Contents

Chapter 1: Solving Linear Equations **2**

1.1 Problem Formulation and Types of Systems . . . 2

1.2 Existence and Uniqueness of Solutions . . . 2

1.3 Gaussian Elimination and LU Decomposition . . . 2

1.4 Error, Stability, and Conditioning . . . 3

1.4.1 Vector and Matrix Norms . . . 3

1.4.2 Condition Number . . . 4

Chapter 2: Curve Fitting **4**

2.1 Interpolation: Curve Passes *Through* Data Points . . . 4

2.1.1 Lagrange Interpolating Polynomial . . . 4

2.1.2 Spline Interpolation . . . 4

2.2 Regression: Least-Squares Curve Fitting . . . 5

2.2.1 Linear Regression ($y = a_1 x + a_0$) . . . 5

2.2.2 High-Order Polynomial Regression ($p_M(x) = a_0 + a_1 x + \cdots + a_M x^M$) . . . 5

Chapter 3: Solving Nonlinear Equations **5**

3.1 Bisection Method . . . 6

3.2 Newton's Method (Newton-Raphson) . . . 6

3.3 Secant Method . . . 7

Chapter 4: Numerical Derivative and Integral **7**

4.1 Numerical Differentiation . . . 7

4.2 Numerical Integration (Quadrature) . . . 7

Chapter 5: Ordinary Differential Equations (ODEs) **8**

5.1 Initial Value Problems (IVPs) . . . 8

5.2 Euler's Method . . . 8

5.3 Runge-Kutta (RK) Methods . . . 8

5.4 Systems of ODEs . . . 9

Chapter 6: Unconstrained Minimization **9**

6.1 Optimality Conditions . . . 9

6.2 Iterative Minimization Methods . . . 10

Chapter 7: Linear Programming **10**

7.1 Problem Formulation and Duality . . . 10

7.2 The Simplex Method . . . 10

Chapter 1: Solving Linear Equations

1.1 Problem Formulation and Types of Systems

The primary goal is to find the solution vector $\boldsymbol{x}$ for a system of m linear equations with n variables, represented as $\boldsymbol{Ax} = \boldsymbol{b}$.

Types of Linear Systems

- **Square System ($m = n$):** Number of equations equals number of unknowns. Often has a unique solution.
- **Underdetermined (Missing) System ($m < n$):** Fewer equations than unknowns. Usually has infinitely many solutions.
- **Overdetermined (Residual) System ($m > n$):** More equations than unknowns. Usually has no solution.

1.2 Existence and Uniqueness of Solutions

- **Determinant Rule:** For a square system, a unique solution exists if and only if the matrix $\boldsymbol{A}$ is non-singular, meaning $\det(\boldsymbol{A}) \neq 0$. If $\det(\boldsymbol{A}) = 0$, the system has either no solution or infinitely many.

- **Rank Rule (Kronecker-Capelli Theorem):** A system $\boldsymbol{Ax} = \boldsymbol{b}$ has at least one solution if and only if the rank of the coefficient matrix equals the rank of the augmented matrix: $\text{rank}(\boldsymbol{A}) = \text{rank}([\boldsymbol{A}|\boldsymbol{b}])$.

1.3 Gaussian Elimination and LU Decomposition

Gaussian elimination is the core method for solving $\boldsymbol{Ax} = \boldsymbol{b}$. It decomposes $\boldsymbol{A}$ into $\boldsymbol{L}$ and $\boldsymbol{U}$ matrices, accounting for pivoting with $\boldsymbol{P}$. The relationship is $\boldsymbol{PA} = \boldsymbol{LU}$.

Solving $\boldsymbol{Ax} = \boldsymbol{b}$ with LU Factorization by Hand

1. **Forward Elimination (Factorization):**
   - Start with the augmented matrix $[\boldsymbol{A}|\boldsymbol{b}]$. Keep track of row swaps in a permutation vector/matrix $\boldsymbol{P}$.
   - For each step k from 1 to $n-1$:

     1.a. **Pivoting:** Find the row $i \geq k$ with the largest absolute value in column k. Swap row i with row k. Record this swap in $\boldsymbol{P}$.

