Table of Contents

Getting Started: 2

Step-by-step walk through of the algorithm. 2

Example (1) 3

Step1: System of Linear Equations 3

Step2: Transforming into Matrix Form: 3

Step3: LU Decomposition 3

Step4: Solving Ly = b for y 4

Step5: Solving Ux = y for x 4

Step6: Verifying the answers: 5

Example (2): 5

Step1: System of Linear Equations 5

Step2: Transforming into Matrix Form: 5

Step3: LU Decomposition 5

Step4: Solving Ly = b for y 6

Step5: Solving Ux = y for x 6

Step6: Verifying the answers: 6

Coding a simple serial program to do this calculation: 7

Testing the program: 8

Version 0.1: 8

Version 0.2: 9

Version 0.3: 9

Running the program serially: 11

Next steps: 11

References: 12

Appendix – 1 (Step by step solution of example 1 using LU Decomposition) 13

Appendix –2 (Basic Example code to solve a 3x3 matrix using LU Decomposition) 16

Getting Started:

First, we started a git repo and created directories with file structures that could be useful in our project. We may not end up using all those files, but this structure will be helpful in enforcing modularity into our program.

```
Parallel Equation Solver
│
├── src/                   # Source files
│   ├── main.c             # Main program entry point
│   ├── lu_serial.c        # Serial LU Decomposition implementation
│   ├── lu_serial.h        # Header for serial LU Decomposition
│   ├── lu_parallel.c      # Parallel LU Decomposition implementation using Pthreads
│   ├── lu_parallel.h      # Header for parallel LU Decomposition
│   └── utilities.c        # Utility functions for matrix operations and timing
│
├── include/               # Header files
│   ├── matrix.h           # Definitions and functions for matrix operations
│   └── timing.h           # Timing and performance measurement utilities
│
├── docs/                  # Documentation files
│   ├── setup.md           # Setup instructions
│   ├── usage.md           # Usage instructions
│   └── development.md     # Notes on development decisions and project structure
│
├── tests/                 # Test files
│   ├── test_serial.c      # Tests for serial implementation
│   └── test_parallel.c    # Tests for parallel implementation
│
├── benchmarks/            # Benchmark scripts and results
│   ├── benchmark_script.sh # Script to run benchmarks
│   └── results.md         # Benchmark results and analysis
│
├── Makefile               # Makefile for building the project
└── README.md              # Project overview and general instructions
```

Before we start any actual coding, we made sure to review how the actual LU Decomposition works, and just see how in general a set of linear equations is transferred into a matrix that's to be decomposed and solved using LU Decomposition.

Step-by-step walk through of the algorithm.

(Making sense of the program we're making): Those examples will be used to code our initial algorithm and verify its working as expected.

Example (1)

Step1: System of Linear Equations

Let's start with the following system of equations:

$$\begin{aligned} 2x_1 + 3x_2 - x_3 &= 5 \\ 4x_1 + x_2 + 2x_3 &= 6 \\ -2x_1 + 2x_2 + 3x_3 &= 8 \end{aligned}$$

Step2: Transforming into Matrix Form:

This system can be represented in matrix form as Ax=b, where:

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix}$$

Step3: LU Decomposition

Next, we decompose A into L and U, where L is a lower triangular matrix and U is an upper triangular matrix.

$$\begin{bmatrix} A00 & A01 & A02 \\ A10 & A11 & A12 \\ A20 & A21 & A22 \end{bmatrix} = \overbrace{\begin{bmatrix} 1 & 0 & 0 \\ L10 & 1 & 0 \\ L20 & L21 & 1 \end{bmatrix}}^{\text{Lower Triangular}} \underbrace{\begin{bmatrix} U00 & U01 & U02 \\ 0 & U11 & U12 \\ 0 & 0 & U22 \end{bmatrix}}_{\text{Upper Triangular}}$$

In our example:
(Step by step derivations of the LU matrices in the appendix)

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 2 & 3 & -1 \\ 0 & -5 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

Step4: Solving Ly = b for y

Given L and b, we solve for y (which is an intermediate vector, not the final solution). Using forward substitution.

$$Ly = b, \qquad \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix}$$

$y_1 = 5$ (from the first row of L and b)

$2y_1 + y_2 = 6, \qquad 2 \times 5 + y2 = 6, \qquad y2 = -4$

$-y1 - y2 + y3 = 8, \qquad -5 - (-4) + y3 = 8, \qquad y3 = 9$

So

$$\therefore \; y = \begin{pmatrix} 5 \\ -4 \\ 9 \end{pmatrix},$$

Step5: Solving Ux = y for x

Now, with U and y, we can solve for x using backward substitution.

$$Ux = y, \qquad \begin{pmatrix} 2 & 3 & -1 \\ 0 & -5 & 4 \\ 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -4 \\ 9 \end{pmatrix}$$

