الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University Akli Mohand Oulhadj -Bouira-
Faculty of Sciences and Applied Sciences
وزارة التعليم العالي والبحث العلمي
جـــامعة أكلي محند أولـــحاج -البويرة-
كلية العلوم والعلوم التطبيقية
جامعة البويرة

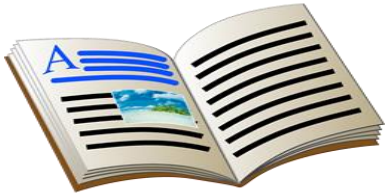
Department of Civil Engineering

PW Handout

In Civil Engineering

Specialty: Civil Engineering / Hydraulics

Level: License



Practical Works in Numerical Methods

By Oussama Ounissi

Appraised by:

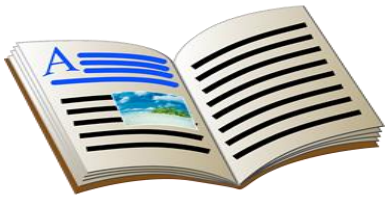
- Dr.Djafer Khodja Hakim, University of Bouira
- Dr.Ferhati Ahmed, University of M'sila

Year: 2023/2024

الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University Akli Mohand Oulhadj -Bouira-
Faculty of Sciences and Applied Sciences
وزارة التعليم العالي والبحث العلمي
جـــامعة أكلي محند أولـــحاج -البويرة-
كلية العلوم والعلوم التطبيقية
جامعة البويرة

Département de Génie Civil

Polycopié de TP



En : Génie Civil

Spécialité : Génie Civil / Hydraulique

Niveau : License

TP – Méthodes Numériques

Par Ounissi Oussama

Expertisé par :

- Dr.Djafer Khodja Hakim, Université de Bouira
- Dr.Ferhati Ahmed, Université de M'sila

Année : 2023/2024

Table of Contents

Preface

This handout provides coverage of numerical methods used to solve a variety of mathematical and modeling problems in engineering and science. The different chapters include theoretical explanations, practical examples with detailed solutions, and MATLAB codes, providing students with the necessary tools to effectively apply these methods in their own work.

Module Identification Sheet

Semester:	S4
Teaching Unit:	UEM 2.2
Subject:	Numerical Methods Lab
Weekly Workload:	22h30 (Lab: 1h30)
Credits:	2
Coefficient:	1

Teaching Objectives:

Programming of various numerical methods for their application in mathematical calculations using a scientific programming language.

Recommended Prerequisite Knowledge:

- Numerical Methods
- Computer Science 2
- Computer Science 3

Evaluation Method:

Continuous assessment: 100%.

Course Content:

Chapter 1: Solving Nonlinear Equations (3 weeks)

(Bisection Method, Fixed Points, Newton-Raphson Method)

Chapter 2: Interpolation and Approximation (3 weeks)

(Newton Interpolation, Chebyshev Approximation)

Chapter 3: Numerical Integration (3 weeks)

(Rectangle Method, Trapezoidal Rule, Simpson's Rule)

Chapter 4: Differential Equations (2 weeks)

(Euler's Method, Runge-Kutta Methods)

Chapter 5: Systems of Linear Equations (4 weeks)

(Gauss-Jordan Method, Crout Decomposition and LU Factorization, Jacobi Method, Gauss-Seidel Method)

Introduction

Numerical methods are essential mathematical techniques used across various disciplines to approximate solutions to complex problems when exact analytical solutions are impractical or unavailable. These methods encompass a diverse range of algorithms and computational tools, including root finding, interpolation, numerical integration, differential equation solving, and linear algebra techniques. Challenges such as balancing accuracy and efficiency, ensuring stability and convergence, and addressing discretization errors are inherent in numerical computations. Despite these challenges, numerical methods empower scientists, engineers, and analysts to simulate real-world phenomena, optimize designs, and make informed decisions by providing efficient and reliable approximate solutions to mathematical problems.

Chapter I:

Nonlinear Equation Solving

There are several numerical technics which can be used to locate the roots of nonlinear equations, particularly for those that don't have a straightforward analytical solution. The following methods are the most commonly used.

I.1. Worksheet 1: Bisection Method

I.1.1. Concept

The bisection method operates by progressively reducing the range within which the root is located. It relies on the Theorem of the Intermediate Value. This theorem asserts that a continuous function f(x) on a closed interval [a, b], will have at least one root within this interval if f(a) and f(b) have different signs.

