Scientific Computing Lab 2

Objectives: After completing this lab, students should be able to:

- Plot 2D curves using MATLAB
- Customize plots (title, labels, legends, fonts, grids)
- Plot multiple functions in a single window
- Use subplot for multiple graphs
- Generate parametric and polar plots
- Create 3D curves using plot3()
- Generate surface plots using mesh() and surf()
- Visualize contour plots
- Solve a system of linear equations using MATLAB
- Interpret graphical and numerical solutions

MATLAB provides powerful tools for visualization of mathematical objects. Graphical representation helps in understanding behaviour of functions, growth, oscillations, symmetry, singularities and geometric structures in higher dimensions. For single variable functions, 2D plots are used. For parametric curves, both x and y are expressed in terms of parameter. For multivariable functions z = f(x,y), surface and contour plots provide geometric interpretation. Logarithmic plots are useful for exponential growth and power laws. Axis control ensures correct scaling and geometric accuracy.

MATLAB can be used for creating various kinds of two-dimensional plots. Type plot(rand(1, 50)) at the prompt. This simply plots a graph where the values are 50 random numbers, each between 0 and 1, and the values are integers in the range 1 to 50. But the graph is continuous, because MATLAB generates the values in between by linear interpolation.

MATLAB 2D Graphical Representation

MATLAB provides the **plot()** command for 2D graphical representation.

Cartesian Plot:

```
x = a:step:b;
y = f(x);
plot(x,y);
```

Parametric plot:

```
t = a:step:b;
x = f(t);
y = g(t);
plot(x,y);
```

Polar Plot:

```
theta = a:step:b;
r = f(theta);
polarplot(theta,r);
```

In MATLAB, **line specifiers, marker specifiers, and colour specifiers** are used inside the plot() command to control the appearance of a graph. A **line specifier** determines the style of the line connecting data points, such as solid ('-'), dashed ('--'), dotted (':'), or dash-dot ('-.'). A **marker specifier** controls the symbol used to represent individual data points, such as circle ('o'), star ('*'), plus ('+'), square ('s'), or diamond ('d'). A **colour specifier** defines the color of the line or markers, such as red ('r'), green ('g'), blue ('b'), black ('k'), or magenta ('m'). These specifiers are combined in a single string inside the plot() function. For example, plot(x,y,'r--o') produces a red dashed line with circular markers. Thus, these specifiers help improve visualization and clarity of graphical representation.

The **hold on and hold off** commands in MATLAB are used to control whether new plots replace existing plots or are added to them. By default, when a new plot command is executed, MATLAB clears the current axes and displays only the new graph. When hold on is used, MATLAB retains the existing plot and allows multiple graphs to be drawn on the same axes, which is useful for comparing functions or displaying multiple datasets together. The hold off command restores the default behavior, meaning any new plot will replace the existing one. Thus, hold on enables overlaying of plots, while hold off resets the plotting behavior.

Important Commands in 2D Plotting

Command	Purpose	Syntax
plot	To draw 2D graph	plot(x,y)
xlabel	Label x-axis	xlabel('Text')
ylabel	Label y-axis	ylabel('Text')
title	Add title	title('Text')
legend	Add legend	legend('label1','label2')
grid on	Display grid	grid on
hold on	Multiple plots in same axes	hold on
subplot	Multiple plots in one window	subplot(m,n,p)
axis	Set axis limits	axis([xmin xmax ymin ymax])
xlim	Set x-axis limit	xlim([xmin xmax])
ylim	Set y-axis limit	ylim([ymin ymax])

Line, Marker and Colour Specifiers in MATLAB

Line Style	Meaning	Marker	Meaning	Colour Code	Colour Name
'-'	Solid line	'o'	Circle	'r'	Red
'--'	Dashed line	'+'	Plus sign	'g'	Green
':'	Dotted line	'*'	Star	'b'	Blue
'-.'	Dash-dot line	'.'	Point	'c'	Cyan
'none'	No line	'x'	Cross	'm'	Magenta
		's'	Square	'y'	Yellow
		'd'	Diamond	'k'	Black
		'^'	Upward triangle	'w'	White
		'v'	Downward triangle		

