COMPY02

Alianjecon Otero
alianjecon_otero@dlsu.edu.ph

July 28, 2025

Contents

0 Syllabus

0.1 COMPY02

Computer Programming for Physics Majors 2

This is an intermediate course in computer programming in MATLAB for B.S. Physics majors with introduction to computational methods. The students will learn how to construct logical formulations or algorithms in arriving at finite numerical solution/s to various scientific problems. Rudiments of computer programming in MATLAB with hands-on training will be incorporated in this course.

0.2 Course-Level Outcomes

- Translate formulas and algorithms into MATLAB statements and subroutines and devise a logical sequence of precise steps in solving scientific problems.
- Trace and debug syntactical and logical errors design and implement MATLAB programs from scratch and analyze and optimize existing MATLAB programs.
- Practice intellectual honesty. Apply existing algorithms to solving scientific problems and practice good programming techniques.
- Volunteer and share the knowledge in Physics for the under-privileged.

0.3 Requirements and Grading

- Machine Problems
- Presentations
- Final Project
- Service Learning Project

Grading System:
Long Exams: 50%
Final Machine Project: 25%
Portfolio of Products: 25%

Score Range	**Grade**
94 - 100	4.0
87 - 93	3.5
80 - 86	3.0
73 - 79	2.5
66 - 72	2.0
58 - 65	1.5
50 - 57	1.0
< 50	0.0

0.4 Topics

- Measuring and limiting errors
- Converting Binary to Decimal
- Differentiation
- Methods in Solving Nonlinear equations through computational methods
- Arrays and multidimensional Arrays
- Solving simultaneous linear equations through computational methods

1 Module 1

1.1 Introduction to Numerical Analysis

Analytic vs. Numerical Solutions

- **Analytic solutions**, also called *closed-form solutions*, are obtained through symbolic manipulations of exact mathematical formulas.
  - **Example:** In an ideal spring-mass system (point mass m, spring constant k), what is the work done by the elastic force in bringing the point mass from the equilibrium point to a displacement of $+x$ (spring is stretched)?

$$W = \int_0^x F(x')\,dx' = \int_0^x (-kx')\,dx' = -\frac{1}{2}kx^2$$

  - If you have a specific numerical value for x, you can just plug it into the exact solution obtained here.
- **Numerical solutions** are approximations that rely on the numerical values of the quantities and not on the exact analytic solutions or mathematical equations.
  - Numerical solutions are now oftentimes associated with computational approaches, i.e., those that use computer programs and algorithms.
  - But that was not always the case. In the early days of mathematics, numerical solutions were done by hand, sometimes in painstakingly iterative fashion. Early Greek mathematicians used numerical solutions.
  - For the ideal spring-mass problem, a numerical approach might approximate the integral as a sum of the areas of rectangular strips that approximate the actual area. You should know the value of x beforehand.
  - The result is also just a number, not a mathematical expression or formula. The final numerical value may be different depending on the choice of approximations used (e.g., width of each rectangular strip).

Advantages of Numerical Approaches

- Sometimes, there are problems where the use of analytic solutions is disadvantageous.
  - For example, in cases where there are no closed-form solutions to the problems.
  - In some cases, analytic solutions exist, but the complexity is intractable.
- Under any of these circumstances, it may be helpful to get numerical solutions.

Some Historical Context

- Archimedes used numerical approximations based on geometry to find a good estimate for the value of π, the ratio of the circumference to the diameter of a circle.
  - Inscribing regular polygons within the circular bounds gave better and better estimates for π as the number of polygonal sides increased.

- Newton, who invented calculus, also designed an approach to finding the roots of functions using numerical approximations:

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

  - This is known as the Newton-Raphson method.

Examples and Applications

The following are just some of the problems where numerical solutions can be used:

- Roots of nonlinear equations
- Solutions of a set of linear equations
- Curve fitting
- Numerical differentiation and integration
- Ordinary differential equations

2 Module 2

2.1 Errors

Significant Figures

- Physics is a quantitative discipline.
- There are inherent limitations in measurements; there are no infinitely precise measurements.
  - Rulers can only report in millimeters.
  - Calipers can measure to fractions of millimeters.
  - Microscopic measurements reach up to the fractions of microns.

Rules for Determining Significant Figures

- Nonzero digits are significant.
- All zeros sandwiched between nonzero digits are significant.
- The zero to the left of the decimal point for fractional (below 1) values is not significant.
- All zeros to the right of the decimal point before the first nonzero digit are not significant.
- All zeros to the right of the decimal point after the last nonzero digit are significant.

