $x_0 \in (X^* - \delta, X^* + \delta)$, the sequence $\{x_n\}_{n=1}^{\infty}$ converges.

Proof.
Let $g(x) := x - \frac{f(x)}{f'(x)}$.
Since f' is continuous and $f'(X^*) \neq 0$, $\exists \bar{\delta} > 0$ such that $f'(x) \neq 0, \forall x \in (X^* - \bar{\delta}, X^* + \bar{\delta})$.
Hence, $g(x)$ is continuous on $(X^* - \bar{\delta}, X^* + \bar{\delta})$ and:

$$g'(x) = 1 - \frac{f'(x)f'(x) - f(x)f''(x)}{[f'(x)]^2} = \frac{f(x)f''(x)}{[f'(x)]^2} \quad \text{and} \quad g'(X^*) = \frac{f(X^*)f''(X^*)}{[f'(X^*)]^2} = 0$$

$\because g'(x)$ is continuous around X^* and $\exists\ k \in (0,1)$ s.t.

$|g'(x)| \le k < 1, \quad \forall x \in [X^* - \delta_1, X^* + \delta_1],$ for some $\delta_1 > 0$.

Let $\delta = \min(\delta_1, \bar{\delta}),$ then $|g(x) - g(X^*)| = |g'(\xi)||X^* - x|$ (MVT for $\xi \in (X^*, x)$)

$\therefore |g(x) - X^*| = |g(x) - g(X^*)| \le k|x - X^*| < |x - X^*|$

Hence, the function $g(x)$ maps $[X^* - \delta_1, X^* + \delta_1]$ onto $[X^* - \delta, X^* + \delta]$. By the fixed point theorem, the sequence $\{x_n\}_{n=1}^{\infty}$ generated by $x_n = g(x_{n-1}), n \ge 1$ will converge to X^* for any initial value $x_0 \in (X^* - \delta, X^* + \delta)$. ■

2.3.3 Error Analysis

Theorem 2.6 (Convergent Rate of Newton's Method)**.**
Newton's method converges quadratically and

$$\lim_{n\to\infty} \frac{x_{n+1} - X^*}{x_n - X^*} < \frac{M}{2} \quad \text{where } g''(x) < M.$$

Proof.
Choose $k \in (0,1)$, by Taylor expansion for $x \in (X^* - \delta, X^* + \delta)$

$$g(x) = g(X^*) + g'(X^*)(x - X^*) + \frac{g''(\xi)}{2}(x - X^*)^2, \quad \text{where } \xi \in (x, X^*)$$

Since, $g(X^*) = X^*$ and $g'(X^*) = 0$, then we have

$$g(x) = X^* + \frac{1}{2}g''(\xi)(x_n - X^*)^2$$

In particularly, $x = x_n$

$$x_{n+1} - X^* = \frac{1}{2}g''(\xi)(x_n - X^*)^2 |x_{n+1} - X^*| \le \frac{1}{2}M|x_n - X^*|^2, \quad \text{where } M = \max(g''(\xi))$$

Hence, we have

$$\lim_{n\to\infty} \frac{x_{n+1} - X^*}{(x_n - X^*)^2} \le \frac{M}{2}$$

■

3 Interpolation and Polynomial approximation

Theorem 3.1 (Weierstrass Approximation Theorem)**.**
Suppose $f \in C[a,b]$ for each $\epsilon > 0$, $\exists P(x)$: polynomial s.t. $|f(x) - P(x)| < \epsilon, \forall x \in [a,b]$

3.1 Lagrange Polynomial

In numerical analysis, when we obtain a set of discrete data points, we often need to construct a polynomial that passes precisely through these points. This process is called **Polynomial Interpolation**. The Lagrange polynomial provides an intuitive method to construct such a function.

Initial Condition:
Suppose we have $n+1$ distinct data points on a function $f(x)$: $x_0, x_1, \ldots, x_n$.
Our goal is to find a polynomial $P(x)$ of degree at most n that satisfies the interpolation condition for all points: $P(x_i) = f(x_i)$.

Standard Derivation

First, we construct the **Lagrange Basis Polynomials** $L_{n,i}(x)$. The key property of this basis function is that it equals 1 at its corresponding node x_i, and 0 at all other nodes:

$$L_{n,i}(x_j) = \begin{cases} 1 & , \text{ if } i = j \\ 0 & , \text{ if } i \neq j \end{cases}$$

By multiplying the terms $(x - x_k)$ for all other nodes and normalizing the result, we can write the general formula for $L_{n,i}(x)$:

$$L_{n,i}(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_0)(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \prod_{\substack{k=0 \\ k\neq i}}^{n} \frac{x-x_k}{x_i-x_k}$$

With the basis functions defined, we simply multiply each basis by its corresponding function value $f(x_i)$ and sum them up to obtain the final interpolation polynomial $P(x)$:

$$P(x) = \sum_{i=0}^{n} L_{n,i}(x) \cdot f(x_i) = \sum_{i=0}^{n} \left(\frac{\prod_{j\neq i}^{n}(x-x_j)}{\prod_{j\neq i}^{n}(x_i-x_j)} \right) \cdot f(x_i)$$

Barycentric Form

The traditional Lagrange polynomial is computationally expensive to evaluate. To resolve this, we introduce an auxiliary **Nodal Polynomial** $Q(x)$ to unify all the numerator terms:

$$\text{Let } Q(x) = \prod_{j=0}^{n}(x-x_j)$$

To find the relationship between the denominator $\prod_{j\neq i}^{n}(x_i - x_j)$ and $Q(x)$, we apply **Logarithmic Differentiation**. This technique converts the product into a sum, making

differentiation much easier:

$$\begin{aligned}
\ln Q(x) &= \sum_{j=0}^{n} \ln(x - x_j) \\
\frac{Q'(x)}{Q(x)} &= \sum_{j=0}^{n} \frac{1}{x - x_j} \quad \text{(Differentiating both sides)} \\
Q'(x) &= \sum_{j=0}^{n} \frac{Q(x)}{x - x_j}
\end{aligned}$$

Next, we evaluate $Q'(x)$ precisely at $x = x_i$. Since $Q(x_i) = 0$, all terms in the summation evaluate to 0 except for the term where the denominator is $(x_i - x_i)$, which is preserved by evaluating the limit (L'Hôpital's rule):

$$Q'(x_i) = \lim_{x \to x_i} \sum_{j=0}^{n} \frac{Q(x)}{x - x_j} = \prod_{\substack{j=0 \\ j \neq i}}^{n} (x_i - x_j)$$

This result is exactly the denominator in our original $L_{n,i}(x)$ formula! Therefore, we can redefine a much more computationally efficient basis function $\phi_{n,i}(x)$:

$$\phi_{n,i}(x) = \frac{\frac{Q(x)}{x - x_i}}{Q'(x_i)} = \frac{Q(x)}{(x - x_i)Q'(x_i)}$$

Finally, the Lagrange polynomial can be elegantly rewritten into the **First Form of Barycentric Interpolation**:

$$P(x) = \sum_{i=0}^{n} \frac{Q(x)}{(x - x_i)Q'(x_i)} \cdot f(x_i)$$

This form drastically reduces computational complexity. If a new data point is added, we only need to update the weights rather than recalculating all polynomials from scratch.

Theorem 3.2 (Uniqueness and Existence of Lagrange Interpolating Polynomial)**.**
If $x_0, \ldots, x_n$ are distinct numbers, and $f(x)$ is a function whose values are given at these numbers, then there exists a unique polynomial $P(x)$ of degree at most n such that $P(x_k) = f(x_k)$ for all $k = 0, \ldots, n$. This polynomial is given by:

$$P(x) = \sum_{k=0}^{n} f(x_k) L_{n,k}(x)$$

3.2 Error Analysis of Lagrange Interpolation

Theorem 3.3 (Error of Lagrange Interpolating Polynomial)**.**
Suppose $x_0, \ldots, x_n$ are distinct numbers in $[a, b]$ and $P(x)$ is the Lagrange interpolating

polynomial of $x_0, \ldots, x_n$. If $f \in C^{n+1}[a,b]$, then for each $x \in [a,b]$, there exists a number $\xi(x) \in (a,b)$ such that:

$$f(x) = P(x) + \frac{f^{(n+1)}(\xi(x))}{(n+1)!}(x-x_0)(x-x_1)\cdots(x-x_n)$$

Proof.

Case 1: $x = x_k$

If $x = x_k$ for any $k = 0, \ldots, n$, then $f(x_k) = P(x_k)$, and $\xi(x)$ can be arbitrarily chosen in (a,b).

Case 2: $x \neq x_k$

If $x \neq x_k$ for all $k = 0, \ldots, n$, we define a function g for $t \in [a,b]$:

$$g(t) = f(t) - P(t) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{t - x_i}{x - x_i}$$

Let $t = x$, we have:

$$g(x) = f(x) - P(x) - [f(x) - P(x)] \prod_{i=0}^{n} \frac{x - x_i}{x - x_i} = f(x) - P(x) - [f(x) - P(x)] = 0$$

Thus $g \in C^{n+1}[a,b]$, and $g(t) = 0$ for all $t \in \{x, x_0, \ldots, x_n\}$. By Rolle's Theorem, there exists $\xi \in (a,b)$ such that $g^{(n+1)}(\xi) = 0$, i.e.,

$$\begin{aligned}
0 = g^{(n+1)}(\xi) &= f^{(n+1)}(\xi) - P^{(n+1)}(\xi) - [f(x) - P(x)] \frac{d^{n+1}}{dt^{n+1}} \left[\prod_{i=0}^{n} \frac{t - x_i}{x - x_i} \right]_{t=\xi} \\
&= f^{(n+1)}(\xi) - 0 - [f(x) - P(x)] \frac{d^{n+1}}{dt^{n+1}} \left[\frac{t^{n+1}}{\prod_{i=0}^{n}(x - x_i)} \right]_{t=\xi} \\
&= f^{(n+1)}(\xi) - [f(x) - P(x)] \frac{(n+1)!}{\prod_{i=0}^{n}(x - x_i)} \\
&\Rightarrow f(x) = P(x) + \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{i=0}^{n}(x - x_i)
\end{aligned}$$

■

3.3 Newton's Divided Difference

Suppose $P_n(x)$ is the n-th Lagrange interpolating polynomial with $f(x)$ at distinct numbers $x_0, \ldots, x_n$. Although the polynomial is unique, we can alternate its representation. Let

$$P_n(x) = a_0 + a_1(x - x_0) + \cdots + a_n(x - x_0)(x - x_1)\cdots(x - x_{n-1})$$

Then, $a_0 = P(x_0) = f(x_0)$, and $a_1 = \frac{f(x_1)-f(x_0)}{x_1-x_0}$.