     1.b. **Factor Calculation:** For each row j below the pivot row ($j > k$), calculate the factor: $L_{jk} = a_{jk}/a_{kk}$.

     1.c. **Elimination:** Update row j: $\text{Row}_j \leftarrow \text{Row}_j - L_{jk} \times \text{Row}_k$. This creates zeros below the pivot a_{kk}.
   - The resulting upper triangular matrix is $\boldsymbol{U}$. The matrix of factors is $\boldsymbol{L}$ (with 1s on the diagonal). The transformed vector is $\boldsymbol{d}$.
2. **Backward Substitution:**
   - You now have the system $\boldsymbol{Ux} = \boldsymbol{d}$.
   - Solve for x_n from the last equation: $x_n = d_n/u_{nn}$.
   - Substitute back to find x_{n-1}, then x_{n-2}, up to x_1:

$$x_i = \frac{1}{u_{ii}}\left(d_i - \sum_{j=i+1}^{n} u_{ij}x_j\right)$$

1.4 Error, Stability, and Conditioning

1.4.1 Vector and Matrix Norms

Norms measure the "size" or "length" of vectors and matrices.

- **Vector Norms:**
  - L_1-norm (Manhattan): $||\boldsymbol{x}||_1 = \sum |x_i|$
  - L_2-norm (Euclidean): $||\boldsymbol{x}||_2 = \sqrt{\sum x_i^2}$
  - L_∞-norm (Max): $||\boldsymbol{x}||_\infty = \max_i |x_i|$
- **Matrix Norms (Induced):**
  - $||A||_1 = \max_j \sum_i |a_{ij}|$ (Max absolute column sum)
  - $||A||_\infty = \max_i \sum_j |a_{ij}|$ (Max absolute row sum)

1.4.2 Condition Number

Condition Number The condition number $\text{cond}(\boldsymbol{A})$ measures how sensitive the solution $\boldsymbol{x}$ is to small changes in $\boldsymbol{A}$ or $\boldsymbol{b}$.

$$\text{cond}(\boldsymbol{A}) = ||\boldsymbol{A}|| \cdot ||\boldsymbol{A}^{-1}||$$

- If $\text{cond}(\boldsymbol{A}) \approx 1$, the matrix is **well-conditioned**. Small changes in input lead to small changes in output.
- If $\text{cond}(\boldsymbol{A}) \gg 1$, the matrix is **ill-conditioned**. Small changes in input can lead to large changes in output.
- The relative error in the solution is bounded by:

$$\frac{||\Delta \boldsymbol{x}||}{||\boldsymbol{x}||} \leq \text{cond}(\boldsymbol{A}) \frac{||\Delta \boldsymbol{b}||}{||\boldsymbol{b}||}$$

Chapter 2: Curve Fitting

2.1 Interpolation: Curve Passes *Through* Data Points

2.1.1 Lagrange Interpolating Polynomial

Lagrange Formula For a set of $N+1$ points (x_i, f_i), the unique polynomial of degree at most N passing through them is:

$$p_N(x) = \sum_{i=0}^{N} f_i \cdot L_i(x)$$

where the Lagrange Basis Polynomial $L_i(x)$ is defined as:

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{N} \frac{x - x_j}{x_i - x_j} = \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_N)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_N)}$$

2.1.2 Spline Interpolation

Motivation for Splines High-degree Lagrange polynomials can oscillate wildly between data points (Runge's phenomenon). Splines avoid this by using a series of connected, lower-degree polynomials, creating a smoother curve.

- **Linear Spline:** Connects points with straight lines. For $x \in [x_{i-1}, x_i]$:

$$S_i(x) = f_{i-1} + \left(\frac{f_i - f_{i-1}}{x_i - x_{i-1}} \right) (x - x_{i-1})$$

- **Cubic Spline:** Uses cubic polynomials for each segment. To be uniquely defined, it requires additional constraints at the endpoints, such as:

- **Natural Spline:** Second derivatives at the endpoints are zero: $S''(x_0) = 0, S''(x_N) = 0$.
- **Not-a-Knot Spline:** The third derivatives are continuous at the first and last interior knots.