Scientific Notation

- A way of reporting numbers in their corresponding powers of ten.
- **Goal:** First nonzero digit on the left of the decimal point, followed by the other digits, followed by the corresponding power of ten as a multiplier.
  - Moving the decimal point leftwards: Add positive powers of ten.
  - Moving the decimal point rightwards: Add negative powers of ten.

True Errors (Absolute and Relative)

- The true error of a numerical computation is given by:

$$TE = TV - AV$$

- The true error can only be computed if there is a true (exact) value.
  - Usually, this is used for gauging the accuracy of the numerical method versus analytic solutions.
- This difference is also reported in absolute value.
  - In some cases, the reporting requires the actual sign.
- The relative true error of a numerical computation is given by:

$$RE = \frac{TE}{TV} = \frac{TV - AV}{TV}$$

Approximate Errors (Absolute and Relative)

- The approximate error of a numerical computation is given by:

$$AE = \text{Present} - \text{Previous}$$

- The approximate error is useful when there is no available true (exact) value.

Tolerance in Numerical Approaches

- In a numerical method that uses iterative methods, a user can calculate relative approximate error ϵ_a at the end of each iteration.
- The user may pre-specify a minimum acceptable tolerance called the pre-specified tolerance, ϵ_s.
- If the absolute value of the relative approximate error satisfies:

$$|\epsilon_a| \leq \epsilon_s$$

- Then the acceptable error has been reached and no more iterations would be required.

2.2 Binary to Decimal

Introduction: Computational Numerical Approaches

Advantages of Using Computers in Numerical Approaches

- Numerical approximations involve repetitive calculations.
- Computers are best for performing numerical approaches due to:
  - Speed
  - Accuracy
  - Automation

Historical Details

- The first programmable computer, the Electronic Numerical Integrator and Computer (ENIAC), was developed to help the war effort.
- Since then, computers have become more ubiquitous, with the availability of personal computing devices to almost everyone.
- Modern computing based on electronic circuits was made possible due to the invention of the transistor.
- Transistors act as switches with ON/OFF states.
- Hence, computers use binary representation to represent, manipulate, and store numbers.

Decimal Representation

- The decimal system uses 10 unique symbols to represent numbers.
- Numbers are reported in powers of 10, 10^n:
  - 1 is the basic unit (10^0).
  - Nonzero numbers ≥ 1 appear to the left of the decimal point.
  - Nonzero numbers < 1 appear to the right.

Binary Representation

- The binary system uses only 2 symbols: 0 and 1.
- Numbers are reported in powers of 2, 2^n:
  - 1 is the basic unit (2^0).
  - Nonzero numbers ≥ 1 appear to the left of the binary point.
  - Nonzero numbers < 1 appear to the right.

Converting Binary to Decimal

Example: Convert 11001.011 to decimal:

$$1 \times 2^4 + 1 \times 2^3 + 1 \times 2^0 + 1 \times 2^{-2} + 1 \times 2^{-3} = 16 + 8 + 1 + \frac{1}{4} + \frac{1}{8} = 25.375$$

Converting Decimal to Binary

1. Find p, the largest power of 2 contained in C.
2. Compute $m = \lfloor C/2^p \rfloor$.
3. Subtract the component: $x = C - m \cdot 2^p$.
4. Repeat until $x = 0$, decreasing p by 1 each time.

Conversion Example Table (Integer)

X	2^p	$\lfloor X/2^p \rfloor$	Remainder	$X - \lfloor X/2^p \rfloor \cdot 2^p$
45	32	1	1	13
13	16	0	0	13
13	8	1	1	5
5	4	1	1	1
1	2	0	0	1
1	1	1	1	0

Example 3.2: Converting 14592.786 **to Binary**

X	2^p	$\lfloor X/2^p \rfloor$	Remainder	$X - \lfloor X/2^p \rfloor \cdot 2^p$
14592.786	8192	1	1	6400.786
6400.786	4096	1	1	2304.786
2304.786	2048	1	1	256.786
256.786	1024	0	0	256.786
256.786	512	0	0	256.786
256.786	256	1	1	0.786
⋮	⋮	⋮	⋮	⋮
0.00075	0.00048828125	1	1	0.00026171875
0.0002617	0.000244140625	1	1	0.000017578125

$$14592.786 \approx 11100100000000.1100100000110_2$$

Project 1: Conversion Script Requirements

- Binary to Decimal:
  - Input: Binary number from user.
  - Output: Decimal equivalent with full precision.
- Decimal to Binary:
  - Input: Decimal number from user.
  - Output: Binary equivalent up to 40–50 digits.
  - Handle non-terminating fractions by limiting loop or checking for repeated patterns.