To evaluate a_k for $k = 0, \ldots, n$, we define the divided difference:

1. $f[x_i] = f(x_i)$
2. $f[x_i, x_{i+1}] = \frac{f[x_{i+1}]-f[x_i]}{x_{i+1}-x_i}$
3. $f[x_1, \ldots, x_n] = \frac{f[x_2,\ldots,x_n]-f[x_1,\ldots,x_{n-1}]}{x_n-x_1}$

Then we can write $f[x_0] = a_0$, $f[x_0, x_1] = a_1$. As might be expected from evaluating a_0 and a_1, $a_k = f[x_0, \ldots, x_k]$. Thus, $P_n(x)$ can be rewritten in a form called Newton's Divided Differences:

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, \ldots, x_k](x - x_0)(x - x_1)\cdots(x - x_{k-1})$$

Theorem 3.4.
Suppose that $f \in C^n[a, b]$ and $x_0, \ldots, x_n$ are distinct numbers in $[a, b]$, then there exists $\xi \in (a, b)$ such that:

$$f[x_0, \ldots, x_n] = \frac{f^{(n)}(\xi)}{n!}$$

Proof.
Let $g(x) = f(x) - P_n(x)$. Since $f(x_i) = P_n(x_i)$ for $i = 0, \ldots, n$, g has $n + 1$ distinct zeros in $[a, b]$. By Rolle's Theorem, there exists $\xi \in (a, b)$ such that $g^{(n)}(\xi) = 0$.

$$\Rightarrow f^{(n)}(\xi) - P_n^{(n)}(\xi) = 0 \Rightarrow f^{(n)}(\xi) = P_n^{(n)}(\xi)$$

Since $P_n(x) = f[x_0, \ldots, x_n]x^n + \ldots$, it follows that $P_n^{(n)}(x) = n!f[x_0, \ldots, x_n]$. Hence, $f^{(n)}(\xi) = n!f[x_0, \ldots, x_n]$. ■

If the nodes are arranged uniformly, Newton's Divided Difference formula can be expressed in a simplified form. Assume $x_i = x_0 + ih$ and let $x = x_0 + sh$, the difference becomes $x - x_i = (s - i)h$. So,

$$\begin{aligned}
P_n(x) &= P_n(x_0 + sh) \\
&= f[x_0] + shf[x_0, x_1] + sh(s-1)hf[x_0, x_1, x_2] + \cdots + s(s-1)\cdots(s-k+1)h^k f[x_0, \ldots, x_k] + \ldots \\
&= f(x_0) + \sum_{k=1}^{n} \binom{s}{k} k!h^k f[x_0, \ldots, x_k] \\
&= f(x_0) + \sum_{k=1}^{n} \binom{s}{k} \Delta^k f(x_0)
\end{aligned}$$

3.4 Cubic Spline Interpolation

Definition 3.1.
If a function f is defined on $[a, b]$ and a set of nodes $a = x_0 < x_1 < \cdots < x_n = b$, a cubic spline interpolant S for f satisfies:

1. $S(x)$ is a cubic polynomial, denoted $S_j(x)$ on $[x_j, x_{j+1}]$ for each $j = 0, 1, \ldots, n-1$.
2. $S_j(x_j) = f(x_j)$ and $S_j(x_{j+1}) = f(x_{j+1})$ for $j = 0, 1, \ldots, n-1$.
3. $S_{j+1}(x_{j+1}) = S_j(x_{j+1})$ for $j = 0, \ldots, n-2$.
4. $S'_{j+1}(x_{j+1}) = S'_j(x_{j+1})$ for $j = 0, \ldots, n-2$.
5. $S''_{j+1}(x_{j+1}) = S''_j(x_{j+1})$ for $j = 0, \ldots, n-2$.
6. One of the following sets of boundary conditions is satisfied:
   - $S''(x_0) = S''(x_n) = 0$ (Natural Splines)
   - $S'(x_0) = f'(x_0)$ and $S'(x_n) = f'(x_n)$

To construct the cubic spline interpolant for a function f:

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

Since $a_j = S_j(x_j)$, and by definition condition (3):

$$a_{j+1} = S_{j+1}(x_{j+1}) = S_j(x_{j+1}) = a_j + b_j(x_{j+1} - x_j) + c_j(x_{j+1} - x_j)^2 + d_j(x_{j+1} - x_j)^3$$

Let $h_j = x_{j+1} - x_j$ for $j = 0, \ldots, n-2$, then:

$$a_{j+1} = a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 \quad \text{for } j = 0, \ldots, n-1 \tag{1}$$

Similarly, from $b_j = S'_j(x_j)$, we have:

$$b_{j+1} = b_j + 2c_j h_j + 3d_j h_j^2 \quad \text{for } j = 0, \ldots, n-1 \tag{2}$$

Similarly, from $2c_j = S''_j(x_j)$, we have $2c_{j+1} = 2c_j + 6d_j h_j$, which yields:

$$d_j = \frac{c_{j+1} - c_j}{3h_j} \tag{3}$$

Substitute (3) into (1) and (2):

$$a_{j+1} = a_j + b_j h_j + \frac{h_j^2}{3}(2c_j + c_{j+1})$$

$$b_{j+1} = b_j + h_j(c_j + c_{j+1}) \tag{4}$$

Isolating b_j and b_{j-1}:

$$b_j = \frac{1}{h_j}(a_{j+1} - a_j) - \frac{h_j}{3}(2c_j + c_{j+1})$$
$$b_{j-1} = \frac{1}{h_{j-1}}(a_j - a_{j-1}) - \frac{h_{j-1}}{3}(2c_{j-1} + c_j)$$

Substitute b_j and b_{j-1} (as b_{j+1} and b_j) into (4), we get:

$$h_{j-1}c_{j-1} + 2(h_{j-1} + h_j)c_j + h_j c_{j+1} = \frac{3}{h_j}(a_{j+1} - a_j) - \frac{3}{h_{j-1}}(a_j - a_{j-1})$$

Where $\{c_j\}_{j=0}^n$ are unknowns, $\{h_j\}_{j=0}^{n-1}$ and $\{a_j\}_{j=0}^n$ are given, and $\{b_j\}_{j=0}^{n-1}$ and $\{d_j\}_{j=0}^{n-1}$ depend on c_j. Thus, once $\{c_j\}_{j=0}^n$ are determined, we can construct $\{S_j(x)\}_{j=0}^{n-1}$.

To find $\{c_j\}_{j=0}^n$ using the definition $S''(x_0) = S''(x_n) = 0$, i.e., $c_0 = c_n = 0$. From the equation above, we can construct a linear system $Ax = b$:

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ h_0 & 2(h_0 + h_1) & h_1 & \cdots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & & h_{n-2} & 2(h_{n-2} + h_{n-1}) & h_{n-1} \\ 0 & \cdots & 0 & 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)}$$

$$x = \begin{bmatrix} c_0 \\ \vdots \\ c_n \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ \frac{3}{h_1}(a_2 - a_1) - \frac{3}{h_0}(a_1 - a_0) \\ \vdots \\ \frac{3}{h_{n-1}}(a_n - a_{n-1}) - \frac{3}{h_{n-2}}(a_{n-1} - a_{n-2}) \\ 0 \end{bmatrix}$$

Since A is strictly diagonally dominant ($|a_{ii}| > \sum_{j\neq i}^{n+1} |a_{ij}|$), the system has a unique solution.

Theorem 3.5 (Natural Splines)**.**
If f is defined at $a = x_0 < x_1 < \cdots < x_n = b$, then f has a unique natural spline interpolant S on the nodes $x_0 \ldots x_n$. S satisfies $S''(x_0) = S''(x_n) = 0$.

Proof.
$\because S''(x_0) = S''(x_n) = 0 \therefore c_n = S''(x_n)/2 = 0$ $S''(x_0) = 2c_0 + 6d_0(x_0 - x_0) = 0 \Rightarrow c_0 = 0$
By deriving $c_0 = c_n$, we can construct a linear system: $Ax = b$, where A, x, b is the same as above. Since A is strictly diagonally dominant, the system has a unique solution. ■

4 Numerical Differentiation and Integration

4.1 Numerical Differentiation

To evaluate $f'(x)$ at $x = x_0$, we can simply compute:

$$f'(x_0) \approx \frac{f(x_0 + h) - f(x_0)}{h}$$

for a small value h. But it is not always success due to the round error in computer. Hence, we construct the first Lagrange polynomial $P_{0,1}(x)$ for f determined by x_0, x_1.

$$f(x) = P_{0,1}(x) + \frac{(x - x_0)(x - x_1)}{2!} f''(\xi(x))$$

$$= f(x_0)\frac{x - x_1}{x_0 - x_1} + f(x_1)\frac{x - x_0}{x_1 - x_0} + \frac{(x - x_0)(x - x_1)}{2!} f''(\xi(x))$$

Since $x_1 = x_0 + h$, for some $\xi(x) \in (x_0, x_1)$:

$$\begin{aligned} f'(x) &= \frac{f(x_0 + h) - f(x_0)}{h} + D_x \left[\frac{(x - x_0)(x - x_1)}{2!} f''(\xi(x)) \right] \\ &= \frac{f(x_0 + h) - f(x_0)}{h} + \frac{2(x - x_0) - h}{2} f''(\xi(x)) + \frac{(x - x_0)(x - x_1)}{2!} D_x[f''(\xi(x))] \end{aligned}$$

Take $x = x_0$:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{h}{2} f''(\xi(x_0)) \tag{5}$$

Suppose $M = \max(|f''(x)|)$ for $x \in (x_0, x_1)$, then we have Error $E \triangleq \left|\frac{h}{2} f''(\xi(x))\right| \leq \frac{h}{2} M$. This formula is known as forward-difference formula if $h > 0$, backward-difference formula if $h < 0$.