I.1.2. Flowchart

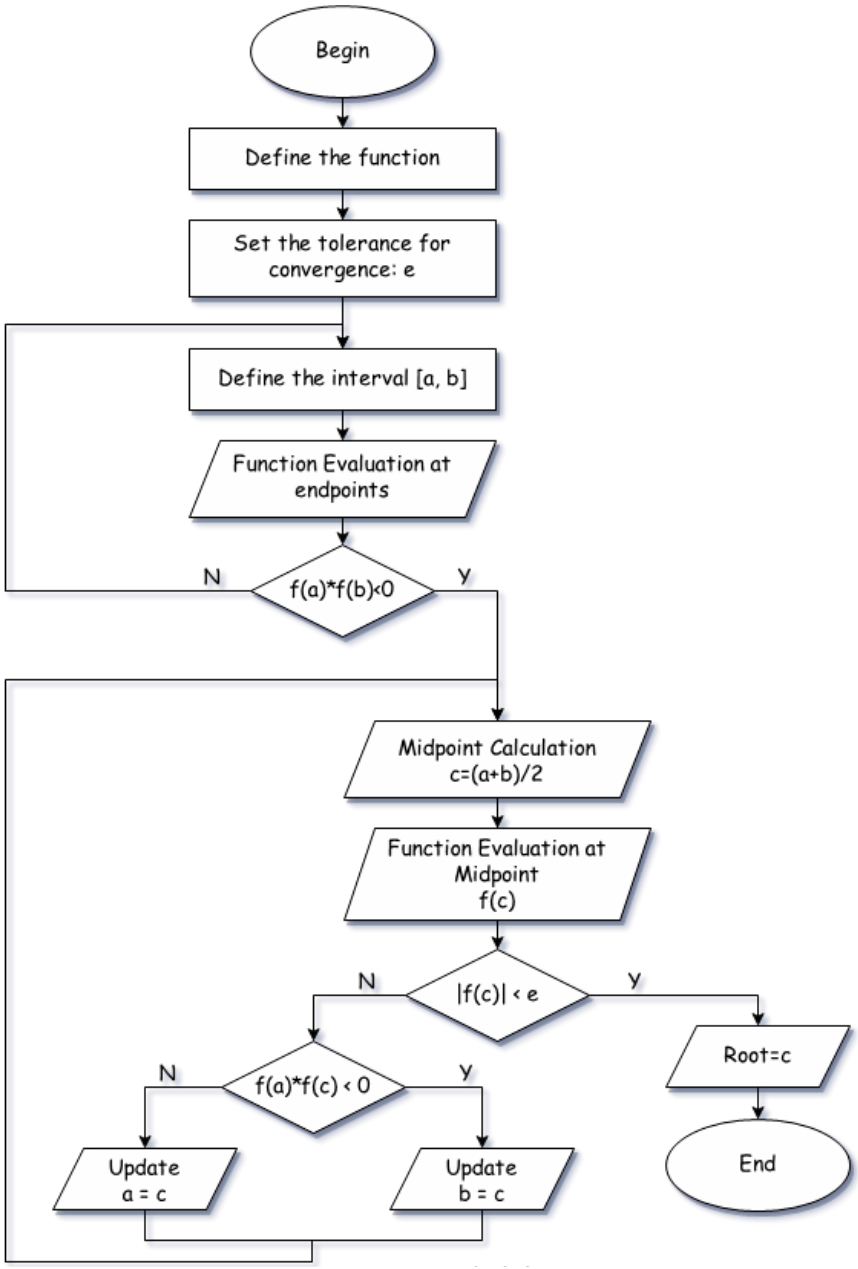


Figure 1: Flowchart of the bisection method

I.1.3. Example

Apply the algorithm:

1. Define the function
   Let's consider solving the following equation: $\boldsymbol{f(x) = x^2 - 4}$
2. Define the tolerance
   Let's consider $\boldsymbol{\varepsilon = 0.3}$
3. Define the initial Interval $[\boldsymbol{a, b}]$
   Let's choose two initial points $\boldsymbol{a = 0}$ and $\boldsymbol{b = 3}$
4. Function evaluation at the endpoints: accept the interval $[a, b]$where the function $f(x)$ has opposite signs at the endpoints.
   $f(a) = f(0) = -4$ (negative)
   $f(b) = f(3) = 5$ (positive)
   They have an opposite sign, so we can continue with these values.
5. Midpoint Calculation: Calculate the midpoint of the interval.
   The midpoint $\boldsymbol{c = \frac{a+b}{2} = \frac{3}{2}}$
6. Function Evaluation at Midpoint: Evaluate the function at the midpoint.
   Evaluate $f(c)$: $\boldsymbol{f(c) = f\left(\frac{3}{2}\right) = \left(\frac{3}{2}\right)^2 - 4 = -\frac{7}{4}}$ (negative).
7. Root Check: $\boldsymbol{f(c) = -\frac{7}{4}}$
   it is not very close to zero ($|f(c)| > \varepsilon$) and it has the same sign as f(a), then the root falls within the interval [b, c].
8. Update the interval a = c.
9. Repeat: Go back to step 5 and repeat the process with the new interval until the desired level of accuracy is achieved for the root approximation.

The Bisection technic converges to the solution of the equation by continuously reducing the size of the interval where the root is located. The evolution of the various parameters of the algorithm is shown in the following table

The table 1 represents the progression of the algorithm for this example.

Table 1: Evolution of the various parameters of the bisection algorithm.

Itr	*a*	*b*	*f(a)*	*f(b)*	*c*	*f(c)*	*|f(c)|*	*Note*
1	0	3	-4	5	1.5	-1.75	>ε	We replace a by c
2	1.5	3	-1.75	5	2.25	1.0625	>ε	We replace b by c
3	1.5	2.25	-1.75	1.0625	1.875	-0.48438	>ε	We replace a by c
4	1.875	2.25	-0.48438	1.0625	**2.0625**	0.25391	<ε	We get the result

I.1.4. MATLAB code

```
    clear all;          close all;          clc;
    disp(['**** Non-Linear Equations Solving with Bisection Method ****'])

    disp(['Please define the tolerance :']);
    Eps=input('Eps =');

    disp(['Please define the initial Interval  [a b] :']);
    a=input('a =');    b=input('b =');

    Fnc= @(x) x.^2 - 4;

    hold on;      fplot(Fnc,[a b]);       fplot(@(x) 0*x,[a b])

    if Fnc(a)*Fnc(b)>0
        disp(['========= There is no roots in this interval =========']);
        disp(['Please redefine the initial Interval  [a b] :']);
        a=input('a =');
        b=input('b =');
    else
        itr= 1;
        c = (a+b)/2;
        while abs(Fnc(c)) > Eps
            disp(['------------ iteration ',num2str(itr),' ------------'])

disp(['c=',num2str(c),',f(c)=',num2str(Fnc(c)),'>Eps(=',num2str(Eps),')']);

            if Fnc(c)*Fnc(a) <=0
                b = c;
                disp('b and c have the same signs, so we replace b by c')
            else
                a = c;
                disp('a and c have the same signs, so we replace a by c')
            end
            c = (a+b)/2;
            itr=itr+1;
        end
disp(['------------ iteration ',num2str(itr),' ------------']);
disp(['c=',num2str(c),',f(c)=',num2str(Fnc(c)),'<Eps(=',num2str(Eps),')']);
disp(['The root of this function in the given interval X=',num2str(c)]);
    end
    plot(c,Fnc(c),'ko');
```

I.2. Worksheet 2: Newton-Raphson Method

I.2.1. Concept

Newton's method, also known as the Newton-Raphson method, functions by progressively refining an initial estimate to approach the true root with each iteration. This method uses the idea of tangents. At each step, it takes an initial guess (x-value) and draws a tangent line to the function at that point. The point where the tangent line intersects the x-axis becomes a better approximation for the root. the process will be repeated with this new approximation, getting an even closer estimate.