Decimal To Binary Pseudocode

Algorithm 1 DecimalToBinary(X)

$MaxIters \leftarrow 50$
$W \leftarrow \lfloor X \rfloor$
$D \leftarrow X - W$
$BinW \leftarrow []$
$BinD \leftarrow []$
if $W == 0$ **then**
  $BinW \leftarrow [0]$
else
  while $W \neq 0$ **do**
    Prepend $W \bmod 2$ to $BinW$
    $W \leftarrow W \div 2$
  end while
end if
if $D \neq 0$ **then**
  for $i = 1$ to $MaxIters$ **do**
    $D_2 \leftarrow D \cdot 2$
    Append $\lfloor D_2 \rfloor$ to $BinD$
    $D \leftarrow D_2 - \lfloor D_2 \rfloor$
    if $D == 0$ **then**
      break
    end if
  end for
end if

Binary Conversion Tables

Integer Conversion

X	$X/2$	Remainder	i	2^i	BinW
63	31	1	0	1	[1]
31	15	1	1	2	[1 1]
15	7	1	2	4	[1 1 1]
7	3	1	3	8	[1 1 1 1]
3	1	1	4	16	[1 1 1 1 1]
1	0	1	5	32	[1 1 1 1 1 1]

Fractional Conversion

X	$X \cdot 2$	$\lfloor X \cdot 2 \rfloor$	i	2^{-i}	BinD
0.9921875	1.984375	1	1	1/2	[1]
0.984375	1.96875	1	2	1/4	[1 1]
0.96875	1.9375	1	3	1/8	[1 1 1]
0.9375	1.875	1	4	1/16	[1 1 1 1]
0.875	1.75	1	5	1/32	[1 1 1 1 1]
0.75	1.5	1	6	1/64	[1 1 1 1 1 1]
0.5	1	1	7	1/128	[1 1 1 1 1 1 1]

$$63.9921875_{10} = 111111.1111111_2$$

2.3 Taylor Series

2.3.1 How Computers Store Numbers: The IEEE-754 Standard

For decimal (base-10) numbers, the mantissa is defined as:

$$101.25 = 1.0125 \times 10^2 \quad (\text{mantissa} = 1.0125)$$

For binary (base-2) numbers:

$$1001.11 = 1.00111 \times 2^3 \quad (\text{mantissa} = 1.00111)$$

The mantissa only includes digits after the binary point because the nonzero 1 is always expected to the left of the point.

2.3.2 Recall: Transistors in Computers

- Computers contain transistors, which act as switches.
- Their ON/OFF states represent 1 and 0 bits.
- All data is stored and manipulated as strings of bits.
- Hence, the need for binary floating point representation.

2.3.3 How Computers Use Bits to Represent Numbers

Computers use the following format to represent numbers:

$$\pm 1.mmmmm\ldots \times 2^{ppp}$$

where:

- 1 bit for the sign
- m is the mantissa
- p is the exponent

Precision	Sign Bits	Mantissa Bits	Exponent Bits	Bits / Bytes
Half Precision	1	10	5	$16 = 2$
bfloat16	1	7	8	$16 = 2$
Single Precision	1	23	8	$32 = 4$
Double Precision	1	52	11	$64 = 8$
Quadruple Precision	1	112	15	$128 = 16$

The smallest increment for double precision is approximately 2^{-16}.

2.3.4 Review: Taylor Series

The Taylor Series expansion of a function $f(x)$ about x_0 is:

$$f(x) = f(x_0) + \frac{1}{1!}(x - x_0)\left.\frac{df}{dx}\right|_{x=x_0} + \frac{1}{2!}(x - x_0)^2\left.\frac{d^2f}{dx^2}\right|_{x=x_0} + \cdots$$

Or more compactly:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(c)}{n!}(x - c)^n$$

- Exact analytic solution for $n \to \infty$
- Approximated numerically by truncating after a finite number of terms

2.3.5 Roundoff and Truncation Errors

- **True Error**: Compared to exact (analytic) value
- **Approximate Error**: Compared to a better numerical estimate

Roundoff Errors:

- Due to limited bit representation
- Includes chipping (truncation) or rounding

Truncation Errors:

- Due to using a finite number of terms in a series expansion

Both types of errors can be expressed as absolute or relative errors.