To obtain general derivates approximation formula, suppose $\{x_i\}_{i=0}^n$, $x_i \neq x_j \ \forall i \neq j$ in some interval I and $f \in C^{n+1}(I)$. By Lagrange polynomial interpolation:

$$f(x) = \sum_{k=0}^{n} f(x_k) L_k(x) + \frac{\prod_{j=0}^{n}(x - x_j)}{(n+1)!} f^{(n+1)}(\xi(x))$$

for some $\xi(x) \in I$. $L_k(x)$ is the k-th Lagrange polynomial. Differentiating this expression:

$$f'(x) = \sum_{k=0}^{n} f(x_k) L_k'(x) + D_x \left[\frac{\prod_{j=0}^{n}(x - x_j)}{(n+1)!} \right] f^{(n+1)}(\xi(x)) + \frac{\prod_{j=0}^{n}(x - x_j)}{(n+1)!} D_x[f^{(n+1)}(\xi(x))]$$

Let $x = x_i$, $i = 0 \ldots n$:

$$f'(x_i) = \sum_{k=0}^{n} f(x_k) L_k'(x_i) + \frac{f^{(n+1)}(\xi(x_i))}{(n+1)!} \prod_{\substack{j=0 \\ j \neq i}}^{n} (x_i - x_j)$$

This formula is called $(n+1)$-point formula. Usually, we use $f'(x_i) \approx \sum_{k=0}^{n} f(x_k)L'_k(x_i)$.

Three-Points Formulas

Suppose $x_1 = x_0 + h$, $x_2 = x_0 + 2h$, $h \neq 0$:

$$\begin{aligned}
L_0(x) &= \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \quad L'_0(x) = \frac{1}{2h^2}(2x - x_1 - x_2) \\
L_1(x) &= \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}, \quad L'_1(x) = -\frac{1}{h^2}(2x - x_0 - x_2) \\
L_2(x) &= \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}, \quad L'_2(x) = \frac{1}{2h^2}(2x - x_0 - x_1)
\end{aligned}$$

$$\begin{aligned}
f'(x_j) = f(x_0)\left[\frac{1}{2h^2}(2x_j - x_1 - x_2)\right] + f(x_1)\left[-\frac{1}{h^2}(2x_j - x_0 - x_2)\right] \\
+ f(x_2)\left[\frac{1}{2h^2}(2x_j - x_0 - x_1)\right] + \frac{f^{(3)}(\xi_j)}{3!}\prod_{\substack{k=0 \\ k\neq j}}^{2}(x_j - x_k)
\end{aligned}$$

Let $x_j = x_0$:

$$f'(x_0) = \frac{1}{h}\left[-\frac{3}{2}f(x_0) + 2f(x_1) - \frac{1}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_0)$$

Let $x_j = x_1$:

$$f'(x_1) = \frac{1}{h}\left[-\frac{1}{2}f(x_0) + \frac{1}{2}f(x_2)\right] - \frac{h^2}{6}f^{(3)}(\xi_1)$$

Let $x_j = x_2$:

$$f'(x_2) = \frac{1}{h}\left[\frac{1}{2}f(x_0) - 2f(x_1) + \frac{3}{2}f(x_2)\right] + \frac{h^2}{3}f^{(3)}(\xi_2)$$

Substitute $x_1 = x_0 + h$, $x_2 = x_0 + 2h$ can get another expression.

Five-Points Formulas

$$f'(x_0) = \frac{1}{12h}[-25f(x_0)+48f(x_0+h)-36f(x_0+2h)+16f(x_0+3h)-3f(x_0+4h)]+\frac{h^4}{5}f^{(5)}(\xi), \quad \xi \in (x_0, x_0+$$

Second Derivative Midpoint Formula

Consider the approximation for $f''(x_0)$ and $f \in C^4[x_0-h, x_0+h]$. The most direct method is to use **Taylor expansions** on $f(x_0 + h)$ and $f(x_0 - h)$:

$$f(x_0 + h) = f(x_0) + f'(x_0)h + \frac{h^2}{2!}f''(x_0) + \frac{h^3}{3!}f'''(x_0) + \frac{h^4}{4!}f^{(4)}(\xi_1) \tag{6}$$

$$f(x_0 - h) = f(x_0) - f'(x_0)h + \frac{h^2}{2!}f''(x_0) - \frac{h^3}{3!}f'''(x_0) + \frac{h^4}{4!}f^{(4)}(\xi_2) \tag{7}$$

By adding (6) and (7), we can cancel out the terms with odd powers of h, then we get:

$$f(x_0+h)+f(x_0-h)=2f(x_0)+h^2f''(x_0)+\frac{h^4}{24}[f^{(4)}(\xi_1)+f^{(4)}(\xi_2)]$$

where $\xi_1 \in (x_0, x_0+h)$, $\xi_2 \in (x_0-h, x_0)$. Since $f \in C^4[x_0-h, x_0+h]$ and $f^{(4)}$ is bounded on the interval. By IVT, $\exists \xi \in [x_0-h, x_0+h]$ s.t. $\frac{1}{2}(f^{(4)}(\xi_1)+f^{(4)}(\xi_2)) = f^{(4)}(\xi)$. Substitute it back, we have:

$$f(x_0+h)+f(x_0-h)=2f(x_0)+h^2f''(x_0)+\frac{h^4}{12}f^{(4)}(\xi)$$

$$\Rightarrow f''(x_0)=\frac{1}{h^2}[f(x_0-h)-2f(x_0)+f(x_0+h)]-\frac{h^2}{12}f^{(4)}(\xi)$$

Round-Off Error Instability

Particularly, it is necessary to pay attention to round-off error. Suppose our computation actually use $\tilde{f}(x_0+h)$ and $\tilde{f}(x_0-h)$ with round-off error $e(x_0+h)$ and $e(x_0-h)$ respectively. i.e.

$$\begin{cases} f(x_0+h)=\tilde{f}(x_0+h)+e(x_0+h) \\ f(x_0-h)=\tilde{f}(x_0-h)+e(x_0-h) \end{cases}$$

If we use these terms to approximate $f'(x_0)$, the total error

$$E=f'(x_0)-\frac{\tilde{f}(x_0+h)-\tilde{f}(x_0-h)}{2h}=\frac{1}{2h}[e(x_0+h)-e(x_0-h)]-\frac{h^2}{6}f^{(3)}(\xi_1)$$

Suppose $|e(x_0 \pm h)| \le \varepsilon$ and $|f^{(3)}(\xi_1)| \le M$, then

$$|E| \le \frac{\varepsilon}{h}+\frac{h^2}{6}M$$

If ε is given, E is a function only regarding to h. We can find the best h to minimize the error:

$$E'(h)=-\frac{\varepsilon}{h^2}+\frac{h}{3}M=0 \implies h=\sqrt[3]{\frac{3\varepsilon}{M}}$$

4.2 Numerical Integration

The Trapezoidal Rule

To derive the Trapezoidal Rule for approximating $\int_a^b f(x)dx$, let $a=x_0$, $b=x_1$, $h=b-a$, and we use Lagrange polynomial.

$$P_1(x)=\frac{(x-x_1)}{(x_0-x_1)}f(x_0)+\frac{(x-x_0)}{(x_1-x_0)}f(x_1)$$

then,

$$\int_a^b f(x)dx = \int_{x_0}^{x_1} P_1(x) + E(x)dx$$

$$\begin{aligned}\int_{x_0}^{x_1} P_1(x)dx &= \int_{x_0}^{x_1} \left[f(x_0)\frac{x-x_1}{x_0-x_1} + f(x_1)\frac{x-x_0}{x_1-x_0} \right] dx \\ &= \left[f(x_0)\frac{(x-x_1)^2}{2(x_0-x_1)} + f(x_1)\frac{(x-x_0)^2}{2(x_1-x_0)} \right]_{x_0}^{x_1} \\ &= \frac{(x_1-x_0)}{2}[f(x_0)+f(x_1)]\end{aligned}$$

$$\int_{x_0}^{x_1} E(x)dx = \frac{1}{2}\int_{x_0}^{x_1} f''(\xi(x))(x-x_0)(x-x_1)dx$$

$\because$ $f \in C^2[a,b]$. By Extreme Value Theorem, $m < f''(x) < M$ $\forall x \in [x_0, x_1]$. Let $g(x) = (x-x_0)(x-x_1)$, $E(x) = \frac{1}{2}f''(x)g(x)$ i.e. $m \cdot g(x) \leq E(x) \leq M \cdot g(x) \Rightarrow \int_{x_0}^{x_1} m\, g(x)dx \leq \int_{x_0}^{x_1} E(x)dx \leq \int_{x_0}^{x_1} M\, g(x)dx$. Due to m and M are constants, we have

$$m\int_{x_0}^{x_1} g(x)dx \leq \int_{x_0}^{x_1} E(x)dx \leq M\int_{x_0}^{x_1} g(x)dx$$

$$\Rightarrow m \leq \frac{\int_{x_0}^{x_1} E(x)dx}{\int_{x_0}^{x_1} g(x)dx} \leq M$$

By IVT, $\exists \xi \in (x_0, x_1)$ s.t.

$$\int_{x_0}^{x_1} E(x)dx = \frac{1}{2}f''(\xi)\int_{x_0}^{x_1}(x-x_0)(x-x_1)dx = -\frac{h^3}{12}f''(\xi)$$

Thus, substitute $P_1(x)$ and $E(x)$:

$$\int_{x_0}^{x_1} f(x)dx = \frac{h}{2}[f(x_0)+f(x_1)] - \frac{h^3}{12}f''(\xi)$$

The Simpson's Rule

Simpson's Rule uses one more point for Lagrange polynomial. $x_0 = a$, $x_1 = a+h$, $x_2 = b$, $h = \frac{b-a}{2}$.

$$\begin{aligned}\int_a^b f(x)dx &= \int_{x_0}^{x_2} f(x_0)\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}dx + \int_{x_0}^{x_2} f(x_1)\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}dx \\ &\quad + \int_{x_0}^{x_2} f(x_2)\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}dx + \int_{x_0}^{x_2} f^{(3)}(\xi(x))(x-x_0)(x-x_1)(x-x_2)\cdot\frac{1}{3!}dx \\ &= \frac{h}{3}f(x_0) + \frac{4h}{3}f(x_1) + \frac{h}{3}f(x_2) + E(x)\end{aligned}$$

$$E(x) = \frac{1}{6}\int_{x_0}^{x_2} f^{(3)}(\xi(x))(x-x_0)(x-x_1)(x-x_2)dx$$

Due to $E(x)$ now is an odd function, $E(x) = 0$. Thus, the error is not determined on a polynomial with 3 degree. Alternatively, we use Hermite interpolation by adding a restriction $P_3'(x_1) = f'(x_1)$.

$$\begin{aligned}
E(x) &= \frac{1}{4!}\int_{x_0}^{x_2} f^{(4)}(\xi(x))(x-x_0)(x-x_1)^2(x-x_2)dx \\
&= \frac{1}{24}f^{(4)}(\xi)\int_{x_0}^{x_2}(x-x_0)(x-x_1)^2(x-x_2)dx \quad \text{(WMVT)},\ \xi \in (a,b) \\
&= \frac{1}{24}f^{(4)}(\xi)\int_{-h}^{h}(t+h)(t^2)(t-h)dt \quad (t = x - x_1) \\
&= \frac{1}{24}f^{(4)}(\xi)\int_{-h}^{h} t^4 - h^2t^2dt = -\frac{h^5}{90}f^{(4)}(\xi)
\end{aligned}$$

4.3 Adaptive Quadrature Methods

Adaptive Quadrature is to logically optimize computational resources. Standard composite formulas use a uniform step size, which oversamples regions of low functional variation and undersamples regions of high variation.