I.2.2. Flowchart

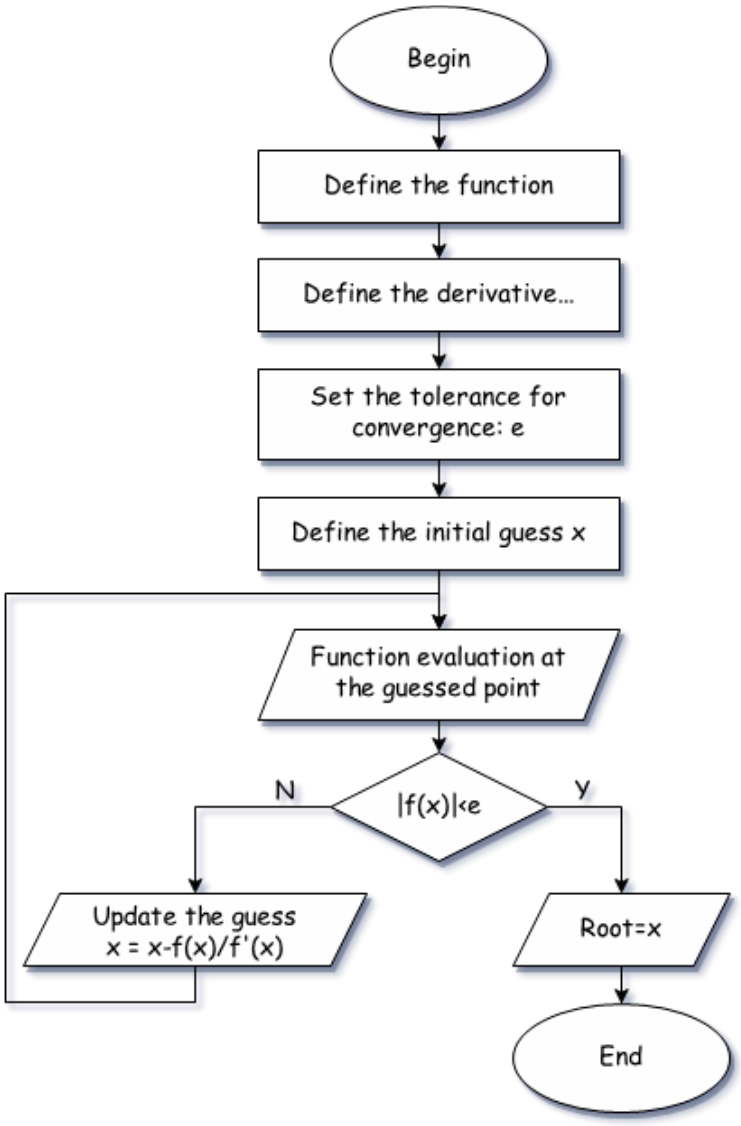


Figure 2: Flowchart of the bisection method

I.2.3. Example

Apply the algorithm to the previous function:

1. Define the function

Let's consider solving the following equation:

$\boldsymbol{f(x) = x^2 - 4}$

2. Define the function's derivative

The derivative of the previous function is:

$\boldsymbol{\dot{f}(x) = 2x}$

3. Define the tolerance

Let's consider: $\boldsymbol{\varepsilon = 0.3}$

4. Define the initial guess for the root:

Let's choose two initial points: $\boldsymbol{x_0 = 9}$

5. Function evaluation at the guessed root:

$\boldsymbol{f(x) = 9^2 - 4 = 77}$

6. Root Check: $\boldsymbol{f(x) = 77,}$

it is not very close to zero ($|f(x)| > \varepsilon$)

7. Establish the iterative formula to update the guess:

$\boldsymbol{x_{n+1} = x_n - \frac{f(x_n)}{\dot{f}(x_n)}}$

8. Repeat: consider the newly obtained x_1 as x_0 and continue iterating from step 5.

The table 2 represents the progression of the algorithm for this example.

Table 2: Evolution of the various parameters of the Newton's algorithm.

Itr	x	f(x)	|f(x)|	
1	9	77	**>ε**	update the guess
2	4.7222	18.2994	**>ε**	update the guess
3	2.7846	3.7542	**>ε**	update the guess
4	2.1105	0.4544	**>ε**	update the guess
5	**2.0029**	0.011589	**<ε**	We get the result

I.2.4. MATLAB code

```
    clear all;          close all;          clc;
    disp(['**** Non-Linear Equations Solving with Bisection Method ****'])
    disp(['Please define the tolerance :']);       Eps=input('Eps =');
    disp(['Please define the initial Interval  [a b] :']);
    a=input('a =');
    b=input('b =');

    Fnc= @(x) x.^2 - 4;

    hold on
    fplot(Fnc,[a b])
    fplot(@(x) 0*x,[a b])

    if Fnc(a)*Fnc(b)>0
        disp(['========= There is no roots in this interval =========']);
        disp(['Please redefine the initial Interval  [a b] :']);
        a=input('a =');    b=input('b =');
    else
        itr= 1;
        c = (a+b)/2;
        while abs(Fnc(c)) > Eps
            disp(['------------ iteration ',num2str(itr),' ------------'])
disp(['c=',num2str(c),',f(c)=',num2str(Fnc(c)),'>Eps(=',num2str(Eps),')']);
            if Fnc(c)*Fnc(a) <=0
                b = c;
                disp('b and c have the same signs, so we replace b by c')
            else
                a = c;
                disp('a and c have the same signs, so we replace a by c')
            end
            c = (a+b)/2;
            itr=itr+1;
        end

disp(['------------ iteration ',num2str(itr),' ------------']);
disp(['c=',num2str(c),',f(c)=',num2str(Fnc(c)),'<Eps(=',num2str(Eps),')']);
disp(['The root of this function in the given interval X=',num2str(c)]);
    end
    plot(c,Fnc(c),'ko');
```

Chapter II:

Approximation and Interpolation

Approximation and interpolation are powerful tools in numerical analysis, each serving different purposes. Approximation simplifies complex functions, while interpolation estimates values within the range of known data. Both techniques are widely used in various fields such as data science, engineering, and computer graphics.

II.1. Worksheet 1: Newton-Interpolation

II.1.1. Concept

Newton's interpolation is a technique in numerical analysis for constructing an interpolating polynomial particularly useful for approximating the values of a function between known data points.

