**INDIAN INSTITUTE OF TECHNOLOGY TIRUPATI**
**DEPARTMENT OF MATHEMATICS AND STATISTICS**

---

Project - 1 **MA517M-Basic Programming Laboratory** Last Date: 09 November 2025

**Name** **Roll No.: MA25M004**

---

# Relations

1. Design a C++ class `Vector` that represents a mathematical vector with `double` precision elements. Implement member functions to:

   - Input and display the vector elements.
   - Compute the **outer product** with another vector to form a `Matrix` object.

   Mathematically,

$$u = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{pmatrix}, \quad v = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}, \quad \Longrightarrow u \otimes v^T = \begin{pmatrix} u_1v_1 & u_1v_2 & \cdots & u_1v_n \\ u_2v_1 & u_2v_2 & \cdots & u_2v_n \\ \vdots & \vdots & \ddots & \vdots \\ u_mv_1 & u_mv_2 & \cdots & u_mv_n \end{pmatrix}$$

   Here $m \neq n$.

2. Add a member function to the `Vector` class that computes the **1-norm** of a vector using recursion:

$$\|u\|_1 = \sum_{i=1}^{m} |u_i|$$

3. Design a second class `Matrix` that stores an $m \times n$ matrix and implements:

   - Input and display of the matrix elements.
   - Computation of the **1-norm (column-sum-norm)**:

$$\|A\|_1 = \max_{1 \leq j \leq n} \sum_{i=1}^{m} |a_{ij}|$$

4. Overload the comparison operators `<`, `>`, and `==` to compare the following quantities:

$$\|u \otimes v^T\|_1 \quad \text{and} \quad \|u\|_1\|v\|_1$$

   so that the statement

```
if (OuterProductMatrix < ProductOfNorms)
```

   returns `true` if $\|u \otimes v^T\|_1 < \|u\|_1\|v\|_1$, and similarly for the other relations.

   Use the following data for testing:

$$u = \begin{pmatrix} 0.8147 \\ 0.9058 \\ 0.1270 \\ 0.9134 \end{pmatrix}, \quad v = \begin{pmatrix} 0.9575 \\ 0.9649 \\ 0.1576 \\ 0.9706 \\ 0.9572 \end{pmatrix}, \quad (m = 4, n = 5)$$

5. Extend the `Matrix` class to include a function that solves the linear system

$$Ax = b$$

using the **Jacobi iterative method**. The function should:

- Accept a vector $b$ as input,
- Initialize $x$ with zeros,
- Iterate until convergence or a fixed number of iterations.

Display the solution vector $x$ and verify the residual norm $\|Ax - b\|_2$.

## Project - 2: Conway's Game of Life Using C++ Classes

### Problem Statement

Design and implement **Conway's Game of Life** using **C++ classes**. This is a zero-player game where the evolution of a 2D grid is determined by its initial state and simple rules. Each cell in the grid can be *alive* or *dead*, and the next generation of the grid is calculated based on the number of alive neighbors. The project should utilize object-oriented programming concepts such as classes, objects, encapsulation, and methods for handling game logic and grid updates.

### Project Requirements

1. Create a `Cell` class to represent a single cell in the grid.
   (a) Data member: current state (alive or dead)
   (b) Methods: setState(), getState()
2. Create a `Grid` class to represent the game board (matrix).
   (a) Initialize the grid with random alive and dead cells.
   (b) Include a method to display the current state of the grid.
   (c) Include a method to count alive neighbors for each cell.
   (d) Include a method to update the grid according to the following rules:
      - Any live cell with fewer than two live neighbors dies (underpopulation).
      - Any live cell with two or three live neighbors lives on.
      - Any live cell with more than three live neighbors dies (overpopulation).
      - Any dead cell with exactly three live neighbors becomes alive (reproduction).
3. Create a `Game` class to manage simulation.
   (a) Display options for the user: `Start`, `Next Generation`, or `Quit`.
   (b) Allow the user to advance the simulation one generation at a time.
   (c) Allow the user to quit and confirm before exiting.
   (d) Optionally, allow the user to specify the number of generations to simulate automatically.
4. Ensure proper encapsulation of cell and grid operations within the respective classes.

## Suggested Class Structure

1. `Cell` Class:
   - Data member: `bool isAlive`
   - Methods: setState(bool state), getState()
2. `Grid` Class:
   - Data member: 2D array of `Cell` objects
   - Methods: initializeGrid(), displayGrid(), countAliveNeighbors(int row, int col), updateGrid()
3. `Game` Class:
   - Data member: `Grid` object, user choice, generation counter
   - Methods: startGame(), nextGeneration(), confirmQuit()

## Reference

For more details about Conway's Game of Life, visit: `https://en.wikipedia.org/wiki/Conway's_Game_of_Life`