Let $S(a,b)$ be the Simpson's Rule for $\int_a^b f(x)dx$. $S(a,b) \triangleq \frac{h}{3}[f(a) + 4f(a+h) + f(b)]$, where $h = \frac{b-a}{2}$, then

$$\begin{cases} S(a, \frac{a+b}{2}) = \frac{h}{6}[f(a) + 4f(a+\frac{h}{2}) + f(a+h)] \\ S(\frac{a+b}{2}, b) = \frac{h}{6}[f(a+h) + 4f(a+\frac{3}{2}h) + f(a+2h)] \end{cases}$$

$$\begin{aligned}
\int_a^b f(x)dx &= S(a, \frac{a+b}{2}) + S(\frac{a+b}{2}, b) - \left(\frac{1}{2}\right)^4 \frac{h^5}{90}f^{(4)}(\xi), \quad \xi \in (a,b) \\
&= S(a,b) - \frac{h^5}{90}f^{(4)}(\xi_1), \quad \xi_1 \in (a,b)
\end{aligned}$$

Suppose $\xi_1 \approx \xi$, $f^{(4)}(\xi_1) \approx f^{(4)}(\xi)$:

$$S(a, \frac{a+b}{2}) + S(\frac{a+b}{2}, b) - \frac{1}{16}\frac{h^5}{90}f^{(4)}(\xi) \approx S(a,b) - \frac{h^5}{90}f^{(4)}(\xi_1)$$

$$\Rightarrow \underbrace{\frac{h^5}{90}f^{(4)}(\xi)}_{\text{Error}} \approx \frac{16}{15}\left[S(a,b) - S(a, \frac{a+b}{2}) - S(\frac{a+b}{2}, b)\right]$$

4.4 Gaussian Quadrature

Gaussian quadrature chooses the points for evaluation in an optimal way. The $x_1 \dots x_n$ in the interval, $c_1 \dots c_n$ are chosen to minimize the expected error obtained in the approximation.

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} c_i f(x_i), \quad x_i \in [a,b] \ \forall i = 1 \sim n$$

Consider $n = 2$, $[a,b] = [-1,1]$:

$$\int_{-1}^{1} f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

Due to we have four parameters, we construct a 3-order polynomial to determine these parameter. Then,

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \approx c_1 f(x_1) + c_2 f(x_2)$$

$$\int_{-1}^{1} f(x)dx = a_0 \int_{-1}^{1} 1dx + a_1 \int_{-1}^{1} xdx + a_2 \int_{-1}^{1} x^2 dx + a_3 \int_{-1}^{1} x^3 dx$$

If

$$\begin{aligned} f(x) = 1 &\Rightarrow c_1 + c_2 = 2 \\ f(x) = x &\Rightarrow c_1 x_1 + c_2 x_2 = 0 \\ f(x) = x^2 &\Rightarrow c_1 x_1^2 + c_2 x_2^2 = \frac{2}{3} \\ f(x) = x^3 &\Rightarrow c_1 x_1^3 + c_2 x_2^3 = 0 \end{aligned}$$

$$\Rightarrow \begin{cases} c_1 = 1 \\ c_2 = 1 \\ x_1 = -\frac{\sqrt{3}}{3} \\ x_2 = \frac{\sqrt{3}}{3} \end{cases}$$

which gives the formula $\int_{-1}^{1} f(x)dx = f\left(-\frac{\sqrt{3}}{3}\right) + f\left(\frac{\sqrt{3}}{3}\right)$.

Legendre Polynomials

This polynomial can get the parameter needed in Gaussian quadrature more easily. **Legendre Polynomials** is a collection $\{P_0(x), P_1(x), \dots, P_n(x)\}$ with the following properties:

1. For each n, $P_n(x)$ is a monic polynomial of degree n ($a_n = 1$).
2. $\int_{-1}^{1} P(x)P_n(x)dx = 0$, whenever $P(x)$ is a polynomial $\deg(P(x)) < n$ which means

$P_n(x)$ and $P(x)$ are orthogonal to each other.

Theorem 4.1.
Suppose that $x_1 \dots x_n$ are the roots of nth Legendre polynomial $P_n(x)$. For each $i = 1 \dots n$, the c_i are defined by

$$c_i = \int_{-1}^{1} \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} dx = \int_{-1}^{1} L_i(x) dx$$

If $P(x)$ is any polynomial of degree less than $2n$, then

$$\int_{-1}^{1} P(x) dx = \sum_{i=1}^{n} c_i P(x_i)$$

Proof.
Case 1: $\deg(f(x)) < n$

$$f(x) = \sum_{i=1}^{n} L_i(x) f(x_i)$$

and

$$\begin{aligned} \int_{-1}^{1} f(x) dx &= \int_{-1}^{1} \sum_{i=1}^{n} L_i(x) f(x_i) dx \\ &= \sum_{i=1}^{n} \int_{-1}^{1} L_i(x) f(x_i) dx \\ &= \sum_{i=1}^{n} \left(\int_{-1}^{1} L_i(x) dx \right) f(x_i) \\ &= \sum_{i=1}^{n} c_i f(x_i) \end{aligned}$$

Case 2: $n \leq \deg(f(x)) < 2n$
Let $P(x) = Q(x)P_n(x) + R(x)$, where $Q(x)$, $R(x)$ are polynomial with degree less than n, $P_n(x)$ is the Legendre polynomial with order n, x_i are the roots of $P_n(x)$. Thus for $i = 1 \dots n$:

$$f(x_i) = Q(x_i)P_n(x_i) + R(x_i) = R(x_i)$$

$$\int_{-1}^{1} f(x) dx = \int_{-1}^{1} R(x) dx = \sum_{i=1}^{n} c_i R(x_i)$$

■

Consider evaluating $\int_a^b f(x) dx$ over an arbitrary $[a, b]$. Let $t = \frac{2x-a-b}{b-a}$, $x = \frac{1}{2}[(b-a)t +$

$a+b]$, $dt = \frac{2}{b-a}dx$:

$$\int_a^b f(x)dx = \int_{-1}^{1} f\left(\frac{(b-a)t+(b+a)}{2}\right)\cdot\frac{(b-a)}{2}dt$$

4.5 Improper Integrals

Left Endpoint Singularity

Consider $\int_a^b \frac{g(x)}{(x-a)^p}dx$, which converges iff $0 < p < 1$, where $g(x) \in C^4[a,b]$ and $0 < p < 1$. In this case, we can construct the forth Taylor Polynomial $P_4(x)$ at $x = a$ of $g(x)$.

$$P_4(x) = g(a) + g'(a)(x-a) + \frac{g''(a)}{2!}(x-a)^2 + \frac{g'''(a)}{3!}(x-a)^3 + \frac{g^{(4)}(a)}{4!}(x-a)^4$$

$\because g(x) = P_4(x) + R_4(x) \Rightarrow R_4(x) = g(x) - P_4(x)$

$$\therefore f(x) = \frac{g(x)}{(x-a)^p} = \frac{P_4(x)}{(x-a)^p} + \frac{R_4(x)}{(x-a)^p} = \frac{P_4(x)}{(x-a)^p} + \frac{g(x)-P_4(x)}{(x-a)^p}$$

$$\int_a^b \frac{P_4(x)}{(x-a)^p}dx = \sum_{k=0}^{4}\frac{g^{(k)}(a)}{k!(k+1-p)}(b-a)^{k+1-p}$$

To determine $\int_a^b \frac{g(x)-P_4(x)}{(x-a)^p}dx$, Define

$$G(x) = \begin{cases} \frac{g(x)-P_4(x)}{(x-a)^p} & \text{if } a < x \le b \\ 0 & \text{if } x = a \end{cases}$$

This gives $G(x)$ continuous on $[a,b]$.

5 Solve the Linear System of Equations

There are two methods to solve the linear system of equations:

- **Direct Methods**: These methods find the exact solution in a finite number of steps. Examples include Gaussian elimination, LU decomposition, and Cholesky decomposition.
- **Iterative Methods**: These methods start with an initial guess and iteratively refine it to approach the solution. Examples include Jacobi method, Gauss-Seidel method, and Conjugate Gradient method.

5.1 Matrix Norms and Condition Number

To effectively evaluate the stability of a linear system and mathematically bound the errors generated by these solvers, we must explicitly define vector and matrix norms.

Vector Norms

For a vector $x = [x_1, x_2, \ldots, x_n]^T \in \mathbb{R}^n$:

- **1-norm:** $\|x\|_1 \triangleq \sum_{i=1}^{n} |x_i|$.
- **p-norm ($p > 0$):** $\|x\|_p \triangleq (\sum_{i=1}^{n} |x_i|^p)^{\frac{1}{p}}$.
- **Infinity-norm:** $\|x\|_\infty \triangleq \max_{1 \le i \le n} |x_i|$.

Matrix Norms

For a matrix $A \in \mathbb{R}^{m \times n}$ with entries a_{ij}:

- **Frobenius Norm (F-norm):** $\|A\|_F \triangleq \left(\sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2 \right)^{\frac{1}{2}}$.
- **Matrix p-norm:** $\|A\|_p \triangleq \sup_{x \in \mathbb{R}^n, x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}$.
- **Column sum norm (1-norm):** $\|A\|_1 = \max_{1 \le j \le n} \sum_{i=1}^{m} |a_{ij}|$.
- **Row sum norm (∞-norm):** $\|A\|_\infty = \max_{1 \le i \le m} \sum_{j=1}^{n} |a_{ij}|$.

Condition Number and Error Bounds

Consider a nonsingular matrix A. If x^* is the exact analytical solution to $Ax^* = b$ and x^k is a numerical approximation, we strictly define the residual vector as $r = b - Ax^k$.

