lu decomposition

## FUNCTION : LU Decomposition

**Purpose:** This function decomposes a square matrix **A** into a **lower triangular matrix (L)** and an **upper triangular matrix (U)** such that **A = LU**.

A screenshot of a computer program

AI-generated content may be incorrect.

**Step-by-Step Explanation**

**1. Matrix Traversal**

* The outer loops iterate over each element of the matrix:
  + i represents the current row.
  + j represents the current column.

**2. Upper Triangular Matrix (U)**

For elements where i ≤ j (diagonal and upper triangle):

* Initialize U[i][j]:

U[i][j] = A[i][j];

* Subtract Contributions from Previous Rows:

for (int k = 0; k < i; k++) {

U[i][j] -= L[i][k] \* U[k][j];

}

This eliminates dependencies on earlier rows using the formula:

* U[i][j]=A[i][j]−
* Set L[i][j]:
  + Diagonal (i == j): L[i][j] = 1.0 (unit lower triangular matrix).
  + Upper triangle (i < j): L[i][j] = 0.0.

**3. Lower Triangular Matrix (L)**

For elements where i > j (lower triangle):

* Initialize L[i][j]:

L[i][j] = A[i][j];

* Subtract Contributions from Previous Columns:

for (int k = 0; k < j; k++) {

L[i][j] -= L[i][k] \* U[k][j];

}

This uses the formula:

* L[i][j]=
* Normalize by U[j][j]:

L[i][j] /= U[j][j];

* Set U[i][j]:

c

U[i][j] = 0.0; *// Ensure upper triangle elements below diagonal are zero*

## function: Forward Substitution

**Purpose:** This function solves a system **Lx = b** where **L** is a **lower triangular matrix**

A computer screen shot of a program code

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Memory Allocation**:
   * Allocates memory for the solution vector x.
   * Exits with an error if allocation fails.
2. **Forward Substitution Loop**:
   * Iterates over each row i from 0 to n-1:
     + **Initialize**: x[i] = b[i] (start with the right-hand side value).
     + **Subtract Known Terms**: For each column j < i, subtract L[i][j] \* x[j] to eliminate dependencies on previously computed x[j].
   * **No Division Needed**: **L** is a **unit lower triangular matrix** (1s on the diagonal), so no division by L[i][i] is required.

## function: Backward Substitution

Purpose: This function solves a system **Ux = b** where **U** is an **upper triangular matrix**.

A computer screen shot of a program code

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Memory Allocation**:
   * Allocates memory for the solution vector x.
   * Exits with an error if allocation fails.
2. **Backward Substitution Loop**:
   * Iterates over rows i from **bottom to top** (n-1 to 0):
     + **Initialize**: x[i] = b[i] (start with the right-hand side value).
     + **Subtract Known Terms**: For each column j > i, subtract U[i][j] \* x[j] to eliminate dependencies on already computed x[j].
     + **Normalize**: Divide by U[i][i] to isolate x[i].

## function : Reading Matrix and Vector Input from a File

**Purpose:** This function reads a square matrix **A** of size n x n and a vector **b** of size n from a file named "inputs.txt". It ensures proper error handling for file operations and input parsing.

A screen shot of a computer program

AI-generated content may be incorrect.

**Step-by-Step Explanation**

1. **Open File**:
   * FILE \*file = fopen("inputs.txt", "r"); opens the file "inputs.txt" in read mode.
   * If the file cannot be opened (e.g., it doesn't exist), the program prints an error message ("File not found") and exits.
2. **Read Matrix Elements**:
   * The nested loop for (int i = 0; i < n; i++) { for (int j = 0; j < n; j++) { ... } } iterates through each row (i) and column (j) of matrix **A**.
   * fscanf(file, "%lf", &A[i][j]) reads a floating-point number from the file and stores it in the corresponding position in **A**.
   * If the input is invalid or missing, the program prints "Invalid input" and exits.
3. **Read Vector Elements**:
   * The loop for (int i = 0; i < n; i++) { ... } iterates through each element of vector **b**.
   * fscanf(file, "%lf", &b[i]) reads a floating-point number from the file and stores it in the corresponding position in **b**.
   * Similar error handling ensures that invalid or missing input causes the program to terminate with an error message.
4. **Close File**:

fclose(file); closes the file after all data has been read to release system resources.

## Main Program for Solving Linear Systems via LU Decomposition

**Purpose:** This program solves a linear system **Ax = b** using LU decomposition. It reads matrix **A** and vector **b** from a file, factors **A** into lower triangular matrix **L** and upper triangular matrix **U**, solves for **x** using forward/backward substitution, and reports timings and results.

A screen shot of a computer

AI-generated content may be incorrect.

**Step-by-Step Explanation**

**1. User Input & Memory Allocation**

* Prompts the user for the number of equations (n).
* Allocates memory for:
  + **A** (n x n matrix).
  + **L** (unit lower triangular matrix, initialized with calloc).
  + **U** (upper triangular matrix, initialized with calloc).
  + **b** (vector of size n).
* Includes error handling for memory allocation failures.

**2. Read Input File**

* Calls read\_file\_input(n, A, b) to load **A** and **b** from "inputs.txt".

**3. LU Decomposition & Timing**

* LU\_Decomposition(n, A, L, U) factors **A** into **L** and **U**.
* Measures and prints the time taken for decomposition in nanoseconds.

**4. Solve Linear System**

* **Forward Substitution**: Solves **Ly = b** to get intermediate vector **y**.
* **Backward Substitution**: Solves **Ux = y** to get the solution vector **x**.
* Measures and prints the time taken for solving the system.

**5. Print Results**

* Displays the solution vector **x** with 6 decimal places.

**6. Memory Cleanup**

* Frees all dynamically allocated memory for **A**, **L**, **U**, **b**, **y**, and **x**.

**Key Notes**

* **Timing**:
  + Uses clock() to measure CPU time.
  + Converts time to nanoseconds for readability.
  + **WSL vs. Terminal**: Timing accuracy may vary between environments.