2.3.6 Roundoff Errors

These arise from representing numbers with a finite number of bits:

- **Chipping**: Truncating the number (discarding digits)
- **Rounding Up**: Rounding based on next digit

2.3.7 Chipping and Rounding Up: True Errors

Consider:

$$P(2) = \frac{1}{6} \approx 0.166 \text{ (chipping)}, \quad 0.167 \text{ (rounding up)}$$

Chipping:

$$\text{Absolute Error} = \left|\frac{1}{6} - \frac{166}{1000}\right| = \frac{1}{1500}$$

$$\text{Relative Error} = \frac{\left|\frac{1}{6} - \frac{166}{1000}\right|}{\frac{1}{6}} = 0.00400 = 0.400\%$$

Rounding Up:

$$\text{Absolute Error} = \left|\frac{1}{6} - \frac{167}{1000}\right| = \frac{1}{3000}$$

$$\text{Relative Error} = \frac{\left|\frac{1}{6} - \frac{167}{1000}\right|}{\frac{1}{6}} = 0.00200 = 0.200\%$$

2.3.8 Truncation Errors

Computers only natively support basic operations ($+$, $-$, $\times$, $\div$). For more complex functions:

- Use a Taylor Series expansion
- Truncation introduces an error

2.3.9 Recall: Errors and Acceptability of Numerical Results

- In iterative numerical methods, relative approximate error ϵ_a is computed each iteration
- A user-defined stopping criterion (tolerance) ϵ_s is set
- Stop iterating when:

$$|\epsilon_a| \leq \epsilon_s$$

3 Module 3

3.1 Review on Differentiation

Definition of Derivative

The derivative of a function represents the rate of change of a variable with respect to another variable.

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Example: Tangent Line

Find the slope of the tangent line of the curve $y = 4x^2$ at the point $(3, 36)$:

$$\begin{aligned} &= \lim_{h \to 0} \frac{f(3+h) - f(3)}{h} \\ &= \lim_{h \to 0} \frac{36 + 24h + 4h^2 - 36}{h} \\ &= \lim_{h \to 0} (24 + 4h) \\ &= 24 \end{aligned}$$

What the computer does is take a secant line to approximate a derivative.

A motion detector, for example, working at 1/60th of a second, gets the approximate velocity by calculating the finite difference of position and dividing by the timeframe of the update.

Newton-Raphson Method

A numerical method based on finding the tangent line to a curve at a point.

The tangent line is a straight line of the form:

$$y = mx + b$$

where m is the slope and b is the y-intercept.

Example: Tangent Line from Derivative

Given the function:

$$f(x) = x^3 - 0.165x + 3.993 \times 10^{-4}$$

find the tangent line at $x = 0.05$.

Step 1: Find the slope m:

$$\begin{aligned} f'(x) &= 3x^2 - 0.165 \\ f'(0.05) &= 3(0.05)^2 - 0.165 = -0.1575 \\ m &= -0.1575 \end{aligned}$$

Step 2: Find the function value at $x = 0.05$:

$$f(0.05) \approx -0.0077257$$

Step 3: Use point-slope form to solve for b:

$$-0.0077257 = -0.1575(0.05) + b$$
$$b = 0.0001493$$

Final tangent line:

$$y = -0.1575x + 0.0001493$$

Applications of Derivatives

- Finding minimum and maximum values
- Optimization

Continuous Differentiation

To find a derivative numerically, we approximate with finite difference:

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

We can also rearrange this (heuristically):

$$f(x + \Delta x) \approx f(x) + f'(x)\Delta x$$

Given a value of x and step size Δx, this lets us estimate $f'(x)$ — provided we know $f(x)$.

Difference Approximations

We can estimate derivatives numerically using:

- Forward Difference
- Backward Difference
- Central Difference

Forward Difference

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

Let $\Delta x = x_{i+1} - x_i$, then:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}$$

Backward Difference

$$f'(x) \approx \frac{f(x) - f(x - \Delta x)}{\Delta x}$$

Let $\Delta x = x_i - x_{i-1}$, then:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}}$$

Central Difference

This has error of order $O(\Delta x^2)$:

$$f'(x) \approx \frac{f(x + \Delta x/2) - f(x - \Delta x/2)}{\Delta x}$$

Or, for grid points:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{2\Delta x}$$

This method is commonly used in Runge-Kutta and other numerical methods.