Theorem 5.1.
Suppose the matrix A is nonsingular. The relative error of the approximated solution is bounded by:

$$\frac{\|x^* - x^k\|}{\|x^*\|} \le \|A\|\|A^{-1}\| \frac{\|r\|}{\|b\|}$$

Proof.
By definition, $r = b - Ax^k = Ax^* - Ax^k = A(x^* - x^k)$.

$$\Rightarrow x^* - x^k = A^{-1}r \Rightarrow \|x^* - x^k\| \le \|A^{-1}\|\|r\|$$

Additionally, since $b = Ax^*$, we have $\|b\| \le \|A\|\|x^*\|$, which implies $\|x^*\| \ge \frac{\|b\|}{\|A\|}$. Combining these inequalities gives:

$$\frac{\|x^* - x^k\|}{\|x^*\|} \le \frac{\|A^{-1}\|\|r\|}{\|b\|/\|A\|} = \|A\|\|A^{-1}\| \frac{\|r\|}{\|b\|}$$

■

Definition 5.1 (Condition Number)**.**
The condition number of a matrix A is defined as:

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

If $\kappa(A) \approx 1$, the matrix is well-conditioned. If $\kappa(A) \gg 1$, the matrix is considered ill-conditioned, and numerical solutions will be highly sensitive to perturbations.

5.2 Direct Methods

5.2.1 Gaussian Elimination

The structural process of Gaussian elimination can be mathematically modeled as a sequential application of elementary linear transformations. Consider a 2-dimensional vector $x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^T$ where the pivot $x_1 \neq 0$. There exists an elementary transformation matrix M such that the second entry is eliminated:

$$Mx = \begin{bmatrix} x_1 \\ 0 \end{bmatrix}$$

By defining the multiplier $\alpha = -x_2/x_1$, the transformation matrix M takes the form:

$$M = \begin{bmatrix} 1 & 0 \\ \alpha & 1 \end{bmatrix}$$

More generally, in $\mathbb{R}^n$, let $x = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix}^T$ with the k-th pivot $x_k \neq 0$. We seek an elimination matrix M_k that annihilates all entries strictly below the pivot:

$$M_k x = \begin{bmatrix} x_1 \\ \vdots \\ x_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

This transformation is defined algebraically as a rank-1 update to the identity matrix:

$$M_k = I - \alpha e_k^T$$

where e_k is the k-th standard basis vector (indicating the k-th position), and the vector

α contains the essential multipliers:

$$\alpha = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \alpha_{k+1} \\ \vdots \\ \alpha_n \end{bmatrix}, \quad \text{with } \alpha_i = \frac{x_i}{x_k} \quad \text{for } k+1 \leq i \leq n$$

Remark: The negative sign for elimination is inherently absorbed by the subtraction operator in the definition $I - \alpha e_k^T$.

By systematically applying these elementary lower triangular matrices (Frobenius matrices) to a dense matrix A, we execute the forward elimination phase. Consider a generic 4×4 dense matrix A. The geometric progression of zeroing out sub-diagonal entries is visualized as follows:

$$A = \begin{bmatrix} \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \\ \times & \times & \times & \times \end{bmatrix} \xrightarrow{M_1} M_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \otimes & \otimes & \otimes \\ 0 & \otimes & \otimes & \otimes \\ 0 & \otimes & \otimes & \otimes \end{bmatrix}$$

$$\xrightarrow{M_2} M_2 M_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \otimes & \otimes & \otimes \\ 0 & 0 & \star & \star \\ 0 & 0 & \star & \star \end{bmatrix} \xrightarrow{M_3} M_3 M_2 M_1 A = \begin{bmatrix} \times & \times & \times & \times \\ 0 & \otimes & \otimes & \otimes \\ 0 & 0 & \star & \star \\ 0 & 0 & 0 & \uparrow \end{bmatrix}$$

The conclusive matrix product, $M_3 M_2 M_1 A$, successfully reduces the original system into an **upper triangular matrix**. This state mathematically guarantees that the exact solution can be swiftly extracted via backward substitution.

Algorithm

Given $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, we seek x such that $Ax = b$. The Gaussian elimination process can be algorithmically implemented as follows:

```
def gaussian_elimination(A, b):
    n = len(A)
    # Forward elimination
    for k in range(n):
        for i in range(k+1, n):
            if A[k][k] == 0:
```

```
                raise ValueError("Zero pivot encountered.")
            alpha = A[i][k] / A[k][k]

            for j in range(k, n):
                A[i][j] -= alpha * A[k][j]
            b[i] -= alpha * b[k]

    # Backward substitution
    x = [0] * n
    for i in range(n-1, -1, -1):
        sum_ax = sum(A[i][j] * x[j] for j in range(i+1, n))
        x[i] = (b[i] - sum_ax) / A[i][i]

    return x
```

Example

Consider the following system:

$$A = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 11 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

We construct the elimination matrices M_1 and M_2 to systematically annihilate the sub-diagonal entries of the first and second columns, respectively:

$$M_1 = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{bmatrix}, \quad M_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -2 & 1 \end{bmatrix}$$

Applying these transformations yields the upper triangular matrix U and the updated right-hand side vector $\tilde{b}$:

$$U = M_2 M_1 A = \begin{bmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 2 \end{bmatrix}, \quad \tilde{b} = M_2 M_1 b = \begin{bmatrix} 1 \\ -1 \\ 0 \end{bmatrix}$$

To solve the equivalent system $Ux = \tilde{b}$, we execute backward substitution:

$$x_3 = 0, \quad -3x_2 - 6(0) = -1 \implies x_2 = 1/3, \quad x_1 + 4(1/3) + 7(0) = 1 \implies x_1 = -1/3$$

Thus, the solution vector is exactly:

$$x = \begin{bmatrix} -1/3 \\ 1/3 \\ 0 \end{bmatrix}$$

5.2.2 LU Decomposition

Assume that after $k-1$ steps, the intermediate matrix is partitioned as follows:

$$A^{(k-1)} = M_{k-1} \cdots M_1 A = \left[\begin{array}{c|c} A_{11}^{(k-1)} & A_{12}^{(k-1)} \\ \hline 0 & A_{22}^{(k-1)} \end{array}\right] \begin{array}{l} k-1 \\ n-k+1 \end{array}$$

where $A_{11}^{(k-1)}$ is an upper triangular matrix, and $A_{22}^{(k-1)}$ is the remaining submatrix:

$$A_{22}^{(k-1)} = \begin{bmatrix} a_{kk}^{(k-1)} & \cdots & a_{kn}^{(k-1)} \\ \vdots & \ddots & \vdots \\ a_{nk}^{(k-1)} & \cdots & a_{nn}^{(k-1)} \end{bmatrix}$$

Assuming the pivot element $a_{kk}^{(k-1)} \neq 0$, we define the elimination matrix $M_k = I - \alpha^{(k)} e_k^T$, where the multiplier vector $\alpha^{(k)}$ is given by:

$$\alpha^{(k)} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{bmatrix}, \quad \text{where } l_{ik} \triangleq \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}, \quad \text{for } k+1 \leq i \leq n$$

Applying M_k advances the elimination process:

$$A^{(k)} = M_k A^{(k-1)} = \left[\begin{array}{c|c} A_{11}^{(k)} & A_{12}^{(k)} \\ \hline 0 & A_{22}^{(k)} \end{array}\right] \begin{array}{l} k \\ n-k \end{array}$$

Finally, after $n-1$ steps, we obtain the upper triangular matrix U:

$$A^{(n-1)} = \left(\prod_{i=1}^{n-1} M_i \right) A = U$$

That is to say, the original matrix A can be decomposed as:

$$A = \left(\prod_{i=1}^{n-1} M_i \right)^{-1} U = LU$$

Utilizing the property $(I - \alpha^{(k)} e_k^T)(I + \alpha^{(k)} e_k^T) = I$, we can explicitly construct L:

$$L = \prod_{i=1}^{n-1} (I + \alpha^{(i)} e_i^T) = I + \sum_{i=1}^{n-1} \alpha^{(i)} e_i^T$$

Here, L is a unit lower triangular matrix (i.e., having 1s on its diagonal part).

Algorithm

```python
import numpy as np

def lu_decomposition(A: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    # Performs LU decomposition of a square matrix A.
    # Returns L (unit lower triangular) and U (upper triangular).
    n = A.shape[0]
    L = np.eye(n) # Initialize L as identity matrix
    U = A.copy().astype(float) # Initialize U as a copy of A

    for k in range(n - 1):
        if U[k, k] == 0:
            raise ValueError(f"Zero pivot encountered at index {k}.")

        for i in range(k + 1, n):
            # Calculate the multiplier l_ik
            factor = U[i, k] / U[k, k]
            L[i, k] = factor

            # Apply the elementary row operation
            # U[i] = U[i] - factor * U[k]
            U[i, k:] -= factor * U[k, k:]

    return L, U

# --- Example Usage based on your notes ---
if __name__ == "__main__":
```

```
    A = np.array([[1, 4, 7],
                  [2, 5, 8],
                  [3, 6, 11]], dtype=float)
    L, U = lu_decomposition(A)

    print("Matrix L:\n", L)
    print("Matrix U:\n", U)
    # Verification: np.allclose(L @ U, A) will return True
```

5.3 Iterative Methods

For solving a linear system $Ax = b$, we define the residual vector as $r \triangleq b - Ax$. There are two main types of iterative methods:

1. Stationary methods
2. Nonstationary methods

We can decompose the matrix A as $A = D + L + U$, where:

- $D = \text{diag}(A)$
- $L =$ strictly lower triangular part of A
- $U =$ strictly upper triangular part of A

Thus, the system $Ax = b$ can be rewritten as $(D + L + U)x = b$.