II.1.2. Algorithm

1. Given Data Points:

Consider a set of data points

2. Newton's Divided Difference Table:

We first compute the divided difference table, which involves calculating the divided differences recursively. The divided differences are used to construct the coefficients of the interpolating polynomial.

$$\mathrm{D} = \begin{bmatrix} f(x_1) & \square & \square & \cdots & \square \\ f(x_2) & f[x_1, x_2] & \square & \ldots & \square \\ f(x_3) & f[x_2, x_3] & f[x_1, x_3] & \ldots & \square \\ \vdots & \vdots & \vdots & \ddots & \square \\ f(x_i) & f[x_{i-1}, x_i] & f[x_{i-2}, x_i] & \ldots & f[x_1, x_i] \end{bmatrix}$$

where $f[x_i]$ represents the divided differences and $f[x_i, x_{i+k}]$ represents higher-order divided differences and denoted by:

$$f[x_i] = y_i$$

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

$$f[x_i, x_{i+2}] = \frac{\frac{f[x_{i+2}] - f[x_{i+1}]}{x_{i+2} - x_{i+1}} - \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}}{x_{i+2} - x_i}$$

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

3. Construct the Interpolating Polynomial:

After having the divided difference table, the interpolating polynomial will be built using Newton's interpolation formula:

$$P(x) = f[x_1] + (x - x_1) f[x_1, x_2]$$
$$+ (x - x_1)(x - x_2) f[x_1, x_3] + \cdots$$
$$+ (x - x_1)(x - x_2) \ldots (x - x_{n-1}) f[x_1, x_n]$$

II.1.3. Example

Apply the previous algorithm:

1. Data Input:

Given a set of data points $((x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$,

where x_i are distinct.

x	y
1	2
2	4
3	13

2. Calculate and fill the divided differences table named D:

$$D = \begin{bmatrix} f(x_1) & \square & \square \\ f(x_2) & f[x_1, x_2] & \square \\ f(x_3) & f[x_2, x_3] & f[x_1, x_3] \end{bmatrix} = \begin{bmatrix} 2 & \square & \square \\ 4 & 2 & \square \\ 13 & 9 & \frac{7}{2} \end{bmatrix}$$

3. Construct the interpolation polynomial in Newton's form:

$$P(x) = 2 + 2(x - x_1) + \frac{7}{2}(x - x_1)(x - x_2)$$

$$= 2 + 2(x - 1) + \frac{7}{2}(x - 1)(x - 2)$$

$$= \frac{7}{2}x^2 - \frac{21}{2}x + 9$$

II.1.4. MATLAB code

```
clear all;          close all;          clc;
x = [1, 2, 3];
y = [2, 4, 13];

% Calculate the number of data points
n = length(x);

% Initializing the table of divided differences
D = zeros(n, n);

% Fill the first column of the table with function values
D(1:n, 1) = y;

% Calculate higher order divided differences
for i = 2:n
    for j = 2:i
        D(i, j) = (D(i, j-1) - D(i-1, j-1)) / (x(i) - x(i-j+1));
    end
end

% Build the interpolation polynomial
syms X;
for i = 2:n
    % Construction du polynôme d'interpolation
    P = D(1, 1)+D(i, i) * prod(X - x(1:i-1));
End

% Simplification du polynôme
P = expand(P);
disp('Le polynôme d''interpolation de Newton est :');
pretty(P)

% Tracé du polynôme
fplot(P, [min(x), max(x)]);
```

II.2. Worksheet 2: Chebyshev Approximation

II.2.1. Concept

Chebyshev approximation is a method for polynomial approximation using orthogonal Chebyshev polynomials which are particularly well-suited for minimizing oscillations and achieving uniform approximation over a given interval.

II.2.2. Algorithm

Here's a step-by-step algorithm:

1. Define the Function to Approximate

2. Choose the Interval:

Determine the interval over which you want to approximate the function.

3. Choose the Number of Terms:

It defines the Degree of Approximation

4. Compute Chebyshev Nodes:

Calculate the n Chebyshev nodes within the chosen interval. These nodes are given by the formula:

$$x_i = \cos\left(\frac{(2i+1)\pi}{2n}\right) \qquad \text{where } i = 1, 2, \ldots, n.$$

5. Evaluate the Function at Chebyshev Nodes:

Evaluate the function to be approximated at the Chebyshev nodes determined in the previous step to obtain $f(x_i)$.

6. Compute Chebyshev Polynomials up to the chosen degree using the recurrence relation:

$T_0(x) = 1, \qquad T_1(x) = x, \qquad T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x)$

$T_k(x) = \cos(k\, arccos(x))$

7. Compute the Chebyshev Coefficients: Compute the Chebyshev coefficients c_k using the formula:

$$c_k = \frac{2}{N}\sum_{i=1}^{n} f(x_i) \cdot T_k(x_i)$$

8. Approximate the Function: Reconstruct the function $P_n(x)$ using the Chebyshev series expansion:

$$P_n(x) \approx \sum_{k=0}^{n-1} c_k T_k(x)$$

II.2.3. Example

Apply the previous algorithm:

1. Define the Function to Approximate: lets choose $f(x) = sin(x)$
2. Define the interval over which the function will be approximated: $[-1,1]$
3. Choose the Number of Terms (Degree of Approximation): n=5
4. Compute Chebyshev Nodes x_i:

$$x_1 = \cos\left(\frac{\pi}{10}\right) \approx 0.9511$$
$$x_2 = \cos\left(\frac{3\pi}{10}\right) \approx 0.5878$$
$$x_3 = \cos\left(\frac{5\pi}{10}\right) = 0$$
$$x_4 = \cos\left(\frac{7\pi}{10}\right) \approx -0.5878$$
$$x_5 = \cos\left(\frac{9\pi}{10}\right) \approx -0.9511$$

5. Evaluate the Function at Chebyshev Nodes determined in the previous step to obtain $f(x_i)$.

$$f(x_1) = sin(0.9511) \approx 0.8150$$
$$f(x_2) = sin(0.5878) \approx 0.5542$$
$$f(x_3) = sin(0) = 0$$
$$f(x_4) = sin(-0.5878) \approx -0.5542$$
$$f(x_5) = sin(-0.9511) \approx -0.8150$$

6. Compute Chebyshev Polynomials up to the chosen degree using the recurrence relation:

k	$T_k(x)$	x_1	x_2	x_3	x_4	x_5
		0.9511	**0.5878**	**0**	**−0.5878**	**−0.9511**
0	**1**	1	1	1	1	1
1	x	0.9511	0.5878	0	−0.5878	−0.9511
2	$2x^2 - 1$	0.809	−0.3072	−1	−0.3072	0.809
3	$3x^3 - 3x$	0.309	−0.7555	0	0.7555	−0.309
4	$8x^4 - 8x^2 + 1$	−0.6910	−0.7860	1	−0.7860	−0.6910