Remark

All of these formulas stem from Taylor series, but the derivations are omitted here.

3.2 Continuous Functions

Finding the Derivative Within a Prespecified Tolerance

In practice:

- We often do not know the true value of the derivative.
- Hence, we cannot compute the true error directly.
- A common approach: start with a step size and keep halving it until the absolute relative approximate error is within a pre-specified tolerance.

Take:

$$v(t) = 2000 \ln\left(\frac{14 \times 10^4}{14 \times 10^4 - 2100t}\right) - 9.8 \quad \text{at } t = 16$$

Using backward difference:

$$v'(t) \approx \frac{v(t) - v(t - \Delta t)}{\Delta t}$$

Finite Difference Approximation of Higher Derivatives

- Forward approximation of second derivative:

$$f''(x) \approx \frac{f(x_{i+2}) - 2f(x_{i+1}) + f(x_i)}{\Delta x^2}$$

Higher Order Accuracy of Higher Derivatives

- Central difference for second derivative:

$$f''(x) \approx \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{\Delta x^2}$$

3.3 Discrete Functions

Discrete Functions

- Discrete functions are defined by data points.
- No governing equation is known.
- Derivatives must be approximated numerically.

Forward Difference Approximation (Discrete)

$$f'(x) \approx \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

- If the desired t value is not available, use the closest forward values.
- For example, if $t = 16$ is desired, and only $t = 15$ and $t = 20$ are known, use $t_i = 15$, $t_{i+1} = 20$.
- This method is reliant on available data.

Direct Fit Polynomials

- Given $n + 1$ data points:
$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$$
- Fit an n-th degree polynomial:
$$P_n(x) = a_0 + a_1 x + \cdots + a_n x^n$$
- First derivative:
$$P_n'(x) = a_1 + 2a_2 x + \cdots + n a_n x^{n-1}$$
- Higher derivatives follow similarly.
- This is a polynomial approximation — not the real function.

The problem: There is no known mathematical equation that describes the relationship between the variables. We only have data points. To determine rates of changes of variables, we must perform approximations.

- FDA, BDA, and CDA are first-order approximation methods.

Matrix Representation of a System of Linear Equations

- n unknowns require n equations to solve.

A system of m equations and n unknowns looks like:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned}$$

This can be written in matrix form as:

$$A\vec{x} = \vec{b}$$

Where A is the coefficient matrix:

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

and $\vec{b}$ is the vector of constants:

$$\vec{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

- The **rank** of a matrix is the number of linearly independent rows or columns.

Lagrange Polynomial

Given data points $(x_0, y_0), \ldots, (x_n, y_n)$, we can construct an n^{th} order Lagrange interpolating polynomial:

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

Where $L_i(x)$ is the Lagrange basis polynomial:

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

- $L_i(x)$ is a weighting function with $(n-1)$ multiplicative terms.
- It omits the $j = i$ term in the product.

Example: Second-Order Lagrange Polynomial

For points (x_0, y_0), (x_1, y_1), and (x_2, y_2), the second-order polynomial is:

$$\begin{aligned} f_2(x) = &\frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) \\ &+ \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \\ &+ \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2) \end{aligned}$$

Sanity Check at a Data Point

Plugging in $x = x_0$:

$$\begin{aligned} f_2(x_0) = &\frac{(x_0 - x_1)(x_0 - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) \\ &+ \frac{(x_0 - x_0)(x_0 - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) \\ &+ \frac{(x_0 - x_0)(x_0 - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2) \\ = &\, f(x_0) \end{aligned}$$

- Similar simplifications will yield $f(x_n)$ when $x = x_n$.
- Very cool: the interpolating polynomial passes exactly through each given point.

4 Module 4: Nonlinear Equations

4.1 Bisection and NR-Methods

Bisection Method

- One of the first numerical methods developed to find the root of a nonlinear equation $f(x) = 0$.
- Also called the binary-search method.