Stopping Criteria for Iterative Methods

Before introducing the specific methods, we define the standard stopping criteria for iterative algorithms. As shown in the general iteration loop, the process terminates when either the step size or the residual falls below predefined tolerances (ε_1 and ε_2):

$$\|x^{(k)} - x^{(k-1)}\| < \varepsilon_1 \quad \text{or} \quad \|b - Ax^{(k)}\| < \varepsilon_2$$

5.3.1 Jacobi Method

Given an initial value $x^{(0)}$, the Jacobi method updates the solution iteratively:

$$Dx^{(k)} = -(L + U)x^{(k-1)} + b, \quad k = 1, 2, \ldots$$

Equivalently:

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

Algorithm

```python
import numpy as np

def jacobi_iteration(A: np.ndarray, b: np.ndarray, x0: np.ndarray,
                     tolerance: float = 1e-6, max_iter: int = 100) -> tuple[np.ndarray, int]:
    """
    Solves the linear system Ax = b using the Jacobi iterative method.
    """
    n = len(A)
    x = x0.copy().astype(float)
    x_new = np.zeros_like(x)

    for k in range(max_iter):
        for i in range(n):
            # Calculate the sum of a_ij * x_j for j != i
            s = sum(A[i, j] * x[j] for j in range(n) if j != i)
            # Update the i-th component of the new state
            x_new[i] = (b[i] - s) / A[i, i]

        # Calculate the residual norm to check for convergence
        if np.linalg.norm(x_new - x, ord=np.inf) < tolerance:
            return x_new, k + 1

        x = x_new.copy()

    print("Warning: Maximum iterations reached without convergence.")
    return x, max_iter
```

5.3.2 Gauss-Seidel Method

Unlike the Jacobi method, which updates the solution vector only after computing all components, the Gauss-Seidel method utilizes the newly computed components $x_i^{(k)}$ as soon as they become available. Using the matrix decomposition $A = D + L + U$, the Gauss-Seidel iteration is formulated as:

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

By isolating $x^{(k)}$, we obtain the explicit iteration form:

$$x^{(k)} = \underbrace{-(D+L)^{-1}U}_{T} x^{(k-1)} + (D+L)^{-1}b$$

where T serves as the iteration matrix.

Algorithm: Gauss-Seidel Iteration

Input: Initial guess $x^{(0)}$, stopping tolerances $\varepsilon_1, \varepsilon_2$, maximum iterations N
For $k = 1 : N$
  For $i = 1 : n$

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

  End
  If $\|x^{(k)} - x^{(k-1)}\| < \varepsilon_1$ **or** $\|b - Ax^{(k)}\| < \varepsilon_2$ **then**
    Stop
  End
End

Convergence of Fixed-Point Iterations

Both the Jacobi and Gauss-Seidel methods are specific instances of fixed-point iterations, which can be generally expressed in the form:

$$x^{(k)} = Tx^{(k-1)} + c$$

To establish the conditions under which these sequences converge, we introduce the following foundational lemma and theorem regarding the iteration matrix T. **Lemma.** If

the spectral radius $\rho(T)$ satisfies $\rho(T) < 1$, then the matrix $(I - T)$ is invertible (i.e., $(I - T)^{-1}$ exists), and its inverse can be expanded as a series:

$$(I - T)^{-1} = I + T + T^2 + \cdots = \sum_{i=0}^{\infty} T^i$$

Theorem 5.2.
For any arbitrary initial value $x^{(0)} \in \mathbb{R}^n$, the sequence $\{x^{(k)}\}_{k=1}^{\infty}$ defined by $x^{(k)} = Tx^{(k-1)} + c$ converges to the unique solution x^* of $x = Tx + c$ if and only if:

$$\rho(T) < 1$$

Corollary. As a practical consequence, if $\|T\| < 1$ for any natural matrix norm, the iteration is mathematically guaranteed to converge.

5.3.3 Conjugate Gradient Method (CG Method)

The **Conjugate Gradient (CG) method** is an advanced iterative algorithm designed to solve systems of linear equations $Ax = b$, where the matrix $A \in \mathbb{R}^{n\times n}$ is strictly symmetric and positive definite. A fundamental property of the CG method is that it constructs the solution space over a sequence of basis vectors $\{v^{(i)}\}_{i=1}^{n}$ which form an A-orthogonal (or conjugate) set. Mathematically, this conjugacy condition is expressed as:

$$\langle v^{(k)}, Av^{(i)}\rangle = 0, \quad \text{for } i = 1, 2, \ldots, k-1$$

Algorithm: CG Method

0. Given an initial guess $x^{(0)}$ and a specified tolerance ε.

1. Initialize the residual and the initial search direction:

$$r^{(0)} = b - Ax^{(0)}$$
$$v^{(1)} = r^{(0)}$$

2. **For** $k = 1 : n$

Compute the step size t_k:

$$t_k = \frac{\langle r^{(k-1)}, r^{(k-1)}\rangle}{\langle v^{(k)}, Av^{(k)}\rangle}$$

Update the solution and the residual vector:

$$x^{(k)} = x^{(k-1)} + t_k v^{(k)}$$
$$r^{(k)} = r^{(k-1)} - t_k Av^{(k)}$$

If $\|r^{(k)}\| < \varepsilon$, **then stop**.

Compute the orthogonalization coefficient s_k:

$$s_k = \frac{\|r^{(k)}\|_2^2}{\|r^{(k-1)}\|_2^2}$$

Update the search direction for the next iteration:

$$v^{(k+1)} = r^{(k)} + s_k v^{(k)}$$

End

Projection Methods

To generalize the principles underlying the CG method, we introduce projection methods. At each iteration k, we define two distinct subspaces: the search subspace $\mathcal{K}_k$ and the constraint subspace $\mathcal{L}_k$, with respective bases V_k and W_k. We enforce the condition that the approximate solution resides in the affine space:

$$x^{(k)} \in x^{(0)} + \mathcal{K}_k \implies x^{(k)} = x^{(0)} + V_k y_k$$

A projection method where the constraint subspace is identical to the search subspace ($\mathcal{L}_k = \mathcal{K}_k$) is formally called the **Galerkin method**. For the CG method specifically, the search space is the **Krylov subspace**, defined as:

$$\mathcal{K}_k = \text{span}\{r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \ldots, A^{k-1}r^{(0)}\}$$

6 Approximation Theory

While Krylov subspace methods are optimal for solving square deterministic systems, empirical observations often lead to overdetermined systems. Consider an experiment evaluating Hooke's Law for a spring, modeled as:

$$e + kF = l$$

where l is the measured length, F is the applied force, and the constants e (initial length) and k (compliance) are unknown. Suppose an experiment yields the following dataset:

F	1	2	3	4	5
l	7.97	10.2	14.2	16.0	21.2

To determine the unknowns e and k, we can formulate this as a linear system:

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix} \begin{bmatrix} e \\ k \end{bmatrix} = \begin{bmatrix} 7.97 \\ 10.2 \\ 14.2 \\ 16.0 \\ 21.2 \end{bmatrix}$$

Because this system has more equations than unknowns, an exact solution rarely exists. This naturally leads to the necessity of **Discrete Least Squares Approximation**. Assume we have a general group of data points $\{(x_i, y_i)\}_{i=1}^n$. We wish to represent these data using a predefined functional form. If we assume a linear function $y = ax + b$, the

objective shifts to answering a fundamental question: how do we determine the optimal parameters a and b to minimize the discrepancy between the model and the data?

6.1 Discrete Least Squares Approximation

Error Definitions in Approximation

To quantify the discrepancy between the measured data points $\{(x_i, y_i)\}_{i=1}^n$ and our proposed linear model $y = ax + b$, we define three fundamental types of measurement errors:

1. **Maximum Error (Chebyshev norm):** $E_\infty(a, b) \triangleq \max_{1 \le i \le n} |y_i - (ax_i + b)|$
2. **Absolute Error (L1 norm):** $E_1(a, b) \triangleq \sum_{i=1}^n |y_i - (ax_i + b)|$
3. **Least Squares Error (L2 norm):** $E_2(a, b) \triangleq (\sum_{i=1}^n [y_i - (ax_i + b)]^2)^{1/2}$

In general, we determine the optimal parameters a and b based on minimizing the smallest error. The most analytically tractable choice is the Best Least Squares approximation. To find the minimum, we set the partial derivatives of the squared error to zero:

$$\frac{\partial (E_2)^2}{\partial a} = 0, \quad \frac{\partial (E_2)^2}{\partial b} = 0$$

This differentiation yields a system of equations:

$$-2\sum_{i=1}^{n} \big(y_i - (ax_i + b)\big)x_i = 0 \implies a\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$
$$-2\sum_{i=1}^{n} \big(y_i - (ax_i + b)\big) = 0 \implies a\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i$$

Solving this 2×2 linear system provides the exact analytical expressions for a and b.

The Normal Equations

To answer the question of determining the optimal parameters $x = \begin{bmatrix} b \\ a \end{bmatrix}$ (or $x = \begin{bmatrix} e \\ k \end{bmatrix}$ in our Hooke's Law example) for an overdetermined system $Ax = y$, we must acknowledge that an exact solution rarely exists. Instead, we seek to minimize the L_2 norm of the residual vector, $\|Ax - y\|_2^2$. Geometrically, the optimal approximation occurs when the residual vector $(y - Ax)$ is orthogonal to the column space of A. This orthogonality principle is algebraically enforced by multiplying both sides of our system by the transpose of the matrix, A^T:

$$A^T Ax = A^T y$$

These are known as the **Normal Equations**. Provided that the columns of A are linearly independent (which is typical for distinct data points in experimental models), the square

matrix A^TA is symmetric and positive definite. This guarantees that its inverse exists, leading to the unique least squares solution:

$$x = (A^TA)^{-1}A^Ty$$

This derivation conceptually unites our previous discussions on projection methods, orthogonalization, and solving systems involving symmetric positive definite matrices.

Polynomial Least Squares Approximation

If we consider approximating the function using a polynomial of degree m for the data $\{(x_i, y_i)\}_{i=1}^N$ (where $0 < m < N-1$):

$$P_m(x) = a_m x^m + a_{m-1}x^{m-1} + \cdots + a_1 x + a_0$$

The total least squares error becomes:

$$E_2(a_0, \ldots, a_m) \triangleq \sum_{i=1}^{N} \big(y_i - P_m(x_i)\big)^2$$

Similarly, we set $\frac{\partial E_2}{\partial a_k} = 0$ for $k = 0, 1, \ldots, m$ to find the minimum value. This leads to a linear system of equations $Au = b$:

$$\begin{bmatrix} N & \sum x_i & \cdots & \sum x_i^m \\ \sum x_i & \sum x_i^2 & \cdots & \sum x_i^{m+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_i^m & \sum x_i^{m+1} & \cdots & \sum x_i^{2m} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} \sum y_i \\ \sum y_i x_i \\ \vdots \\ \sum y_i x_i^m \end{bmatrix}$$

> **Note:** You can use iterative methods like the CG method to solve this linear system. In software environments like MATLAB, built-in functions such as `polyfit` or `lsqlin` can be used to efficiently approximate these linear functions.