Line Style	Meaning	Marker	Meaning	Colour Code	Colour Name
		'>'	Right triangle		
		'<'	Left triangle		
		'p'	Pentagon		
		'h'	Hexagon		

Example 1:

Aim: To plot y = sin(x), y = cos(x)

```
x = 0:0.01:2*pi;
y1 = sin(x);
y2 = cos(x);

plot(x,y1,'r','LineWidth',2);
hold on;
plot(x,y2,'b--','LineWidth',2);

xlabel('x-axis','FontSize',12);
ylabel('y-axis','FontSize',12);
title('Sine and Cosine Functions','FontSize',14);
legend('sin(x)','cos(x)');
grid on;
```

Example 2:

Aim: To plot $y = e^x$ and $y = \log(x)$, xy = 4

```
% Exponential
x1 = -2:0.01:2;
y1 = exp(x1);

% Logarithmic
x2 = 0.1:0.01:5;     % log defined for x > 0
```

```
y2 = log(x2);

% Hyperbola xy = 4  =>  y = 4/x
x3 = -5:0.01:-0.1;   % Negative branch
x4 = 0.1:0.01:5;     % Positive branch

y3 = 4./x3;
y4 = 4./x4;

plot(x1,y1,'b','LineWidth',2);
hold on;
plot(x2,y2,'r--','LineWidth',2);
plot(x3,y3,'g','LineWidth',2);
plot(x4,y4,'g','LineWidth',2);

xlabel('x-axis','FontSize',12);
ylabel('y-axis','FontSize',12);
title('Exponential, Logarithmic and Hyperbola','FontSize',14);
legend('e^x','log(x)','xy=4');
grid on;
```

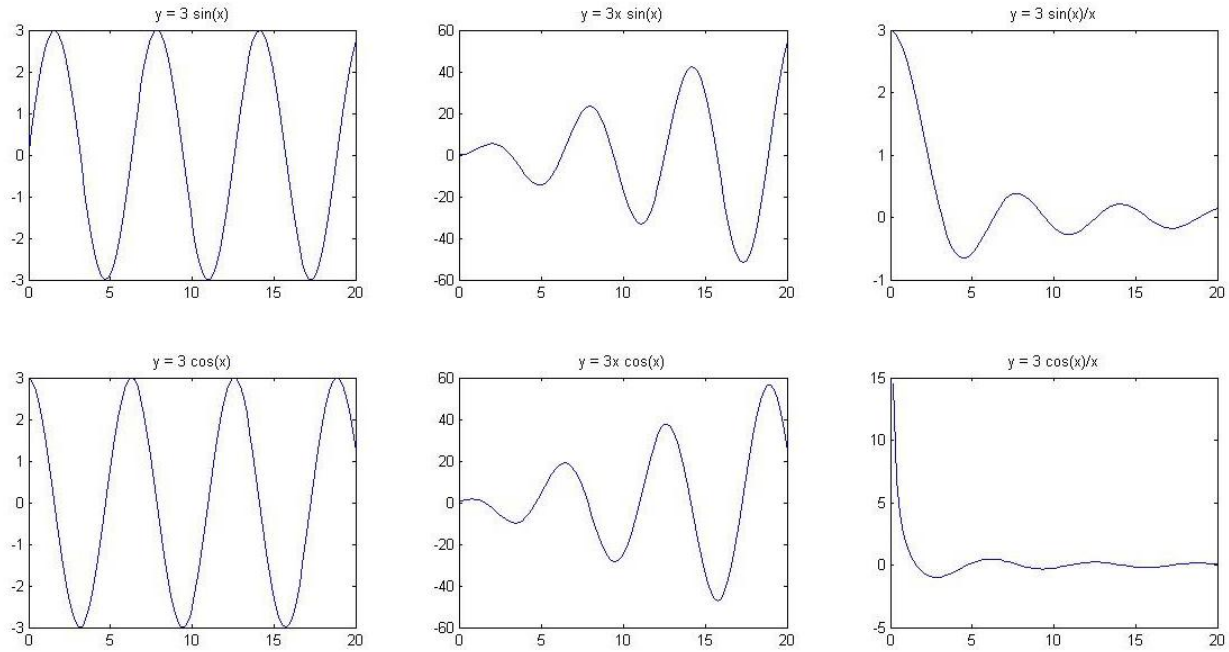
The subplot command is used to display **multiple graphs in a single figure window**. It divides the figure into a grid of rows and columns and allows each plot to be placed in a specified position. This is very useful for comparing different functions side by side or one above the other.