Basis of Bisection Method

Theorem: An equation $f(x) = 0$, where $f(x)$ is a real continuous function, has at least one root between x_l and x_u if

$$f(x_l)f(x_u) < 0$$

- At least one root exists between the two points if the function is real, continuous, and changes sign.
- If the function does not change sign between x_l and x_u, roots *may* still exist in the interval.
- If $f(x_l)f(x_u) > 0$, there may or may not be a root between x_l and x_u.
- If $f(x_l)f(x_u) < 0$, there may be more than one root between x_l and x_u.

The method falls under the category of **bracketing methods**, as it finds a root between two endpoints.

One finds the midpoint x_m between x_l and x_u, generating two subintervals:

1. $[x_l, x_m]$
2. $[x_m, x_u]$

Steps of the Bisection Method

1. Choose x_l and x_u such that $f(x_l)f(x_u) < 0$ (sign change between them).
2. Estimate the root x_m as the midpoint:

$$x_m = \frac{x_l + x_u}{2}$$

3. Evaluate the sign of $f(x_m)$:
   - If $f(x_l)f(x_m) < 0$, then the root lies in $[x_l, x_m]$. Set $x_u = x_m$.
   - If $f(x_l)f(x_m) > 0$, then the root lies in $[x_m, x_u]$. Set $x_l = x_m$.
   - If $f(x_m) = 0$, then x_m is the exact root. Stop the algorithm.

4. Update the estimate of the root:

$$x_m = \frac{x_l + x_u}{2}$$

   Compute the absolute relative approximate error:

$$|\varepsilon_a| = \left| \frac{x_m^{\text{new}} - x_m^{\text{old}}}{x_m^{\text{new}}} \right| \times 100$$

   - x_m^{old} = previous estimate
   - x_m^{new} = current estimate

5. Compare $|\varepsilon_a|$ with the pre-specified tolerance ε_s:
   - If $|\varepsilon_a| > \varepsilon_s$, repeat from Step 2.
   - Else, stop the algorithm.

6. Check maximum number of iterations:
   - If exceeded, stop and notify the user.

Advantages of the Bisection Method

- Always convergent.
- Easy to implement.

Drawbacks of the Bisection Method

- Slow to converge.
- Cannot find root if the function does not change sign.
- Even if sign changes, a root may not exist (e.g., discontinuity or cusp).

Newton-Raphson Method

The Newton-Raphson method is an open method to find the root of the equation $f(x) = 0$. It uses the first-order Taylor series expansion around a point.

Formula:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Steps:

- Choose an initial guess x_0.
- Compute the next approximation using the formula.
- Repeat until the approximate error is less than the desired tolerance.
- The approximate relative error is

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100$$

Requirements:

- The function $f(x)$ must be differentiable.
- Derivative $f'(x)$ must not be zero near the root.

Advantages:

- Rapid convergence (quadratic).
- Fewer iterations needed if the guess is close to the root.

Drawbacks:

- Requires computation of $f'(x)$.
- May diverge if the guess is poor or if $f'(x)$ is zero or near zero.

4.2 Secant and Iterative Methods

The Secant method is another open method to find the root of $f(x) = 0$. Unlike Newton-Raphson, it does not require the derivative of $f(x)$.

Formula:

$$x_{i+1} = x_i - f(x_i) \cdot \frac{x_i - x_{i-1}}{f(x_i) - f(x_{i-1})}$$

Steps:

- Choose two initial guesses x_0 and x_1.
- Compute x_2 using the formula above.
- Repeat the process, updating the last two guesses each time.
- Stop when the approximate relative error is less than the desired tolerance:

$$\varepsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| \times 100$$

Advantages:

- Faster than bisection.
- Does not require derivative computation.

Drawbacks:

- Not guaranteed to converge.
- Can be slower than Newton-Raphson.

5 Module 6

5.1 Introduction to Matrix Algebra and Solving Simultaneous Linear Equations

Matrix algebra is a branch of mathematics used for solving systems of linear equations. A **matrix** is a rectangular array of elements, which can be numbers or symbolic expressions, arranged in rows and columns. Matrices provide a clear way to present numerical data and simplify the programming of complex calculations. The order of elements is crucial in matrix algebra.

A general $m \times n$ matrix A, which has m rows and n columns, is written as:

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

The element in the i-th row and j-th column is denoted by a_{ij}.

- **Row** i of A is given by $[a_{i1}, a_{i2}, \ldots, a_{in}]$.
- **Column** j of A is given by $\begin{bmatrix} a_{1j} \\ a_{2j} \\ \vdots \\ a_{mj} \end{bmatrix}$.