7. Compute the Chebyshev Coefficients c_k:

$k = 0$:

$c_0 = \frac{1}{5}\sum_{i=1}^{5} f(x_i)$

$c_0 = \frac{1}{5}(0.8150 + 0.5542 + 0 - 0.5542 - 0.8150) = 0$

$k = 1$:

$c_1 = \frac{2}{5}\sum_{i=1}^{n} f(x_i)T_1(x_i)$

$c_1 = \frac{2}{5}\big((0.8150)(0.9511) + (0.5542)(0.5878) + (0)(0) + (-0.5542)(-0.5878) + (-0.8150)(-0.9511)\big) \approx 0.88$

$k = 2$:

$c_2 = \frac{2}{5}\sum_{i=1}^{n} f(x_i)T_2(x_i)$

$c_2 = \frac{2}{5}\big((0.8150)(0.809) + (0.5542)(-0.3072) + (0)(-1) + (-0.5542)(-0.3072) + (-0.8150)(0.809)\big) = 0$

$k = 3$:

$c_3 = \frac{2}{5}\sum_{i=1}^{n} f(x_i)T_3(x_i)$

$c_3 = \frac{2}{5}\big((0.8150)(0.309) + (0.5542)(-0.7555) + (0)(0) + (-0.5542)(0.7555) + (-0.8150)(-0.309)\big) = 0$

$k = 4$:

$c_4 = \frac{2}{5}\sum_{i=1}^{n} f(x_i)T_4(x_i)$

$c_4 = \frac{2}{5}\big((0.8150)(-0.6910) + (0.5542)(-0.7860) + (0)(1) + (-0.5542)(-0.7860) + (-0.8150)(-0.6910)\big) = 0$

8. Reconstruction of the function $P(x)$ using the Chebyshev series expansion:

$P_5(x) = \frac{1}{2}c_0 + c_1T_1(x) + c_2T_2(x) + c_3T_3(x) + c_4T_4(x)$

$P_5(x) = 0.88x$

II.2.4. MATLAB code

```
clear;          close;          clc
% Define the function to approximate
f = @(x) sin(x);

% Define the interval
a = -1;
b = 1;

% Degree of the Chebyshev polynomial
n = 5;

% Generate the Chebyshev nodes
x_nodes = cos((2*(0:n)+1)*pi/(2*(n+1)));

% Evaluate the function at the Chebyshev nodes
y_nodes = f(x_nodes);

% Compute the Chebyshev coefficients
cf = zeros(1, n+1);
for k = 0:n
    for j = 0:n
      cf(k+1)=cf(k+1)+y_nodes(j+1)*cos(k*acos(cos(pi*(j+0.5)/(n+1))));
    end
    cf(k+1) = (2/(n+1)) * cf(k+1);
end

% Define the Chebyshev polynomial approximation
ChApprox = @(x) ChFunc(x, cf);

% Plot the original function and the approximation
x_plot = linspace(a, b, 1000);

figure;
plot(x_plot, f(x_plot), 'b-', x_plot, ChApprox(x_plot), 'r--');

legend('sin(x)', 'Chebyshev Approximation');
title('Approx of sin(x) using a 5th-degree Chebyshev polynomial');

xlabel('x');
ylabel('y');

% evaluation of Chebyshev polynomial
function y = ChFunc(x, coeff)
    n = length(cf) - 1;
    y = zeros(size(x));
    for k = n:-1:0
        y = cf(k+1) + 2*x.*y - y;
    end
end
```

Chapter III:

Numerical Integration

Numerical integration is a fundamental aspect of numerical analysis, providing techniques to approximate the definite integral of a function when an analytical solution is difficult or impossible to obtain. Let's explore the basics, methods, and applications of numerical integration.

III.1. Worksheet 1: Rectangle Rule

III.1.1. Concept

The rectangle method, alternatively referred to as the midpoint rule, is a numerical integration technique that estimates the integral of a function by segmenting the integration interval into subintervals and approximating the curve's area using rectangles.

III.1.2. Algorithm

Here's a step-by-step algorithm:

1. Define the Function: $f(x)$

2. Define the interval $[a, b]$.

3. Choose the Number of Subintervals:
N into which the interval $[a, b]$ will be divided. The larger the value of N, the better the approximation.

4. Compute the Width of Each Subinterval: Δx using the formula:

$$\Delta x = \frac{b-a}{N}$$

5. Compute the Midpoint x_i of Each Subinterval using the formula:

$$x_i = a + \frac{(2i-1)}{2}\Delta x \qquad \text{for } i=1, 2, \ldots, N.$$

6. Approximate the Integral: using the formula:
$\int_a^b f(x)dx \approx \sum_{i=1}^{N} f(x_i)\Delta x$
where $f(x_i)$ is the value of the function at the midpoint x_i of each subinterval.

III.1.3. Example

Suppose we want to calculate the integral of a function $f(x)=x^2$ over the interval $[0,1]$ using the rectangle method.

1. Define the Function:

Let's consider the function $f(x) = x^2$

2. Define the interval [0,1].

3. Choose the Number of Subintervals:

N =4.

4. Compute the Width of Each Subinterval:

$$\Delta x = \frac{b-a}{N}$$

$$= \frac{1-0}{4}$$

$$= 0{,}25$$

5. Compute the Midpoint x_i of each Subinterval using the formula:

i	**1**	**2**	**3**	**4**
x_i	0.125	0.375	0.625	0.875

6. Approximate the Integral: using the formula:

i	**1**	**2**	**3**	**4**
x_i	0.125	0.375	0.625	0.875
$f(x_i)$	0.0156	0.1406	0.3906	0.7656

$$\int_a^b f(x)dx \approx (0.0156 * 0.25) + (0.1406 * 0.25) + (0.3906 * 0.25) + (0.7656 * 0.25)$$

$$= 0.3281$$

III.1.4. MATLAB code

```
    clear all;          close all;          clc;
    disp(['******* Numerical Integration with Rectangle Method *******'])

    disp(['Please define the Interval [a b] :']);
    a=input('a =');
    b=input('b =');

    disp(['Please define the number of subintervals N :']);
    N=input('N =');

    % Define the function
    f= @(x) x.^2;

    % Compute the width of each subinterval
    dx = (b - a) / N;

    % Compute the midpoint of each subinterval
    x_midpoints = a + (0.5:1:N-0.5) * dx;

    % Evaluate the function at each midpoint
    f_values = f(x_midpoints);

    % Approximate the integral using the rectangle method
    integral_approx = sum(f_values) * dx;

    % Display the result
    disp(['Approximated integral using rectangle method: ',
num2str(integral_approx)]);
```

III.2. Worksheet 2: Trapezoidal rule

III.2.1. Concept

The trapezoidal rule, another numerical integration method, approximates a function's integral by segmenting the integration interval into trapezoids and summing the areas of these shapes.