The general syntax is: subplot(m,n,p)
where m = number of rows, n = number of columns, p = position of the current plot
For example, subplot(2,1,1) creates the first plot in a grid of 2 rows and 1 column.
Subplots help in better visualization, comparison, and analysis of multiple mathematical functions within the same figure window.
For example the following code

```
x = linspace(0, 20);
    subplot(2, 3, 1); y = 3*sin(x); plot(x, y); title('y = 3 sin(x)')
    subplot(2, 3, 2); y = 3*x.*sin(x); plot(x, y); title('y = 3x sin(x)')
```

```
subplot(2, 3, 3); y = 3*sin(x)./x; plot(x, y); title('y = 3 sin(x)/x')
subplot(2, 3, 4); y = 3*cos(x); plot(x, y); title('y = 3 cos(x)')
subplot(2, 3, 5); y = 3*x.*cos(x); plot(x, y); title('y = 3x cos(x)')
subplot(2, 3, 6); y = 3*cos(x)./x; plot(x, y); title('y = 3 cos(x)/x')
```

gives the output as follows:



Example 3:

Aim: To plot sin(x), cos(x), tan(x) using subplot.

```
x = 0:0.01:2*pi;

subplot(3,1,1)
plot(x,sin(x));
title('sin(x)'); grid on;

subplot(3,1,2)
plot(x,cos(x));
title('cos(x)'); grid on;

subplot(3,1,3)
plot(x,tan(x));
title('tan(x)'); grid on;
```

Example 4:

Aim: To draw the circle $x^2 + y^2 = 1$ using parametric form

```
t = 0:0.01:2*pi;
x = cos(t);
y = sin(t);

plot(x,y,'LineWidth',2);
axis equal;
title('Unit Circle');
grid on;
```

Example 5:

Aim: Polar Plot of $r = \sin(3\theta)$

```
theta = 0:0.001:2*pi;      % Parameter theta
r = sin(3*theta);          % r = sin(3θ)

polarplot(theta, r,'r','LineWidth',2);

title('Polar Plot: r = sin(3\theta)');
grid on;
```

MATLAB 3D Graphical Representation

Example 6:

Aim: To plot helix

```
t = 0:0.01:10*pi;
x = cos(t);
y = sin(t);
z = t;
plot3(x,y,z,'LineWidth',2);
grid on;
xlabel('X'); ylabel('Y'); zlabel('Z');
title('3D Helix');
```

Example 7:

Aim: To plot the surface $z = x^2 + y^2$

```
[x,y] = meshgrid(-5:0.2:5);
z = x.^2 + y.^2;

surf(x,y,z);
xlabel('X'); ylabel('Y'); zlabel('Z');
title('Surface Plot: z = x^2 + y^2');
colorbar;
```

Example 8: Saddle surface

Aim: to plot $z = x^2 - y^2$

```
[x,y] = meshgrid(-5:0.2:5);
z = x.^2 - y.^2;

surf(x,y,z);
title('Saddle Surface');
colorbar;
```

Example 8:

Aim : Contour Plot

```
[x,y] = meshgrid(-5:0.2:5);
z = x.^2 + y.^2;

contour(x,y,z,20);
title('Contour Plot');
colorbar;
grid on;
```

The mesh and surf commands in MATLAB are used to visualize functions of two variables in three dimensions. The mesh command is mainly used when we want to understand the **geometric structure and shape** of a surface, as it produces a wireframe representation that clearly shows the grid lines and curvature of the function. It is helpful in analyzing the mathematical behavior of surfaces such as saddle points or paraboloids. On the other hand, the surf command is used when we want a **better**

visual representation of magnitude and variation, since it displays a filled and colored surface. The color shading helps to identify peaks, valleys, gradients, and symmetry more clearly. Thus, mesh is useful for structural understanding, while surf is preferred for detailed visualization and presentation.

