ASSIGNMENT 1- LU Decomposition

BY CO24BTECH11023

Question-

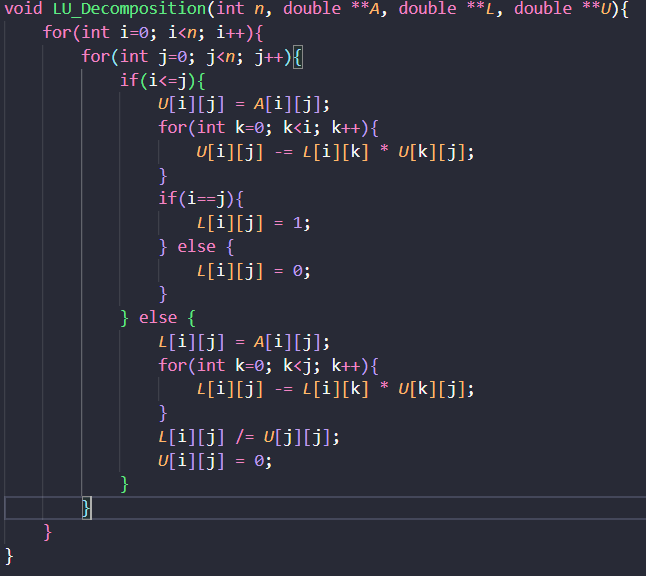
Solve

=

1. Find the L and U matrices.
2. Find the value of X1, X2, X3, X4.

CODE

LU Decomposition Snippet:



**The Inputs for are :**

* n: The size of the square matrix (n x n)
* A: The input matrix to be decomposed
* L: The resulting lower triangular matrix
* U: The resulting upper triangular matrix

**Main Loop Structure:**

The function uses three nested loops to iterate through the matrix elements:

1. Outer loop i (rows): Iterates from 0 to n-1
2. Middle loop j (columns): Iterates from 0 to n-1
3. Inner loop k: Used for summation in the calculations

Algorithm Steps:

1. **For Upper Triangular Matrix (U) Calculation**:
   1. When i <= j (upper triangle including diagonal):
      1. Set U[i][j] = A[i][j]
      2. Subtract the sum of products: U[i][j] -= L[i][k] \* U[k][j] for k from 0 to i-1
   2. When i > j (lower triangle):
      1. Set U[i][j] = 0
2. **For Lower Triangular Matrix (L) Calculation**:
   1. When i == j (diagonal):
      1. Set L[i][j] = 1 (unit lower triangular matrix)
   2. When i < j (upper triangle):
      1. Set L[i][j] = 0
   3. When i > j (lower triangle):
      1. Set L[i][j] = A[i][j]
      2. Subtract the sum of products: L[i][j] -= L[i][k] \* U[k][j] for k from 0 to j-1
      3. Divide by the diagonal element of U: L[i][j] /= U[j][j]

Forward Substitution Snippet:

A computer screen shot of a program code

AI-generated content may be incorrect.

**Inputs of Function**

1. n: The size of the square matrix L and vectors x and b
2. L: The lower triangular matrix
3. b: The right-hand side vector

The function returns a pointer to the solution vector **x**.

**Memory Allocation:**

This section dynamically allocates memory for the solution vector x. It includes error checking to ensure the allocation was successful.

**Main Algorithm Loop:**

The algorithm uses two nested loops:

1. Outer loop i: Iterates from 0 to n-1, representing each row of L and element of x
2. Inner loop j: Iterates from 0 to i-1, used for the summation in each step

**Algorithm Steps:**

For each i from 0 to n-1:

1. Initialize x[i] with b[i]
2. Subtract the sum of products L[i][j] \* x[j] for all j from 0 to i-1
3. The result is the value of x[i]

Backward Substitution Snippet:

A computer screen shot of a program code

AI-generated content may be incorrect.

**Inputs For functions**

* **n**: The size of the square matrix U and vectors x and b
* **U**: The upper triangular matrix
* **b**: The right-hand side vector

The function returns a pointer to the solution vector **x**.

**Memory Allocation:**

This section dynamically allocates memory for the solution vector x. It includes error checking to ensure the allocation was successful.

**Main Algorithm Loop:**

The algorithm uses two nested loops:

1. Outer loop i: Iterates from n-1 down to 0, representing each row of U and element of x
2. Inner loop j: Iterates from i+1 to n-1, used for the summation in each step

**Algorithm Steps:**

For each i from n-1 down to 0:

1. Initialize x[i] with b[i]
2. Subtract the sum of products U[i][j] \* x[j] for all j from i+1 to n-1
3. Divide the result by the diagonal element U[i][i] to get the final value of x[i]

**Input Function snippet:**

A computer screen shot of a program code

AI-generated content may be incorrect.

**Inputs For Function:**

* n: The size of the square matrix A and vector b
* A: A pointer to a 2D array to store the matrix
* b: A pointer to an array to store the right-hand side vector

**Function Explanation:**

***FILE* \*file = fopen("input.txt", "r");**

**if (file == NULL) {**

**fprintf(stderr, "File not found\n");**

**exit(EXIT\_FAILURE);**

**}**

This section opens the file "input.txt" in read mode. If the file cannot be opened (e.g., it doesn't exist), the program prints an error message and exits.

**Reading Matrix A:**

**for(int i=0; i<*n*; i++){**

**for(int j=0; j<*n*; j++){**

**if (fscanf(file, "%lf", &*A*[i][j]) != 1) {**

**fprintf(stderr, "Invalid input\n");**

**fclose(file);**

**exit(EXIT\_FAILURE);**

**}**

**}**

**}**

This nested loop reads n² double values from the file into matrix A. It uses fscanf to read each value, checking if the read operation was successful. If not, it prints an error message, closes the file, and exits.