A **vector** is a special type of matrix that has only one row (a row vector) or one column (a column vector). A **submatrix** is a matrix formed by deleting some rows or columns from a larger matrix.

Special Types of Matrices

Square Matrix

A matrix where the number of rows equals the number of columns ($m = n$). The entries $a_{11}, a_{22}, \ldots, a_{nn}$ form the **main diagonal**. Non-square matrices also have diagonal entries, which are the elements a_{ii} for $i = 1, 2, \ldots, \min(m, n)$.

Triangular Matrices

An **Upper Triangular Matrix** is a square matrix where all elements below the main diagonal are zero ($a_{ij} = 0$ for $i > j$). A **Lower Triangular Matrix** is a square matrix where all elements above the main diagonal are zero ($a_{ij} = 0$ for $i < j$).

Diagonal Matrix

A square matrix where all non-diagonal elements are zero ($a_{ij} = 0$ for $i \neq j$).

Identity Matrix (I)

A diagonal matrix where all diagonal elements are equal to 1. It is the matrix equivalent of the number 1.

Zero Matrix

A matrix where all elements are zero.

Tridiagonal Matrix

A square matrix where all elements are zero except for those on the main diagonal, the diagonal directly above it, and the diagonal directly below it.

Diagonally Dominant Matrix

A square matrix where for each row, the absolute value of the diagonal element is greater than or equal to the sum of the absolute values of all other elements in that row. For at least one row, the inequality must be strict.

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}| \quad \text{for all } i$$

Vectors

A vector is a collection of numbers arranged in a definite order.

Vector Equality and Operations

- **Vector Equality:** Two vectors a and b are equal if they have the same dimension and their corresponding components are equal ($a_i = b_i$ for all i).
- **Vector Addition:** Two vectors can be added only if they have the same dimension. Addition is performed by adding corresponding components.
- **Scalar Multiplication:** Multiplying a vector by a scalar involves multiplying each component of the vector by that scalar.

Vector Types and Properties

- **Zero Vector:** A vector where all components are zero.
- **Unit Vector:** A vector with a norm (or magnitude) of 1. For a vector $u = [u_1, u_2, \ldots, u_n]$, this means $\sqrt{u_1^2 + u_2^2 + \cdots + u_n^2} = 1$.
- **Linear Combination:** Given vectors $A_1, \ldots, A_m$ and scalars $k_1, \ldots, k_m$, the expression $k_1 A_1 + \cdots + k_m A_m$ is a linear combination.

- **Linear Independence:** A set of vectors is linearly independent if the only solution to their linear combination equaling the zero vector is for all scalars to be zero. Otherwise, they are linearly dependent.
- **Rank:** The rank of a set of vectors is the maximum number of linearly independent vectors in the set.

Binary Matrix Operations

Addition, Subtraction, and Scalar Multiplication

Two matrices can be added or subtracted only if they have the same size. The operation is element-wise: if $C = A \pm B$, then $c_{ij} = a_{ij} \pm b_{ij}$. Scalar multiplication involves multiplying every element by the scalar.

Matrix Multiplication

The product $C = AB$ is defined only if the number of columns in A (an $m \times p$ matrix) equals the number of rows in B (a $p \times n$ matrix). The result C is an $m \times n$ matrix. The element c_{ij} is the dot product of the i-th row of A and the j-th column of B:

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj}$$

Rules of Binary Operations

- Commutative Law of Addition: $A + B = B + A$
- Associative Law of Addition: $(A + B) + C = A + (B + C)$
- Associative Law of Multiplication: $A(BC) = (AB)C$
- Distributive Law: $A(C + D) = AC + AD$
- In general, matrix multiplication is not commutative: $AB \neq BA$.

Unary Matrix Operations

Transpose (A^T)

The transpose of a matrix A is obtained by interchanging its rows and columns. If $A = [a_{ij}]$, then $A^T = [a_{ji}]$.

Symmetric and Skew-Symmetric Matrices

A **symmetric matrix** satisfies $A = A^T$. A **skew-symmetric matrix** satisfies $A = -A^T$.

Trace ($\mathbf{tr}(A)$)

The sum of the diagonal elements of a square matrix. Note that $\text{tr}(A) = \text{tr}(A^T)$.

Determinant

The determinant of a square matrix, denoted $\det(A)$ or $|A|$, is a scalar value representing the scaling factor of the linear transformation described by the matrix.