Exercises:

1) Plot $y = x^3 - 3x + 1$for $-3 \leq x \leq 3$. Add title, axis labels, grid. Highlight turning points using markers.
2) Using subplot, plot: x^2 , x^3 , sinx + cosx. Include legend and proper formatting.
3) Plot the surface: $z = x^2 - y^2$. Use surf. Add colorbar
4) Plot both surface and contour of: $z = e^{-(x^2+y^2)}$ using subplot.
5) Plot the piecewise function $f(x) = \begin{cases} x^2 & 0 \leq x \leq 1 \\ e^{\frac{x-1}{2}} & 1 \leq x \leq 4 \\ e^{3/2} sin\left(\frac{4\pi}{x}\right) & 4 \leq x \leq 10 \end{cases}$
6) Plot the Lemniscate of Bernoulli which has the parametric equation

   x $= \frac{cost}{1+sin^2 t}$, y $= \frac{costsint}{1+sin^2 t}$, $0 \leq t \leq 2\pi$
7) Create a subplot layout with the following

   Top-left: $y = x^2$ Top-right: $y = \sqrt{x}$

   Bottom-left: $y = e^x$ Bottom-right: $y = \ln(x)$
8) Create a meshgrid for $x, y \in [-5,5]$.

   Plot $z = \sin\left(\sqrt{x^2 + y^2}\right)$ using both mesh and surf. Compare the difference in visualization.
9) Plot a Möbius strip using parametric equations. Use surf and apply a colormap for visual effect.

****

Systems of Linear Equations

Consider a system of linear equations:

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

This system can be represented as the matrix equation

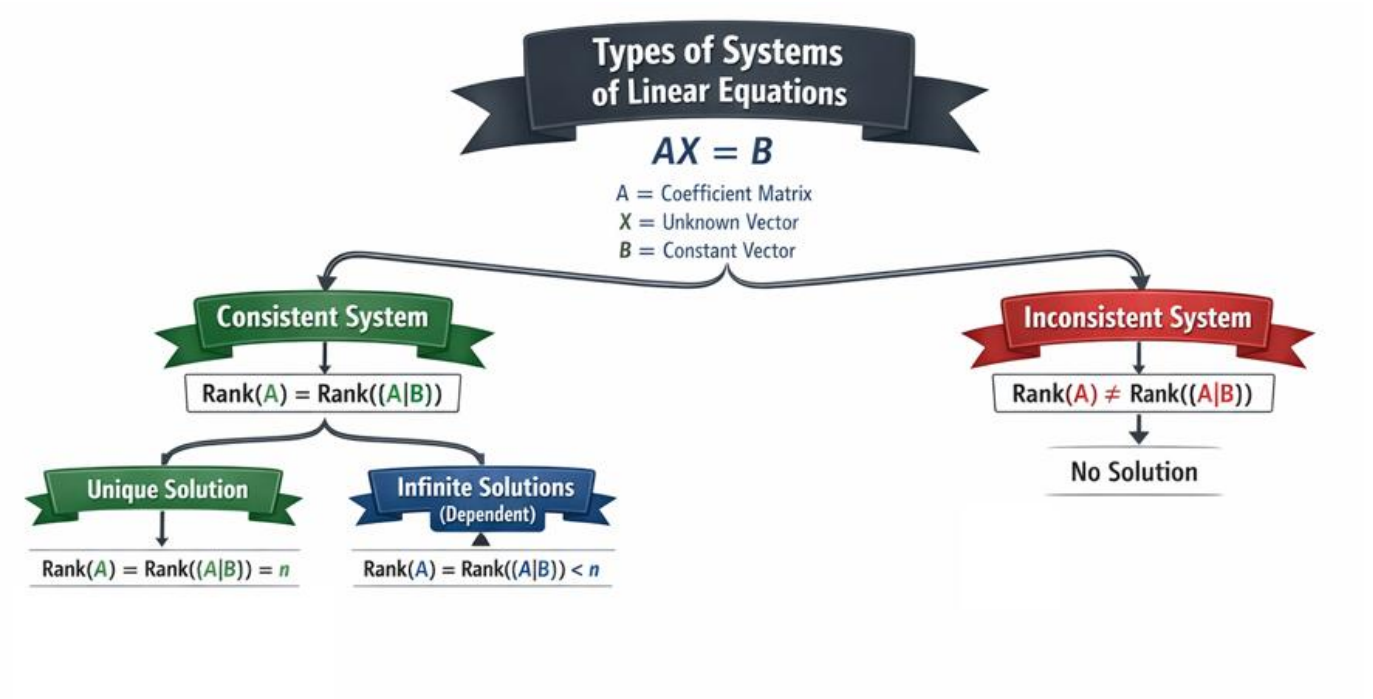
$$A \cdot \vec{x} = \vec{b}$$

where A is the coefficient matrix:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

$\vec{b}$ is the vector containing the right-hand sides of the equations:

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



Solving Systems of Linear Equations

1. Direct Methods:

1.1 Gaussian Elimination (Naïve Method): Gaussian elimination is one of the most fundamental techniques in numerical linear algebra. The method transforms the system of equations into an equivalent upper triangular system using elementary row operations, followed by back substitution to obtain the solution. The procedure consists of two phases:

1. **Forward Elimination:** Convert matrix A into an upper triangular matrix using elimentary row operations
2. **Back Substitution:** Solve the triangular system starting from the last equation.

Example 1: Solve the system

$$0.0001x + y = 1$$
$$x + y = 2$$

Algorithm:

Step 1: Input the matrix A and vector b.
Step 2: Form the augmented matrix $[\,A \mid b\,]$.
Step 3: For $k = 1$ to $n - 1$

- For $i = k + 1$to n

$$m_{ik} = \frac{a_{ik}}{a_{kk}}$$
$$R_i \leftarrow R_i - m_{ik}R_k$$

Step 4: Obtain upper triangular system.

Step 5: Back substitution:

$$x_n = \frac{b_n}{a_{nn}}$$
$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}\,x_j\right)$$

Step 6: Output solution vector x.

Drawback of Gaussian Elimination: In this example, the pivot element is very small. Consequently, the multiplier becomes very large, producing large intermediate numbers and possible round-off errors in numerical computation. **To avoid this difficulty, pivoting strategies are introduced.**

1.2 Gaussian Elimination with Partial Pivoting

In partial pivoting, before each elimination step, the equation having the largest coefficient in absolute value in the pivot column is interchanged with the current pivot equation. The objective is to transform the system into an equivalent upper triangular system using partial pivoting and then obtain the solution by back substitution.

Thus, we ensure

$$| a_{kk} | = \max_{i \geq k} | a_{ik} |.$$

Example 2: Solve the system

$$0.0001x + y = 1$$
$$x + y = 2$$

Algorithm:

Step 1: Input the coefficient matrix A and the right-hand-side vector b.

Step 2: Form Augmented Matrix $[A \mid b]$.

Step 3: Forward Elimination with Partial Pivoting

For $k = 1,2, \dots, n-1$:

1. Pivot Selection

   Find the index $p$such that

   $$| a_{pk} | = \max_{i=k,k+1,\dots,n} | a_{ik} |.$$

2. Row Interchange

   If $p \neq k$, interchange row $k$with row p.

3. Elimination

   For each $i = k+1, k+2, \dots, n$, compute the multiplier

   $$m_{ik} = \frac{a_{ik}}{a_{kk}}.$$

   Perform the row operation

   $$R_i \leftarrow R_i - m_{ik} R_k.$$

After completing all steps, the matrix is transformed into an upper triangular form.

Step 4: Back Substitution

Compute the unknowns starting from the last equation:

$$x_n = \frac{b_n}{a_{nn}}.$$

For $i = n-1, n-2, \ldots, 1$:

$$x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=i+1}^{n} a_{ij}\, x_j\right).$$

Step 5: Output

Display the solution vector

$$x = (x_1, x_2, \ldots, x_n)^T.$$

1.3 MATLAB Built-in Solver (A\b)

In MATLAB, systems of linear algebraic equations of the form

$$Ax = b$$

can be solved efficiently using the left division operator

$$x = A\backslash b$$

Depending on the structure of the matrix, MATLAB uses:

1. **LU Decomposition** — for general square matrices.
2. **Cholesky Decomposition** — when the matrix is symmetric and positive definite.
3. **QR Decomposition** — for rectangular or overdetermined systems.
4. **Specialized sparse solvers** — when the matrix contains many zero elements.