III.2.2. Algorithm

Here's a step-by-step algorithm:

1. Define the Function.
2. Define the Interval.
3. Choose the Number N of Subintervals.
4. Compute the Width Δx of Each Subinterval using the formula:

   $\Delta x = \frac{b-a}{N}$.

5. Compute the Values of the Function at the Endpoints of Each Subinterval.
6. Approximate the Integral using the trapezoidal rule formula:

   $\int_a^b f(x)dx \approx \frac{\Delta x}{2}(f(a) + 2\sum_{i+1}^{N-1} f(x_i) + f(b))$

   where x_i are the points within the interval $[a, b]$ excluding the endpoints.

III.2.3. Example

Suppose we want to calculate the integral of a function $f(x) = x^2$ over the interval [0,1] using the trapezoidal rule:

1. Define the Function: Let's consider

   $f(x) = x^2$

2. Define the interval: Let's consider the interval [0,1].
3. Choose the Number of Subintervals: Let's consider N=4.

4. Compute the Width of Each Subinterval:

$\Delta x = \frac{b-a}{N} = \frac{1-0}{4} = 0.25.$

5. Compute the Values of the Function at the Endpoints of Each Subinterval:

p	0	0.25	0.5	0.75	1
F(p)	0	0.0625	0.25	0.5625	1

6. Approximate the Integral:

Approximate the integral using the trapezoidal rule formula:

$\int_a^b f(x)dx \approx 0.3438$

where x_i are the points within the interval $[a, b]$ excluding the endpoints.

III.2.4. MATLAB code

```
    clear all;          close all;          clc;
    disp(['****** Numerical Integration with trapezoidal Method ******'])
    disp(['Please define the Interval [a b]:']);
    a=input('a =');     b=input('b =');
    disp(['Please define the number of subintervals N:']);
    N=input('N =');
    % Define the function f(x)
    f= @(x) x.^2;
    % Compute the width of each subinterval
    dx = (b - a) / N;
    % Evaluate the function at the endpoints of each subinterval
    x_val = linspace(a, b, N+1);
    f_val = f(x_val);
    % Approximate the integral using the trapezoidal rule
    int_approx = dx/2 * (f_val (1) + 2*sum(f_val (2:end-1)) + f_val(end));
    % Display the result
    disp(['Approximated integral using trapezoidal rule: ',
num2str(int_approx)]);
```

III.3. Worksheet 3: Simpson's rule

III.3.1. Concept

Simpson's rule is an approach in numerical integration used to approximate the integral of a function by dividing the interval of integration into smaller subintervals and fitting each subinterval with a quadratic polynomial.

III.3.2. Algorithm

Here's a step-by-step algorithm:

1. Define the Function.
2. Define the Interval.
3. Choose the Number of Subintervals N into which the interval $[a, b]$ will be divided.
4. Compute the Width Δx of Each Subinterval using the formula:

   $\Delta x = \frac{b-a}{N}$.

5. Compute the Midpoint x_i of Each Subinterval using the formula:

   $$x_i = a + \frac{(2i-1)}{2}\Delta x \qquad \text{for } i=1, 2, \ldots, N.$$

6. Compute the Values of the Function at the Endpoints and Midpoints of Each Subinterval.
7. Approximate the Integral using the Simpson's rule formula:

   $\int_a^b f(x)dx \approx \frac{\Delta x}{3}(f(a) + 4\sum_{i+1}^{N-1} f(x_{2i}) + f(b))$

   Where:

   x_i are the points within the interval $[a, b]$ including the endpoints.

III.3.3. Example

Suppose we want to calculate the integral of a function $f(x) = x^2$ over the interval [0,1] using the trapezoidal rule:

1. Define the Function: Let's consider

   $f(x) = x^2$

2. Define the interval: Let's consider the interval [0,1].

3. Choose the Number of Subintervals: Let's consider N=4.

4. Compute the Width of Each Subinterval:

   $\Delta x = \frac{b-a}{N} = \frac{1-0}{4} = 0.25.$

5. Compute the Values of the Function at the Endpoints and Midpoints of Each Subinterval:

i	1	2	3	4
x_i	0.125	0.375	0.625	0.875

6. Compute the Values of the Function at the Endpoints and Midpoints of Each Subinterval:

p	0	0.125	0.25	0.375	0.5	0.625	0.75	0.875	1
$F(p)$	0	0.0156	0.0625	0.1406	0.25	0.3906	0.5625	0.7656	1

1. Approximate the Integral:

   Approximate the integral using the trapezoidal rule formula:

   $\int_a^b f(x)dx \approx 0.3438$

   where x_i are the points within the interval $[a, b]$ excluding the endpoints.

III.3.4. MATLAB code

```
    clear all;          close all;          clc;

    disp(['******* Numerical Integration with Simpson Method *******'])

    disp(['Please define the Interval [a b] :']);
    a=input('a =');
    b=input('b =');

    disp(['Please define the number of subintervals N :']);
    N=input('N =');

    % Define the function f(x)
    f= @(x) x.^2;

    % Compute the width of subintervals
    dx = (b - a) / N;

    % Compute the values of the function at the midpoints and endpoints of
each subinterval
    x_values = linspace(a, b, N+1);
    f_values = f(x_values);

    % Approximate the integral using Simpson's rule
    integral_approx = dx/3 * (f_values(1) + 4*sum(f_values(2:2:end-1)) +
2*sum(f_values(3:2:end-2)) + f_values(end));

    % Display the result
    disp(['Approximated integral using Simpson''s rule: ',
num2str(integral_approx)]);
```

Chapter IV:

Differential Equation Solving

A differential equation is an equation that involves an unknown function and its derivatives. These equations are fundamental in describing various physical, biological, and economic phenomena.

IV.1. Worksheet 1: Euler's method

IV.1.1. Concept

Euler's method is a numerical technic employed for approximating solutions to ordinary differential equations (ODEs). It's one of the simplest and most straightforward numerical methods for solving initial value problems.