6.2 Continuous Least Squares Approximation

Suppose $f \in C[a, b]$. We want to approximate it through a polynomial of degree at most n, $P_n(x) = \sum_{k=0}^n a_k x^k$. The error function is defined continuously as:

$$E_2(a_0, a_1, \ldots, a_n) \triangleq \int_a^b [f(x) - P_n(x)]^2 dx = \|f - P_n\|_2^2$$

To minimize this error, for each index $j = 0, 1, \ldots, n$, we set $\frac{\partial E_2}{\partial a_j} = 0$:

$$\frac{\partial}{\partial a_j}\int_a^b (f^2 - 2fP_n + P_n^2)dx = -2\int_a^b x^j f(x)dx + 2\sum_{k=0}^{n} a_k \int_a^b x^{j+k}dx = 0$$

$$\implies \sum_{k=0}^{n} a_k \int_a^b x^{j+k}dx = \int_a^b x^j f(x)dx$$

This leads to the continuous counterpart of our matrix system $Au = b$:

$$\begin{bmatrix} \int_a^b 1\,dx & \int_a^b x\,dx & \cdots & \int_a^b x^n\,dx \\ \int_a^b x\,dx & \int_a^b x^2\,dx & \cdots & \int_a^b x^{n+1}\,dx \\ \vdots & \vdots & \ddots & \vdots \\ \int_a^b x^n\,dx & \int_a^b x^{n+1}\,dx & \cdots & \int_a^b x^{2n}\,dx \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} \int_a^b f(x)\,dx \\ \int_a^b xf(x)\,dx \\ \vdots \\ \int_a^b x^n f(x)\,dx \end{bmatrix}$$

Remark: The monomials $\{1, x, x^2, \ldots, x^n\}$ act as the basis for the subspace of $P_n(x)$. However, this basis can lead to ill-conditioned matrices (the Hilbert matrix). We can consider another subspace with orthogonal bases $\{\phi_i(x)\}_{i=1}^n$ for better numerical stability. Let Π_n be the set of all polynomials of degree at most n. Any polynomial $P \in \Pi_n$ can be represented as a linear combination of these basis functions: $P = \sum_{i=1}^n c_i\phi_i(x)$.

6.3 Orthogonal Polynomials

1. Legendre Polynomials

A simple basis function set for approximation on the interval $[a, b] = [-1, 1]$ is the monic Legendre polynomials:

$$\begin{aligned} P_0(x) &= 1 \\ P_1(x) &= x \\ P_2(x) &= x^2 - \frac{1}{3} \\ P_3(x) &= x^3 - \frac{3}{5}x \\ P_4(x) &= x^4 - \frac{6}{7}x^2 + \frac{3}{35} \end{aligned}$$

These satisfy the orthogonality condition: $\int_{-1}^1 P_k(x)P_j(x)dx = 0$ if $k \neq j$.

2. Chebyshev Polynomials

Defined as $T_n(x) \triangleq \cos(n \cos^{-1} x)$ for $n \geq 0$ and $x \in [-1, 1]$. The sequence $\{T_n(x)\}_{n=1}^N$ is orthogonal with respect to the weighting function $w(x) = \frac{1}{\sqrt{1-x^2}}$:

$$\int_{-1}^{1} T_m(x) T_n(x) w(x) dx = 0 \quad \text{if } m \neq n$$

Let $\theta = \cos^{-1} x \implies x = \cos\theta$. Then $T_n(\theta) = \cos(n\theta)$. Utilizing trigonometric identities:

$$T_{n+1}(\theta) = \cos((n+1)\theta) = \cos(n\theta)\cos\theta - \sin(n\theta)\sin\theta$$
$$T_{n-1}(\theta) = \cos((n-1)\theta) = \cos(n\theta)\cos\theta + \sin(n\theta)\sin\theta$$

Adding these yields the recurrence relation: $T_{n+1}(\theta) + T_{n-1}(\theta) = 2\cos(n\theta)\cos\theta$. Converting back to x:

$$T_{n+1}(x) = 2xT_n(x) - T_{n-1}(x) \quad \text{for } n \geq 1$$

Generating the first few polynomials:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_2(x) = 2xT_1(x) - T_0(x) = 2x^2 - 1$$
$$T_3(x) = 2xT_2(x) - T_1(x) = 4x^3 - 3x$$

7 Approximating Eigenvalues

7.1 Singular Value Decomposition (SVD)

Theorem 7.1 (Eckart-Young-Mirsky Theorem for Low-Rank Approximation)**.**
If $\text{rank}(A) = r$ (i.e., the dimension of the matrix A), and its SVD is $A = \sum_{i=1}^{r} \sigma_i u_i v_i^T$. For some constant k where $1 \leq k \leq r$, let the truncated SVD be $A_k \triangleq \sum_{i=1}^{k} \sigma_i u_i v_i^T$. Then for any matrix $\hat{A} \in \mathbb{R}^{m \times n}$ with $\text{rank}(\hat{A}) = k$, the minimum approximation error in the 2-norm is exactly the $(k+1)$-th singular value:

$$\min_{\text{rank}(\hat{A})=k} \|A - \hat{A}\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

Proof.
First, we verify that A_k achieves this error bound:

$$\|A - A_k\|_2 = \left\|U(A - A_k)V^T\right\|_2 = \|\text{diag}(0, \ldots, 0, \sigma_{k+1}, \sigma_{k+2}, \ldots, \sigma_p)\|_2 = \sigma_{k+1}$$

Since $\text{rank}(\hat{A}) = k$, for all $\hat{A} \in \mathbb{R}^{m\times n}$, there exist orthonormal vectors $\{x_1, \ldots, x_{n-k}\}$ such that $\text{Null}(\hat{A}) = \text{span}\{x_1, \ldots, x_{n-k}\}$. (If $y \in \text{Null}(\hat{A})$, $y = \sum_{i=1}^{n-k} c_i x_i$, then $\hat{A}y = 0$). Assume we construct the intersection space $Z = \text{span}\{x_1, x_2, \ldots, x_{n-k}\} \cap \text{span}\{v_1, v_2, \ldots, v_{k+1}\}$. Because the sum of the dimensions is $(n-k)+(k+1) = n+1 > n$, this intersection space is non-empty ($Z \neq \{0\}$). Then $\exists z \in Z$ with $\|z\|_2 = 1$ such that $\hat{A}z = 0$. We evaluate Az:

$$Az = \left(\sum_{i=1}^{k+1} \sigma_i u_i v_i^T\right) z = \sum_{i=1}^{k+1} \sigma_i (v_i^T z) u_i$$

$$\implies \|Az\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z)^2$$

Therefore, the error is bounded from below:

$$\|A - \hat{A}\|_2^2 \geq \|(A - \hat{A})z\|_2^2 = \|Az\|_2^2 = \sum_{i=1}^{k+1} \sigma_i^2 (v_i^T z)^2 \geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} (v_i^T z)^2 = \sigma_{k+1}^2 \|z\|_2^2 = \sigma_{k+1}^2$$

Hence, $\|A - \hat{A}\|_2 \geq \sigma_{k+1}$. ■

Remark: If $\text{rank}(\hat{A}) = r_\varepsilon$ with a threshold $\varepsilon = M\|A\|_2$, then $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{r_\varepsilon} > \varepsilon > \sigma_{r_\varepsilon+1} \geq \cdots \geq \sigma_p \geq 0$. We say $\hat{A} \sim A$ within the significance of error ε. **Geometric Interpretation of Singular Values (Hyper-space):** Consider $A = \begin{bmatrix} 0.96 & 1.72 \\ 1.72 & 0.96 \end{bmatrix}$. The SVD is $A = U\Sigma V^T = \begin{bmatrix} \frac{3}{5} & -\frac{4}{5} \\ \frac{4}{5} & \frac{3}{5} \end{bmatrix} \begin{bmatrix} 3 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{4}{5} & \frac{3}{5} \\ \frac{3}{5} & -\frac{4}{5} \end{bmatrix}^T$. If we define a set $E = \{y \mid y = Ax, \forall x \in \mathbb{R}^2, \|x\|_2 = 1\}$, the singular values represent the lengths of the semi-axes of the resulting hyper-ellipse.

7.2 Orthogonal Projection

Let $S = \text{span}\{v\}$ for a vector $v \in \mathbb{R}^n$. We seek a matrix P such that for any $x \in \mathbb{R}^n$, $Px \in S$ and the residual $(I - P)x \in S^\perp$. Let $P \triangleq \frac{vv^T}{\|v\|_2^2} = \left(\frac{v}{\|v\|_2}\right)\left(\frac{v}{\|v\|_2}\right)^T$. Therefore, for any $x \in \mathbb{R}^n$:

$$Px = v\left(\frac{v^T x}{\|v\|_2^2}\right)$$

This matrix P satisfies:

$$\begin{cases} P^2 = P \cdot P = P \\ P^T = P \end{cases}$$

so we call P an **orthogonal projection**. If $V = [v_1, v_2, \ldots, v_k]$ with orthonormal basis v_i for S, then $P = VV^T$ is the unique orthogonal projection onto S. **Projections from SVD:**

Let $A = [U_r, \bar{U}_r]\Sigma[V_r, \bar{V}_r]^T$ be the SVD of A, where $\text{rank}(A) = r$. $U_r = [u_1, \ldots, u_r]$, $\bar{U}_r = [u_{r+1}, \ldots, u_m]$, $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_p)$, $p = \min(m, n)$. $V_r = [v_1, \ldots, v_r]$. Then we have the following results:

1. $P = V_r V_r^T$ is the orthogonal projection onto $N(A)^\perp = R(A^T)$.
2. $P = \bar{V}_r \bar{V}_r^T$ is the orthogonal projection onto $N(A)$.
3. $P = U_r U_r^T$ is the orthogonal projection onto $R(A)$.
4. $P = \bar{U}_r \bar{U}_r^T$ is the orthogonal projection onto $R(A)^\perp = N(A^T)$.