Example 3: Solve the system of equations:

$$\begin{aligned} 2x - y + z &= 2 \\ 3x + 3y + 9z &= -1 \\ 3x + 3y + 5z &= 4 \end{aligned}$$

Algorithm:

Step 1: Input matrix A and vector b

Step 2: Determine the size of matrix

$$[n, m] = \text{size}(A)$$

Step 3: Check whether matrix is square

If $n \neq m$, display "Matrix must be square" and terminate the program.

Step 4: Check compatibility of vector b

If length of $b \neq n$, display "Dimension mismatch between A and b" and terminate the program.

Step 5: Compute the solution using MATLAB operator

$$x = A\backslash b$$

Step 7: Display the solution vector

2. Iterative Methods:

Iterative methods start with an initial guess and generate a sequence of approximations that converge to the exact solution. These methods are especially useful for:

- Large systems of equations
- Sparse matrices
- Scientific simulations and engineering problems

Unlike direct methods, iterative methods require convergence conditions to be satisfied.

A sufficient condition for convergence is **diagonal dominance** of the coefficient matrix:

$$| a_{ii} | > \sum_{j \neq i} | a_{ij} |$$

2.1 Jacobi Method

The Jacobi method is an iterative technique where each unknown is computed using values from the previous iteration only. The iteration formula is derived by rewriting each equation as:

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} \, x_j^{(k)} \right)$$

The method is simple but may converge slowly.

Example 4: Solve the system:

$$10x + y + z = 12$$
$$2x + 10y + z = 13$$
$$2x + 2y + 10z = 14$$

Initial guess: $x^{(0)} = (0,0,0)$

Algorithm:

Input: A, b, Initial guess $x^{(0)}$, Tolerance ϵ, Maximum number of iterations N

Output: Approximate solution vector x

Step 1: Read $A, b, x^{(0)}, \epsilon, N$

Step 2: For $k = 1$ to N

For each variable $i = 1$ to n

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} \, x_j^{(k)} \right)$$

End For

Step 3: Check convergence

If

$$\| x^{(k+1)} - x^{(k)} \| < \epsilon$$

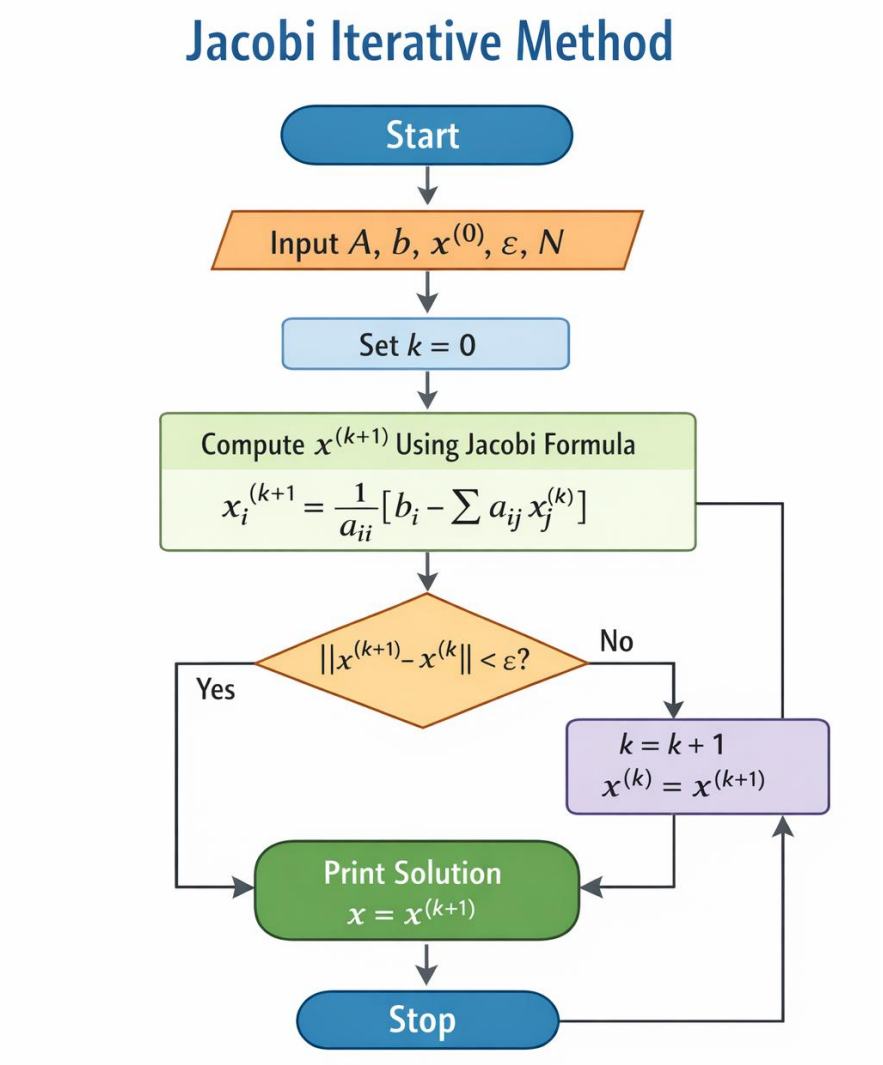
then stop.