2.2 Regression: Least-Squares Curve Fitting

Finds a function that best approximates the data by minimizing the sum of squared errors $E = \sum(f_i - p(x_i))^2$.

2.2.1 Linear Regression ($y = a_1 x + a_0$)

The coefficients a_0 and a_1 are found by solving the 2×2 system of **Normal Equations**:

$$\left(\sum_{i=1}^{N} x_i^2\right) a_1 + \left(\sum_{i=1}^{N} x_i\right) a_0 = \sum_{i=1}^{N} (x_i f_i)$$
$$\left(\sum_{i=1}^{N} x_i\right) a_1 + N a_0 = \sum_{i=1}^{N} f_i$$

2.2.2 High-Order Polynomial Regression ($p_M(x) = a_0 + a_1 x + \cdots + a_M x^M$)

This generalizes linear regression. Solve the $(M+1) \times (M+1)$ system of normal equations:

$$\begin{pmatrix} N & \sum x_i & \dots & \sum x_i^M \\ \sum x_i & \sum x_i^2 & \dots & \sum x_i^{M+1} \\ \vdots & \vdots & \ddots & \vdots \\ \sum x_i^M & \sum x_i^{M+1} & \dots & \sum x_i^{2M} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_M \end{pmatrix} = \begin{pmatrix} \sum f_i \\ \sum f_i x_i \\ \vdots \\ \sum f_i x_i^M \end{pmatrix}$$

Chapter 3: Solving Nonlinear Equations

Goal: Find the root x such that $f(x) = 0$. These methods are iterative.

3.1 Bisection Method

Bisection Method

- **Geometric Idea:** Repeatedly halve the interval that is known to contain a root.
- **Requirement:** An initial interval $[a, b]$ where $f(a)$ and $f(b)$ have opposite signs ($f(a) \cdot f(b) < 0$).
- **Algorithm:**
  1. Calculate the midpoint: $c = \frac{a+b}{2}$.
  2. Evaluate $f(c)$.
  3. If $f(a) \cdot f(c) < 0$, the root is in the left half. Set $b \leftarrow c$.
  4. Otherwise, the root is in the right half. Set $a \leftarrow c$.
  5. Repeat until the interval $|b - a|$ is sufficiently small.
- **Convergence:** Linear. Reliable and guaranteed to converge, but slow.

3.2 Newton's Method (Newton-Raphson)

Newton's Method

- **Geometric Idea:** Start at an initial guess x_0. Find the tangent line to the curve at that point. The next guess, x_1, is the root of that tangent line.
- **Requirement:** An initial guess x_0 and the ability to compute the derivative $f'(x)$.
- **Iterative Formula:**
$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$
- **Convergence:** Quadratic (very fast) if the initial guess is close to the root. Can diverge if the guess is poor or if $f'(x_k) \approx 0$.

3.3 Secant Method

Secant Method

- **Geometric Idea:** A modification of Newton's method. Instead of a tangent line, it uses a secant line passing through the two most recent points to find the next guess.
- **Requirement:** Two initial guesses, x_0 and x_1.
- **Iterative Formula:**

$$x_{k+1} = x_k - f(x_k) \cdot \left(\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \right)$$

- **Convergence:** Superlinear (rate ≈ 1.618). Faster than Bisection but slower than Newton's. An excellent practical choice when the derivative is unavailable.

Chapter 4: Numerical Derivative and Integral

4.1 Numerical Differentiation

Finite Difference Formulas

- **Forward Difference:** $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ (Order of error $O(h)$)
- **Backward Difference:** $f'(x) \approx \frac{f(x)-f(x-h)}{h}$ (Order of error $O(h)$)
- **Central Difference (1st Deriv):** $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$ (Order of error $O(h^2)$)
- **Central Difference (2nd Deriv):** $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$ (Order of error $O(h^2)$)

The central difference formulas are more accurate for the same step size h.

4.2 Numerical Integration (Quadrature)

Approximate $I = \int_a^b f(x)dx$. We divide $[a, b]$ into n subintervals of width $h = (b - a)/n$.