IV.1.2. Algorithm

1. Defining the ODE to resolve: $\frac{dy}{dx} = f(x, y)$
2. Define the starting point x_0
3. Define the initial value y_0.
4. Define the interval $[a,b]$.
5. Define the number of steps (n) for the approximation.
6. Determine the time steps (h) for the approximation.
7. Applying Euler's formula and iterate n times, updating y and x at each step using the formulas: $x_{n+1} = x_n + h$ and $y_{n+1} = y_n + hf(x_n, y_n)$
8. Plotting the solution: Use the plot function to visualize the approximate solution with respect to x.
9. Comparison with the analytical solution.
10. Improving the accuracy: We can improve the accuracy of the solution by decreasing the time step.

IV.1.3. Example

Use Euler's method to solve the differential equation $\frac{dy}{dx} = 2x^2 + y$ with the initial condition: $y(0) = 1$ over the interval [0,1].

1. Define the differential equation: $\frac{dy}{dx} = 2x^2 + y$
2. Define the starting point: $x(1) = 0$
3. Define the initial value: $y(1) = 1$
4. Define the interval: [0,1]
5. Define the number of steps: $n = 10$
6. Determine the time step: $h = \frac{1-0}{10} = 0.1$

7. Calculating successive values of y using the Euler formula.

   We obtain the following results:

n	x_n	y_n
0	0	1
1	0,1	1.1000
2	0,2	1.2120
3	0,3	1.3412
4	0,4	1.4933
5	0,5	1.6747
6	0,6	1.8921
7	0,7	2.1533
8	0,8	2.4667
9	0,9	2.8413
10	1	3.2875

IV.1.4. MATLAB Code

```
clear;  clc;    close

% Define the function f(x,y)
f=@(x, y) 2*x^2 +y;

% Define the analytical solution
fexact=@(x)  (x.^2 + 1) .* exp(x);

% Set Interval
a=0;        b=1;

% Number of steps
n = 10;

% Time step
h = (b-a)/n;

% Set initial values
x(1) = 0;   y(1) = 1;

% Apply Euler's formula
for i = 1:n
  y(i+1) = y(i) + f(x(i), y(i))*h;
  x(i+1) = x(i) + h;
end

%Exact Solution
ye=fexact(x);

% Display results
disp('x values are:');        disp(x);
disp('y values are:');        disp(y);

% Plot the solution
plot(x, y,  x,  ye);    xlabel('x');      ylabel('y');
title('Solution of y'' = 2x^2 + y using Euler''s method');
legend('Euler','Exact')
```

IV.2. Worksheet 2: Runge-Kutta's method

IV.2.1. Concept

The fourth-order Runge-Kutta method is a numerical approach utilized for solving ordinary differential equations.

IV.2.2. Algorithm

1. Defining the ODE to resolve: $\frac{dy}{dx} = f(x, y)$
2. Define the starting point x_0
3. Define the initial value y_0.
4. Define the interval [a,b].
5. Define the number of steps (n) for the approximation.
6. Determine the time steps (h) for the approximation.
7. Iterate in loop n times, updating y and x at each step using the Rung-Kutta formulas:

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

Where:

- $k_1 = hf(t_i\ ,\ y_i)$
- $k_2 = hf\left(t_i + \frac{h}{2}\ ,\ y_i + \frac{k_1}{2}\right)$
- $k_3 = hf\left(t_i + \frac{h}{2}\ ,\ y_i + \frac{k_2}{2}\right)$
- $k_4 = hf(t_i + h\ ,\ y_i + k_3)$

IV.2.3. Example

We'll use the same differential equation: $\frac{dy}{dx} = 2x^2 + y$, with $y(0) = 1$ over the interval [0,1].

1. Define the ODE: $\frac{dy}{dx} = 2x^2 + y$
2. Define the starting point: $x(1) = 0$
3. Define the initial value: $y(1) = 1$
4. Define the interval: [0,1]
5. Define the number of steps: $n = 10$
6. Determine the time step: $h = \frac{1-0}{10} = 0.1$

7. Calculating successive values of x and y. We obtain the following results:

n	x_n	k_1	k_2	k_3	k_4	y_n
0	0	-	-	-	-	1
1	0,1	0.1000	0.1055	0.1058	0.1126	1.1059
2	0,2	0.1126	0.1207	0.1211	0.1307	1.2270
3	0,3	0.1307	0.1417	0.1423	0.1549	1.3693
4	0,4	0.1549	0.1692	0.1699	0.1859	1.5391
5	0,5	0.1859	0.2037	0.2046	0.2244	1.7436
6	0,6	0.2244	0.2461	0.2472	0.2711	1.9906
7	0,7	0.2711	0.2971	0.2984	0.3269	2.2888
8	0,8	0.3269	0.3577	0.3593	0.3928	2.6477
9	0,9	0.3928	0.4289	0.4307	0.4698	3.0780
10	1	0.4698	0.5118	0.5139	0.5592	3.5914

IV.2.4. MATLAB Code

```
clear;  clc;    close
% Define the differential equation function
f = @(x, y) 2*x^2 + y;

% The analytical solution
g = @(x, y) (x.^2 + 1) .* exp(x);

% Set Interval
a=0;        b=1;

% Define the starting point and the initial value
x(1) = 0;          y(1) = 1;

% Define the number of steps
n = 10;

% Determine the time step
h = (b-a)/n;

% Implement Runge-Kutta loop
for i=1:n
  k1 = h * f(x(i), y(i));
  k2 = h * f(x(i) + h/2, y(i) + k1/2);
  k3 = h * f(x(i) + h/2, y(i) + k2/2);
  k4 = h * f(x(i) + h, y(i) + k3);

  y(i+1) = y(i) + (k1 + 2*k2 + 2*k3 + k4) / 6;
  x(i+1)=x(i)+h;
  s(i)=g(x(i),y(i));
end

  s(i+1)=g(x(i+1),y(i+1));

% Plot the solution
plot(x, y, x, s)
title('Runge-Kutta 4th Order Solution')
xlabel('x');      ylabel('y');
```

Chapter V:

Systems of Linear Equations

A system of linear equations is a set of equations where each equation is linear, meaning it involves only the first powers of the variables. Solving such systems is fundamental in various fields such as engineering, physics, economics, and computer science.