Otherwise set

$$x^{(k)} = x^{(k+1)}$$

and continue.

Step 4: Print solution



2.2 Gauss–Seidel Method

The Gauss–Seidel method improves upon the Jacobi method by using updated values immediately as soon as they are available during the iteration. Thus, it generally converges faster than the Jacobi method. The iteration formula becomes:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}\right)$$

Example: Solve the system

$$\begin{aligned} 10x_1 - x_2 + 2x_3 &= 6 \\ -x_1 + 11x_2 - x_3 + 3x_4 &= 25 \\ 2x_1 - x_2 + 10x_3 - x_4 &= -11 \\ 3x_2 - x_3 + 8x_4 &= 15 \end{aligned}$$

Initial guess: $x^{(0)} = (0,0,0,0)$

Algorithm:

Step 1: Input matrix A, vector b, tolerance ε, and maximum number of iterations N.

Step 2: Choose an initial approximation

$$x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \dots, x_n^{(0)}).$$

Step 3: For $k = 0,1,2,\dots,N-1$, compute

For $i = 1,2,\dots,n$:

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j<i} a_{ij}\, x_j^{(k+1)} - \sum_{j>i} a_{ij}\, x_j^{(k)}\right).$$

Step 4: Check convergence condition

$$\| x^{(k+1)} - x^{(k)} \| < \varepsilon.$$

If satisfied, stop; otherwise continue.

Step 5: Output approximate solution x.

Exercises:

1. Solve the system using Gauss Elimination Method:

$$2x - y + z = 8$$
$$-3x - y + 2z = -11$$
$$-2x + y + 2z = -3$$

   Write MATLAB code for Gauss elimination, Show augmented matrix after each elimination step.

2. Solve using Gauss Elimination with and without Partial Pivoting:

$$10^{-5}x + y + z = 2$$
$$x + y + z = 3$$
$$x + 2y + 3z = 6$$

3. Solve using Gauss–Jacobi Method:

$$10x - y + 2z = 6$$
$$-x + 11y - z + 3w = 25$$
$$2x - y + 10z - w = -11$$
$$3y - z + 8w = 15$$

   Take initial guess $(0, 0, 0, 0)$. Perform iterations until tolerance 10^{-4}, display iteration table and show number of iterations required.

4. In an electrical network, currents satisfy:

$$10I_1 - 2I_2 - I_3 = 3$$
$$-2I_1 + 10I_2 - I_3 = 15$$
$$-I_1 - I_2 + 10I_3 = 27$$

   Solve using Jacobi method and Gauss–Seidel method. Plot convergence graph.

****