Composite Trapezoidal Rule **Idea:** Approximate the area under the curve using n trapezoids.

$$I \approx \frac{h}{2} \left[f(x_0) + 2 \sum_{i=1}^{n-1} f(x_i) + f(x_n) \right]$$

Composite Simpson's 1/3 Rule **Idea:** Approximate the area using parabolas over pairs of intervals. Requires an **even** number of intervals, n.

$$I \approx \frac{h}{3}\left[f(x_0) + 4\sum_{\text{i odd}} f(x_i) + 2\sum_{\text{i even}} f(x_i) + f(x_n)\right]$$

The pattern of coefficients is 1, 4, 2, 4, ..., 2, 4, 1. More accurate than the trapezoidal rule.

Chapter 5: Ordinary Differential Equations (ODEs)

5.1 Initial Value Problems (IVPs)

The goal is to find the function $y(t)$ that satisfies:

$$y'(t) = f(t, y), \quad \text{with an initial condition} \quad y(t_0) = y_0$$

5.2 Euler's Method

Forward Euler Method **Idea:** Use the slope at the beginning of an interval to project a straight line to the next point.

$$y_{i+1} = y_i + h \cdot f(t_i, y_i)$$

This method is simple but has low accuracy ($O(h)$) and can be unstable.

5.3 Runge-Kutta (RK) Methods

RK methods improve accuracy by using a weighted average of slopes at several points within each interval.

Fourth-Order Runge-Kutta (RK4) The most commonly used RK method. It has high accuracy ($O(h^4)$).

1. **Calculate four intermediate slopes:**

$$\begin{aligned}
k_1 &= h \cdot f(t_i, y_i) && \text{(Slope at the beginning)} \\
k_2 &= h \cdot f\left(t_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) && \text{(Slope at the first midpoint)} \\
k_3 &= h \cdot f\left(t_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) && \text{(Slope at the second midpoint)} \\
k_4 &= h \cdot f(t_i + h, y_i + k_3) && \text{(Slope at the end)}
\end{aligned}$$

2. **Calculate the next value** using a weighted average of these slopes:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

5.4 Systems of ODEs

A higher-order ODE can be converted into a system of first-order ODEs.

Example: Converting a 2nd-Order ODE Given $y'' - 0.05y' + 0.15y = 0$, with $y(0) = 1, y'(0) = 0$.

1. **Define new variables:** Let $z_1 = y$ and $z_2 = y'$.
2. **Create the system:**
   - The first equation is the definition of z_2: $z_1' = y' = z_2$.
   - The second equation is from the original ODE: $z_2' = y'' = 0.05y' - 0.15y = 0.05z_2 - 0.15z_1$.
3. **The First-Order System is:**

$$\begin{aligned} z_1' &= z_2, & z_1(0) &= 1 \\ z_2' &= -0.15z_1 + 0.05z_2, & z_2(0) &= 0 \end{aligned}$$

This system can now be solved with methods like Euler or RK4, applying the formulas to the vector $\boldsymbol{z} = [z_1, z_2]^T$.

Chapter 6: Unconstrained Minimization

Goal: Find the vector $\boldsymbol{x} = [x_1, \dots, x_n]^T$ that minimizes a multivariable function $f(\boldsymbol{x})$.

6.1 Optimality Conditions

Conditions for a Local Minimum

- **First-Order Necessary Condition:** The gradient at a local minimum $\boldsymbol{x}^*$ must be the zero vector.

$$\nabla f(\boldsymbol{x}^*) = \begin{pmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_n \end{pmatrix}_{\boldsymbol{x}=\boldsymbol{x}^*} = \mathbf{0}$$

  Points satisfying this are called **stationary points**.

- **Second-Order Sufficient Condition:** A stationary point $\boldsymbol{x}^*$ is a strict local minimum if its Hessian matrix $\boldsymbol{H}(\boldsymbol{x}^*)$ is **positive definite**.

$$\boldsymbol{H} = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

  To check for positive definiteness by hand (Sylvester's Criterion), all leading principal minors of the Hessian must be positive.