Application: Distance Between Subspaces

To measure the distance between two subspaces $S_1, S_2 \in \mathbb{R}^n$ with $\dim(S_1) = \dim(S_2)$. Assume P_1, P_2 are two orthogonal projections for S_1, S_2 respectively. Then:

$$\text{dist}(S_1, S_2) = \|P_1 - P_2\|_2$$

Example: Let $S_1 = \text{span}\{x\} \in \mathbb{R}^2$ where $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$, and $S_2 = \text{span}\{y\} \in \mathbb{R}^2$ where $y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$. Assume $\|x\|_2 = \|y\|_2 = 1$. The angle between them is defined by $x^T y = \cos\theta$, for $0 \leq \theta \leq \frac{\pi}{2}$.

$$P_1 = \frac{xx^T}{x^T x} = xx^T, \quad P_2 = \frac{yy^T}{y^T y} = yy^T$$

The difference is $P_1 - P_2 = xx^T - yy^T$. Through SVD analysis, this difference matrix has non-zero singular values equal to $\sin\theta$.

$$\therefore \text{dist}(S_1, S_2) = \|P_1 - P_2\|_2 = \sin\theta$$

Solving Non-linear Vector Functions

Consider solving a non-linear vector function $F(x) = 0$, where $F : \mathbb{R}^n \to \mathbb{R}^m$. For example, let $F : \mathbb{R}^n \to \mathbb{R}^1$ be a quadratic function:

$$F(x) = \frac{1}{2}x^T Ax - x^T b, \quad A \in \mathbb{R}^{n \times n},\ x, b \in \mathbb{R}^n$$

Using the gradient method (such as CG) for solving $Ax = b$, we essentially find the roots of the derivative:

$$\frac{\partial F}{\partial x} = Ax - b = 0$$

to ensure the minimum of $F(x)$. To generalize this to broader non-linear systems, we must establish rigorous definitions for multivariable derivatives.

8 Multivariable Calculus

8.1 F-Differentiable (Fréchet Derivative)

Recall Newton's method in 1D: for $f(x) = 0$, the update rule is $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$. To extend this, let $D \subset \mathbb{R}^n$ be a given domain and let $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$ be a given non-linear function.

Definition 8.1 (F-Differentiable)**.**
We say F is F-differentiable at $x \in D$ if there is a matrix $A \in \mathbb{R}^{m\times n}$ such that for all $\varepsilon > 0$, there is a $\delta > 0$ satisfying:

$$\|F(x+h) - F(x) - Ah\| \leq \varepsilon\|h\|$$

for all $h \in \mathbb{R}^n$ with $\|h\| < \delta$ and $x + h \in D$.

Remark:

1. If A exists, A is the **Jacobian matrix**. $A = (a_{ij})$ for $1 \leq i \leq m$, $1 \leq j \leq n$, where $a_{ij} = \frac{\partial F_i}{\partial x_j}$ and $F = [F_1, F_2, \ldots, F_m]^T$.
2. We say F is continuously differentiable, denoted by $F \in C^1(D)$, if $F'(x)$ is continuous.

Example 1:

$$f(x_1, x_2) = \begin{cases} x_1 & \text{if } x_2 = 0 \\ x_2 & \text{if } x_1 = 0 \\ 1 & \text{otherwise} \end{cases}$$

where $f : \mathbb{R}^2 \to \mathbb{R}^1$. The partial derivatives $\frac{\partial f}{\partial x_1}(0)$ and $\frac{\partial f}{\partial x_2}(0)$ exist. However, f is not F-differentiable at 0 because it is not continuous there.

8.2 G-Differentiable (Gâteaux Derivative)

Definition 8.2 (G-Differentiable)**.**
We say F is G-differentiable at $x \in D$ if there is a matrix $A \in \mathbb{R}^{m\times n}$ such that for all $h \in \mathbb{R}^n$:

$$\lim_{t\to 0} \frac{1}{t}\|F(x+th) - F(x) - tAh\| = 0$$

Example 2:

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = 0 \\ \frac{2x_2 e^{-x_1^2}}{x_2^2 + 2^{-2x_1^2}} & \text{if } x_1 \neq 0 \end{cases}$$

f has a G-derivative at 0, but f is discontinuous at $x = 0$.

Example 3:

$$f(x_1, x_2) = \begin{cases} 0 & \text{if } (x_1, x_2) = (0, 0) \\ \frac{x_2(x_1^2+x_2^2)^{3/2}}{x_1^4+x_2^2} & \text{if } (x_1, x_2) \neq (0, 0) \end{cases}$$

This is a classic function used to show that f has a G-derivative at $x = 0$, but not an F-derivative at $x = 0$.

Exercise

- **Lemma 1:** If $F : D \subset \mathbb{R}^n \to \mathbb{R}^m$ is F-differentiable, then F is continuous.
- **Lemma 2:** If F is F-differentiable at $x \in D$, then $\frac{\partial F_i}{\partial x_j}$ exists at $x \in D$ and forms the Jacobian matrix.

9 Systems of Nonlinear Equations

Solving a nonlinear vector function $F(x) = 0$, where $F : \mathbb{R}^n \to \mathbb{R}^n$, requires extrapolating iterative scalar methods into multi-dimensional spaces.

9.1 Fixed-Point Iteration for Systems

Let $G : D \subset \mathbb{R}^n \to \mathbb{R}^n$ where D is an open set. The iterative process identifies the fixed point $x^* \in D$ such that $x^* = G(x^*)$ by continuously applying the recurrence relation:

$$x_{k+1} = G(x_k)$$

A fixed point x^* is defined theoretically as a "point of attraction" if there exists a bounded continuous region $B \subset D$ encompassing x^* such that $G(B) \subset B$ and any initialization $x_0 \in B$ results in $x_k \to x^*$. If $G \in C^1(D)$ and the spectral radius of $G'(x^*)$ is strictly less than 1, the iteration converges reliably. This is intrinsically linked to the Contraction Mapping Theorem, stating that if $\|G(x) - G(y)\| \leq \alpha\|x - y\|$ with $0 < \alpha < 1$, then the sequence will be a Cauchy sequence and inevitably converge linearly.

9.2 Newton's Method for Systems

Consider $F : D \subset \mathbb{R}^n \to \mathbb{R}^n$ where $F(x^*) = 0$. Using the multidimensional Taylor expansion near x_k:

$$F(x) = F(x_k) + F'(x_k)(x - x_k) + R(x, x_k)$$

Setting the linear tangent approximation $T_k(x_{k+1}) = 0$ provides the iterative update rule:

$$0 = F(x_k) + F'(x_k)(x_{k+1} - x_k) \implies x_{k+1} = x_k - [F'(x_k)]^{-1}F(x_k)$$

In practical computational implementations, explicitly calculating the matrix inverse is severely inefficient. Instead, we structurally define a step vector s_k and solve the equivalent linear system:

$$F'(x_k)s_k = -F(x_k)$$

$$x_{k+1} = x_k + s_k$$

This method achieves local quadratic convergence provided the Jacobian $F'(x^*)$ is definitively nonsingular. To eliminate the significant overhead of computing analytic derivatives, the Jacobian matrix can be programmatically approximated utilizing forward finite differences:

$$A_{ij}(x) \triangleq \frac{f_i(x + he_j) - f_i(x)}{h}$$

where e_j is the canonical standard basis vector and the perturbation factor h is generally scaled to $h \approx \sqrt{\epsilon_{machine}}$ to perfectly balance theoretical truncation error against numerical round-off error.

9.3 Inexact Newton's Method and GMRES

Solving the exact linear system $F'(x_k)s_k = -F(x_k)$ incurs an extreme $\mathcal{O}(n^3)$ cost per iteration. The Inexact Newton's Method deliberately relaxes this strict condition, solving the system approximately such that the residual decreases up to a designated tolerance limit:

$$\|F'(x_k)s_k + F(x_k)\| \leq \eta_k \|F(x_k)\|$$

For highly asymmetric and sparse Jacobian configurations, the Generalized Minimum Residual (GMRES) method becomes strictly necessary. GMRES approximates the iterative solution by effectively minimizing the residual norm over a continuously expanding Krylov subspace $\mathcal{K}_k = \text{span}\{r_0, Ar_0, \ldots, A^{k-1}r_0\}$. The Finite Difference GMRES (FDGMRES) variant operates entirely matrix-free by directly estimating the Jacobian-vector products:

$$F'(x)v \approx \frac{F(x + hv) - F(x)}{h}$$

FDGMRES methodically constructs an orthonormal basis V_k using the Arnoldi process, converting the high-dimensional problem into minimizing $\|\beta e_1 - H_k y_k\|$ across a much smaller Upper-Hessenberg matrix H_k.

10 Numerical Solutions for Boundary Value Problems

To accurately model physical displacement distributions, we utilize specialized discretization frameworks.

10.1 Finite Difference Method (FDM)

To numerically solve a boundary value problem, the Finite Difference Method discretizes the domain into a uniform grid. Consider a theoretical physical string displacement problem constrained by:

$$-u''(x) = f(x), \quad 0 < x < 1$$

$$u(0) = u(1) = 0$$

We partition the interval uniformly, defining $x_i = ih$ for $i = 0, 1, \ldots, n$ where $h = \frac{1}{n}$. The continuous second derivative is analytically approximated employing the central difference scheme:

$$u''(x_i) = \frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} + \mathcal{O}(h^2)$$

Substituting this localized approximation directly into the governing differential equation yields:

$$-\left(\frac{u_{i-1} - 2u_i + u_{i+1}}{h^2}\right) = f(x_i)$$

This explicitly translates the continuum calculus problem into an algebraically structured linear system $Au = b$. Because each node connects exclusively to its immediate adjacent neighbors, the resulting interaction matrix A becomes a definitively tridiagonal matrix scaled by $\frac{1}{h^2}$, with diagonal elements measuring 2 and sub-diagonal elements evaluating to -1.

10.2 Finite Element Method (FEM)

The Finite Element Method approaches the differential equation problem by reformulating it integrally into its "weak form". Multiplying the initial equation by a generic test function $v(x)$ and integrating continuously over the entire domain generates:

$$-\int_0^1 u''v\,dx = \int_0^1 fv\,dx$$

Performing integration by parts directly on the left side transfers one structural derivative onto the specific test function. Given the inherent boundaries $v(0) = v(1) = 0$, the

boundary integral terms identically evaluate to zero, concluding with:

$$\int_0^1 u'v'\,dx = \int_0^1 fv\,dx$$

Instead of mathematically solving for isolated discrete points, the global solution space is systematically projected onto a highly localized finite-dimensional subspace formed by piecewise linear basis "hat" functions $\phi_j(x)$. Letting $u(x) \approx \sum_{j=1}^{n-1} c_j\phi_j(x)$ and choosing $v(x) = \phi_i(x)$, we assemble a discrete linear system $Ac = b$:

$$A_{ij} = \int_0^1 \phi_i'(x)\phi_j'(x)\,dx, \quad b_i = \int_0^1 f(x)\phi_i(x)\,dx$$

Because A operates as structurally symmetric and definitively positive definite, it perfectly pairs with Conjugate Gradient solvers.