V.1. Worksheet 1: Gauss-Jordan method

V.1.1. Concept

When employing the Gauss-Jordan elimination method to solve a system of linear equations, the process entails converting the augmented matrix of the system into reduced row-echelon form.

V.1.2. Algorithm

Here is the Gauss-Jordan algorithm:

1. Defining the system to resolve (Matrix A and B).
2. Determine the size (n) of the proposed system.
3. Form the Augmented Matrix $[A|B]$ by combining the coefficient matrix A and the constant matrix B.
4. Make the pivot of each row i 1 by dividing the entire row by the pivot value.
5. Eliminate all elements below and above the pivot element in column i by subtracting appropriate multiples of row i from the rows below it or above it.
6. After the matrix is in reduced row-echelon form, the solution to the system is given by the last column of the augmented matrix.

V.1.3. Example

Let's go through an example of how to implement this in MATLAB.

1. Defining the system to resolve (Matrix A and B): Suppose we have the following system of linear equations: $\begin{cases} 2x + y + z = 8 \\ -6x + 3y + 2z = -3 \\ -5x + 8y + 2z = -1 \end{cases}$

   It can be represented in matrix form $Ax=B$: $\begin{bmatrix} 2 & 1 & 1 \\ -6 & 3 & 2 \\ -5 & 8 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ -3 \\ -1 \end{bmatrix}$

2. Determine the size (n) of the proposed system: n=3

3. Form the Augmented Matrix $[A|B]$. For this system, the augmented matrix is: $[A \mid B] = \begin{bmatrix} 2 & 1 & 1 & 8 \\ -6 & 3 & 2 & -3 \\ -5 & 8 & 2 & -1 \end{bmatrix}$
4. Make the pivot of each row 1 by dividing the entire row by the pivot value.

   $R_1 = R_1/R_1(1)$, $R_2 = R_2/R_2(2)$, $R_3 = R_3/R_3(3)$

   The new $[A|B]$ is: $[A \mid B] = \begin{bmatrix} 1 & 0.5 & 0.5 & 4 \\ -2 & 1 & 0 & -1 \\ -2.5 & 4 & 1 & -0.5 \end{bmatrix}$

5. Eliminate all elements below and above the pivot elements by successive addition or substruction of rows.

   The new $[A|B]$ is: $[A \mid B] = \begin{bmatrix} 1 & 0 & 0 & 1.9020 \\ 0 & 1 & 0 & 0.0196 \\ 0 & 0 & 1 & 4.1765 \end{bmatrix}$

6. After the matrix is in reduced row-echelon form, the solution to the system is given by the last column of the augmented matrix: Solution $= \begin{bmatrix} 1.9020 \\ 0.0196 \\ 4.1765 \end{bmatrix}$

V.1.4. MATLAB Code

```
clear;      clc
% Define the system
A = [2 1 1; -6 3 2; -5 8 2];
B = [8;-3;-1];

% Determine the Number of rows
n = size(A, 1);

% Forming the Augmented Matrix
AB = [A B];

% Gauss-Jordan elimination
for i = 1:n
    % Make the diagonal element 1
    AB(i, :) = AB(i, :) / AB(i, i);

    % Make the other elements in the current column 0
    for j = 1:n
        if i ~= j
            AB(j, :) = AB(j, :) - AB(j, i) * AB(i, :);
        end
    end
end

% The last column of A is the solution
solution = AB(:, end);

% Display the solution
disp('The solution is:');
disp(solution);
```

V.2. Worksheet 2: Gauss-Seidel method

V.2.1. Concept

The Gauss-Seidel method is an iterative technique for solving a square system of linear equations. The process involves solving each equation in the system for a particular variable, and then iterating using the updated values.

V.2.2. Algorithm

Here is the Gauss-Jordan algorithm:

1. Defining the system to resolve (Matrix A and B).
2. Determine the size (n) of the proposed system.
3. Initial values for all variables.
4. Solve each equation in the system for a particular variable
5. Repeat the last step until the solution converges to a desired tolerance

V.2.3. Example

Consider the same system of linear equations:

1. Defining the system to resolve: $\begin{cases} 2x + y + z = 0 \\ 9x + 3y + 2z = -3 \\ 5x + 8y + 2z = -1 \end{cases}$
2. Determination of the size: n=3
3. Initial values: $x(1) = 0, \quad y(1) = 0, \quad z(1) = 0.$
4. Solve each equation in the system for a particular variable
   $x(i+1) = \frac{1}{2}\left(-y(i) - z(i)\right)$
   $y(i+1) = \frac{1}{3}\left(-3 - 9x(i) - 2z(i)\right)$
   $z(i+1) = \frac{1}{2}\left(-1 - 5x(i) - 8y(i)\right)$
5. Repeat the last step until the solution converges to a desired tolerance

i	$x(i)$	$y(i)$	$z(i)$
0	0	0	0
1	0.5	-2.6667	-0.5
2	0	-2.8333	-13.0832

V.2.4. MATLAB Code

```
% Gauss-Seidel Method to Solve Linear Systems

clear;      clc

% Define the augmented matrix
A = [2 1 1; 9 3 2; 5 8 2];

B = [0; -3; -1];

% Number of rows
n = size(A, 1);

% Initialize variables
x = zeros(3, 1); % Initial guess [0; 0; 0]

tolerance = 1e-6;

max_itr = 1000;

itr = 0;

% Gauss-Seidel iteration
for k = 1:max_itr
    x_old = x;

    for i = 1:n
        sum = 0;

        for j = 1:n
            if j ~= i
                sum = sum + A(i,j) * x(j);
            end
        end
        x(i) = (B(i) - sum) / A(i,i);

    end

    itr = itr + 1;

    % Check for convergence
    if norm(x - x_old, inf) < tolerance
        break;
    end
end

% Display the result
fprintf('Solution after %d iterations:\n', itr);

disp(x);
```

Conclusion

In conclusion, numerical methods serve as indispensable tools in modern scientific and engineering endeavors, providing efficient and reliable solutions to complex mathematical problems. From simulating physical phenomena to optimizing designs and making informed decisions, these techniques play a pivotal role in diverse fields. While challenges such as balancing accuracy and efficiency, ensuring stability and convergence, and addressing discretization errors exist, the versatility and effectiveness of numerical methods continue to drive innovation and advancement across disciplines. As technology evolves and computational resources become more accessible, numerical methods will remain fundamental in pushing the boundaries of knowledge and solving increasingly intricate problems in our quest for understanding and progress.