**Reading Vector b:**

**for(int i=0; i<*n*; i++){**

**if (fscanf(file, "%lf", &*b*[i]) != 1) {**

**fprintf(stderr, "Invalid input\n");**

**fclose(file);**

**exit(EXIT\_FAILURE);**

**}**

**}**

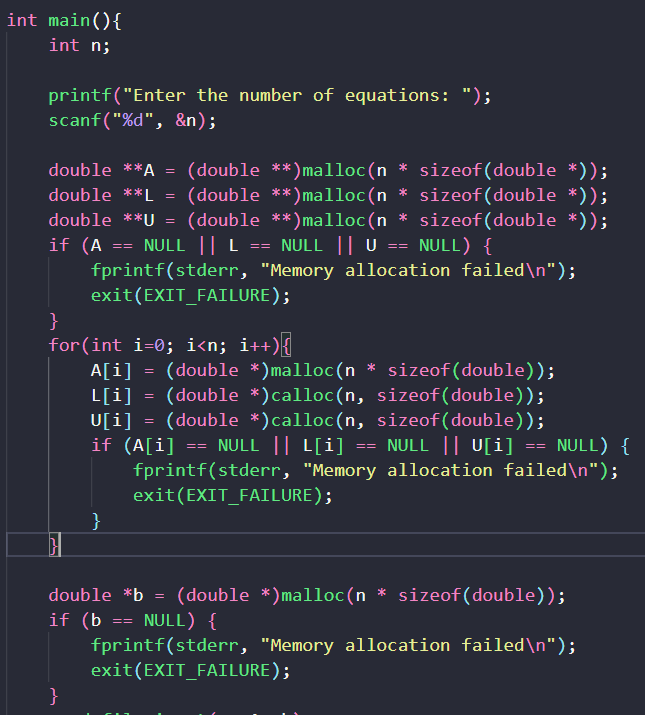
This loop reads n double values from the file into vector b, using the same error checking as for matrix A.

**File Closing:**

**fclose(file);**

After reading all required data, the function closes the file.

**Main Function (Memory Allocation part):**



**Function Explanation:**

**User Input:**

int n;

    printf("Enter the number of equations: ");

    scanf("%d", &n);

This section prompts the user to enter the number of equations (which is also the size of the square matrix) and stores it in the variable n.

**Memory Allocation:**

Allocating Memory for Matrices:

double \*\*A = (double \*\*)malloc(n \* sizeof(double \*));

    double \*\*L = (double \*\*)malloc(n \* sizeof(double \*));

    double \*\*U = (double \*\*)malloc(n \* sizeof(double \*));

    if (A == NULL || L == NULL || U == NULL) {

        fprintf(stderr, "Memory allocation failed\n");

        exit(EXIT\_FAILURE);

    }

This code allocates memory for three 2D arrays: A (input matrix), L (lower triangular matrix), and U (upper triangular matrix). It uses malloc to allocate an array of pointers for each matrix.

**Allocating Memory for Matrix Rows**

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

        A[i] = (double \*)malloc(n \* sizeof(double));

        L[i] = (double \*)calloc(n, sizeof(double));

        U[i] = (double \*)calloc(n, sizeof(double));

        if (A[i] == NULL || L[i] == NULL || U[i] == NULL) {

            fprintf(stderr, "Memory allocation failed\n");

            exit(EXIT\_FAILURE);

        }

    }

    double \*b = (double \*)malloc(n \* sizeof(double));

    if (b == NULL) {

        fprintf(stderr, "Memory allocation failed\n");

        exit(EXIT\_FAILURE);

    }

This loop allocates memory for each row of the matrices:

* For A, it uses malloc to allocate uninitialized memory.
* For L and U, it uses calloc to allocate memory initialized to zero.
* For b, it uses malloc to allocate uninitialized memory.

Error Handling

After each memory allocation, the code checks if the allocation was successful. If not, it prints an error message and exits the program.

**Main Function Cont.:**

A screenshot of a computer program

AI-generated content may be incorrect.

**Reading Input**

read\_file\_input(n, A, b);

This calls the read\_file\_input function to populate matrix A and vector b from a file.

LU Decomposition:

LU\_Decomposition(n, A, L, U);

This performs the LU decomposition on matrix A, storing the results in L and U.

Printing L and U Matrices

printf("The Lower Triangular Matrix is: \n");

**for**(**int** i=0; i<n; i++){

**for**(**int** j=0; j<n; j++){

printf("%lf ", L[i][j]);

}

printf("\n");

}

printf("The Upper Triangular Matrix is: \n");

**for**(**int** i=0; i<n; i++){

**for**(**int** j=0; j<n; j++){

printf("%lf ", U[i][j]);

}

printf("\n");

}

These nested loops print the L and U matrices to the console.

Solving the Linear System

**double** \*y = Forward\_Substitution(n, L, b);

**double** \*x = Backward\_Substitution(n, U, y);

This solves the linear system Ax = b using the LU decomposition:

1. Solve Ly = b using forward substitution
2. Solve Ux = y using backward substitution

**Printing the Solution**

**for**(**int** i=0; i<n; i++){

printf("The value of x[%d] is: %lf\n", i+1, x[i]);

}

This loop prints the solution vector x.

**Memory Deallocation**

The last section frees all dynamically allocated memory to prevent memory leaks.

Solution

**L and U matrix :**

A screenshot of a computer

AI-generated content may be incorrect.

X Vector:

A screenshot of a computer screen

AI-generated content may be incorrect.