Determinant of a 2×2 Matrix

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

Determinant by Cofactor Expansion

The determinant can be found by expanding along any row or column. The cofactor C_{ij} is $C_{ij} = (-1)^{i+j} M_{ij}$, where M_{ij} is the determinant of the submatrix formed by deleting row i and column j. For a 3×3 matrix, expanding along the first row gives:

$$|A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$|A| = a(ei - fh) - b(di - fg) + c(dh - eg)$$

Properties of Determinants

- $\det(A^T) = \det(A)$
- $\det(I) = 1$
- Swapping two rows of a matrix negates its determinant.

5.2 Gaussian Elimination

Representing Systems of Equations

A system of two linear equations with two variables,

$$\begin{aligned} Ax + By &= C \\ Dx + Ey &= F \end{aligned}$$

can be represented in matrix form as:

$$\begin{bmatrix} A & B \\ D & E \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} C \\ F \end{bmatrix}$$

Solving Systems of Equations

Sets of simultaneous linear equations arise in various scientific and engineering problems, often from models based on:

- A set of nonlinear equations
- Polynomial interpolation
- Numerical integration
- Discretization of differential equations

Example: Polynomial Interpolation

Assume the velocity of an object, $v(t)$, follows the quadratic model $v(t) = at^2 + bt + c$. Given the following data points:

Time (t)	Velocity (v)
5	106.8
8	177.2
12	279.2

Substituting these values into the model yields a system of three linear equations for the coefficients a, b, c:

$$\begin{aligned} a(5^2) + b(5) + c &= 106.8 \\ a(8^2) + b(8) + c &= 177.2 \\ a(12^2) + b(12) + c &= 279.2 \end{aligned}$$

This system can be written in matrix form as:

$$\begin{bmatrix} 25 & 5 & 1 \\ 64 & 8 & 1 \\ 144 & 12 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 106.8 \\ 177.2 \\ 279.2 \end{bmatrix}$$

Nature of Solutions

Consistent and Inconsistent Systems

A system of equations is **consistent** if at least one solution exists. It is **inconsistent** if no solution exists. A consistent system can have either a unique solution or infinitely many solutions.

The consistency of a system $AX = C$ can be determined by comparing the rank of the coefficient matrix A with the rank of the augmented matrix $[A|C]$.

- **Consistent:** The system is consistent if $\text{rank}(A) = \text{rank}([A|C])$.
- **Inconsistent:** The system is inconsistent if $\text{rank}(A) \neq \text{rank}([A|C])$.

Rank of a Matrix

The **rank** of a matrix can be defined in several ways:

- The number of linearly independent column (or row) vectors of the matrix.
- The order (size) of the largest square submatrix whose determinant is not zero.

Unique and Infinite Solutions for Consistent Systems

For a consistent system with n unknowns:

- A **unique solution** exists if the rank of the coefficient matrix is equal to the number of unknowns ($\text{rank}(A) = n$).
- **Infinite solutions** exist if the rank of the coefficient matrix is less than the number of unknowns ($\text{rank}(A) < n$).

Methods for Solving Linear Systems

Solving with the Inverse Matrix

If A is a square matrix and its inverse A^{-1} exists, A is called **invertible** or **nonsingular**. The inverse satisfies the property:

$$AA^{-1} = A^{-1}A = I$$

where I is the identity matrix. If A^{-1} does not exist, A is called **singular** or **noninvertible**.

For a system $AX = C$ where A is invertible, the solution can be found by:

$$A^{-1}(AX) = A^{-1}C \implies (A^{-1}A)X = A^{-1}C \implies IX = A^{-1}C \implies X = A^{-1}C$$

Gaussian Elimination

Gaussian elimination is a systematic procedure for solving systems of linear equations by transforming the augmented matrix into a simpler form. The method uses **elementary row operations**:

1. Swapping two rows.
2. Multiplying a row by a non-zero constant.
3. Adding a multiple of one row to another row.

The process consists of two main stages:

- **Forward Elimination:** Applying row operations to transform the coefficient matrix into an **upper triangular matrix**.
- **Back Substitution:** Solving for the variables starting from the last equation and working backwards.

This process can be continued to transform the matrix into **Reduced Row Echelon Form (RREF)**, where the solution can be read directly.

5.3 Gauss-Seidel

5.4 LU Decomposition

5.5 Cholesky Decomposition and LDLT Method