6.2 Iterative Minimization Methods

Gradient Descent Method **Idea:** Start at $\boldsymbol{x}_0$ and iteratively take steps in the direction of the negative gradient (the direction of steepest descent).

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha_k \nabla f(\boldsymbol{x}_k)$$

where α_k is the step size or "learning rate". For hand-solved problems, α_k is often a fixed constant or determined by a line search.

Newton's Method for Optimization **Idea:** At each step, approximate $f(\boldsymbol{x})$ with a quadratic function (a paraboloid) and jump to the minimum of that approximation.

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - [\boldsymbol{H}(\boldsymbol{x}_k)]^{-1} \nabla f(\boldsymbol{x}_k)$$

This is equivalent to solving the linear system $\boldsymbol{H}(\boldsymbol{x}_k)\boldsymbol{p}_k = -\nabla f(\boldsymbol{x}_k)$ for the step direction $\boldsymbol{p}_k$, and then updating $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{p}_k$. It is very fast but computationally expensive.

Chapter 7: Linear Programming

7.1 Problem Formulation and Duality

A linear programming (LP) problem seeks to optimize a linear objective function subject to linear constraints.

Primal and Dual Problems Every LP problem (the Primal) has a corresponding Dual problem.

Primal Problem (Minimization)	**Dual Problem (Maximization)**
$\min \boldsymbol{c}^T \boldsymbol{x}$	$\max \boldsymbol{b}^T \boldsymbol{y}$
subject to:	subject to:
$\boldsymbol{A}\boldsymbol{x} \geq \boldsymbol{b}$	$\boldsymbol{A}^T \boldsymbol{y} \leq \boldsymbol{c}$
$\boldsymbol{x} \geq \boldsymbol{0}$	$\boldsymbol{y} \geq \boldsymbol{0}$

Strong Duality Theorem: If either problem has an optimal solution, so does the other, and their optimal objective values are equal.

7.2 The Simplex Method

Core Idea of the Simplex Method The optimal solution to an LP, if it exists, must occur at a **vertex** (corner point) of the feasible region. The Simplex Method is an efficient algorithm that travels from vertex to adjacent vertex, improving the objective function value at each step until the optimum is found.

The Simplex Algorithm in Tabular Form (for standard max problems) A standard max problem has constraints of the form $\leq$ and non-negative variables.

1. **Step 1: Setup Initial Tableau.**
   - Convert each constraint $\sum a_{ij}x_j \leq b_i$ into an equation by adding a non-negative **slack variable** s_i: $\sum a_{ij}x_j + s_i = b_i$.
   - Rewrite the objective function as $z - \sum c_j x_j = 0$.
   - Construct the tableau. The initial **basic variables** are the slacks.
2. **Step 2: Check for Optimality.**
   - Look at the bottom (indicator) row. If all entries corresponding to variables are **non-negative** (≥ 0), the current solution is optimal. **STOP**.
3. **Step 3: Select Pivot Column (Entering Variable).**
   - Find the **most negative** value in the bottom row. The column containing it is the **pivot column**. The corresponding variable is the **entering variable**.
4. **Step 4: Select Pivot Row (Leaving Variable).**
   - For each row, calculate the **test ratio**: Ratio $= \frac{\text{Value in Solution Column}}{\text{Value in Pivot Column}}$.
   - Ratios are only calculated for rows where the pivot column entry is **strictly positive** (> 0).
   - The row with the **smallest non-negative ratio** is the **pivot row**. The basic variable in this row is the **leaving variable**.
5. **Step 5: Perform Pivot Operation to find New Tableau.**
   - The element at the intersection of the pivot row and column is the **pivot element**.
   - Divide the entire pivot row by the pivot element. This becomes the new pivot row.
   - For every other row (including the bottom row), update it using the rectangular rule:
     New Row = Old Row - (Pivot Column Entry in that Row) $\times$ (New Pivot Row).
6. **Step 6: Repeat.** Go back to Step 2.

Note: For minimization, the rules in steps 2 and 3 are reversed (stop if all ≤ 0, pick most positive).