$6x_3 = 9, \qquad \therefore x_3 = \frac{3}{2}, \qquad$ (from the last row of U and y)

$-5x_2 + 4x_3 = -4, \qquad -5x_2 + 4 \times \left(\frac{3}{2}\right) = -4, \qquad x_2 = 2$

$2x_1 + 3x_2 - x_3 = 5, \qquad 2x_1 + 3 \times 2 - \frac{3}{2} = 5, \qquad x_1 = \frac{1}{4}$

$$\therefore \; x = \begin{pmatrix} \frac{1}{4} \\ 2 \\ \frac{3}{2} \end{pmatrix}, \qquad \therefore x_1 = \frac{1}{4}, \qquad x_2 = 2, \qquad x_3 = \frac{3}{2}$$

Step6: Verifying the answers:

$$2x_1 + 3x_2 - x_3 = 5$$

$$2\left(\frac{1}{4}\right) + 3(2) - \left(\frac{3}{2}\right) = 5, \qquad \#$$

$$4x_1 + x_2 + 2x_3 = 6$$

$$4\left(\frac{1}{4}\right) + (2) + 2\left(\frac{3}{2}\right) = 6, \qquad \#$$

$$-2x_1 + 2x_2 + 3x_3 = 8$$

$$-2\left(\frac{1}{4}\right) + 2(2) + 3\left(\frac{3}{2}\right) = 8, \qquad \#$$

Example (2):

Step1: System of Linear Equations

$$\begin{gathered} x_1 + x_2 - x_3 = 4 \\ x_1 - 2x_2 + 3x_3 = -6 \\ 2x_1 + 3_{x2} + x_3 = 7 \end{gathered}$$

Step2: Transforming into Matrix Form:

$$\begin{pmatrix} 1 & 1 & -1 \\ 1 & -2 & 3 \\ 2 & 3 & 1 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -6 \\ 7 \end{pmatrix}$$

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 1 & -2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \qquad b = \begin{pmatrix} 4 \\ -6 \\ 7 \end{pmatrix}$$

Step3: LU Decomposition

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & -\frac{1}{3} & 1 \end{pmatrix}, \qquad U = \begin{pmatrix} 1 & 1 & -1 \\ 0 & -3 & 4 \\ 0 & 0 & \frac{13}{3} \end{pmatrix}$$

Step4: Solving Ly = b for y

Given L and b, we solve for y (which is an intermediate vector, not the final solution). Using forward substitution.

$$Ly = b, \quad \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 2 & -\frac{1}{3} & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -6 \\ 7 \end{pmatrix}$$

$y_1 = 4$ (from the first row of L and b)
$y_1 + y_2 = -6, \quad y2 = -10$
$2y_1 - \frac{1}{3}y_2 + y_3 = 7, \quad y_3 = -\frac{13}{3}$

So

$$\therefore \; y = \begin{pmatrix} 4 \\ -10 \\ -\frac{13}{3} \end{pmatrix},$$

Step5: Solving Ux = y for x

Now, with U and y, we can solve for x using backward substitution.

$$Ux = y, \quad \begin{pmatrix} 1 & 1 & -1 \\ 0 & -3 & 4 \\ 0 & 0 & \frac{13}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 4 \\ -10 \\ -\frac{13}{3} \end{pmatrix}$$

$\frac{13}{3}x_3 = -\frac{13}{3}, \quad \therefore x_3 = -1,$ (from the last row of U and y)
$-3x_2 + 4x_3 = -10, \quad x_2 = 2$
$x_1 + x_2 - x_3 = 4, \quad x_1 = 1$

$$\therefore \; x = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix}, \qquad \therefore x_1 = 1, \quad x_2 = 2, \quad x_3 = -1$$

Step6: Verifying the answers:

$$x_1 + x_2 - x_3 = 4$$
$$x_1 - 2x_2 + 3x_3 = -6$$
$$2x_1 + 3_{x2} + x_3 = 7$$

$$x_1 + x_2 - x_3 = 4$$

$$(1) + (2) - (-1) = 4, \qquad \#$$

$$x_1 - 2x_2 + 3x_3 = -6$$
$$(1) - 2(2) + 3(-1) = -6, \qquad \#$$

$$2x_1 + 3_{x2} + x_3 = 7$$
$$2(1) + 3(2) + (-1) = 7, \qquad \#$$

Coding a simple serial program to do this calculation:

To get started, we will code a simple C code implementing this algorithm, then we'll use those same examples above to verify the correctness of that algorithm. Next, we'll improve the algorithm to work on bigger matrices, and will use pthreads.
Code structure:

```
.gitignore M    test.c
test.c > ...
// Author: Mohamed Ghonim
// Created: 02/18/2024
// Last Modified: 02/18/2024
// Functionality: Perform LU decomposition and solve a system of linear equations using forward and backward substitution.
// Version: 0.1

#include <stdio.h>
#define N 3 // Size of the matrix (3x3)

// Function to perform LU decomposition
void luDecomposition(double A[N][N], double L[N][N], double U[N][N]) {…

// Function to solve the equation Ly = b
void forwardSubstitution(double L[N][N], double b[N], double y[N]) {…

// Function to solve the equation Ux = y
void backwardSubstitution(double U[N][N], double y[N], double x[N]) {…

int main() {…
```

The matrix is defined in the main here, as a next step, we'll have It defined in a separate file or function. (The complete code is in appendix 2).

```
> gcc -o bin/simple src/simple.c
> ./bin/simple
Solution:
x[0] = 0.250000
x[1] = 2.000000
x[2] = 1.500000
```

Testing the program:

Version 0.1:

First example:
The first example we tried was this matrix:

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -2 & 2 & 3 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \qquad b = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix}$$

And the answer was:

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \frac{1}{4} \\ 2 \\ \frac{3}{2} \end{pmatrix} = \begin{pmatrix} 0.25 \\ 2 \\ 1.5 \end{pmatrix}$$

When we give the program this matrix:

```
int main() {
    double A[N][N] = {{2, 3, -1}, {4, 1, 2}, {-2, 2, 3}};
    double b[N] = {5, 6, 8};
```

We get:

```
> ./bin/simple
Solution:
x[0] = 0.250000
x[1] = 2.000000
x[2] = 1.500000
```

Which is the expected solution.

Second example:

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 1 & -2 & 3 \\ 2 & 3 & 1 \end{pmatrix}, \qquad x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}, \qquad b = \begin{pmatrix} 4 \\ -6 \\ 7 \end{pmatrix}$$

And the answer was:

$$x = \begin{pmatrix} 1 \\ 2 \\ -1 \end{pmatrix},$$

When we give the program this matrix:

```
    double A[N][N] = {{1, 1, -1}, {1, -2, 3}, {2, 3, 1}};
    double b[N] = {4, -6, 7};
```

We get:

```
> gcc -o bin/simple src/simple.c
> ./bin/simple
Solution:
x[0] = 1.000000
x[1] = 2.000000
x[2] = -1.000000
```

Which is the expected solution.

Version 0.2:

We then made the program more structured but adding the matrices in the /matrices directory as text files passed to the program using the command-line terminal as the first argument.

For example:

```
> ./bin/simple matrices/3x3_1.txt
Solution:
x[0] = 0.250000
x[1] = 2.000000
x[2] = 1.500000
```

and

```
> ./bin/simple matrices/3x3_2.txt
Solution:
x[0] = 1.000000
x[1] = 2.000000
x[2] = -1.000000
```

Version 0.3:

Now we parameterized the size of the matrix, such that it's passed to the program from the first line in the matrix txt file.

```
// Version: 0.3 : readin the matrix from a file passed as an argument to the program
//              : added a function to free the allocated memory
//              : the matrix size is parameterized, and passed as the first line in
the matrix file
```

We tested the program on multiple 3x3 and 4x4 matrices which we know the answers to previously as a way to debug the code. The program solved all the matrices correctly.

To get and test much bigger matrices, we wrote a python script <utilities/matrix_generator.py> which takes in an argument of the dimension of nodes, and generates a matrix which gates saved in the <matrices/py_generated> directory with $matrix_dimension.txt tile.

```
# matrix_generator.py
# Author: Mohamed Ghonim
# Created: 02/18/2024
```

```
# Last Modified: 02/18/2024
# Function: This script generates a random matrix of a given dimension and saves it to
a file in matrices/py_generated
# Usage: python matrix_generator.py <dimension>
# version: 0.1

import numpy as np
import sys
import os

def generate_matrix_and_save(dimension):

    # Define the target directory
    target_directory = os.path.join(os.path.dirname(__file__),
"../matrices/py_generated")
    #target_directory = "../matrices"

    # Ensure the target directory exists
    os.makedirs(target_directory, exist_ok=True)

    # Generating a dimension x dimension matrix with random integers between -10 and
10
    matrix = np.random.randint(-10, 10, size=(dimension, dimension))

    # Generating the final row with random integers between -10 and 10
    final_row = np.random.randint(-10, 10, size=(dimension,))

    # Preparing the string representation and defining the file path
    matrix_str = f"{dimension}\n" + "\n".join(" ".join(map(str, row)) for row in
matrix) + "\n" + " ".join(map(str, final_row))
    file_name = os.path.join(target_directory, f"{dimension}x{dimension}.txt")

    # Saving to file
    with open(file_name, 'w') as file:
        file.write(matrix_str)

    print(f"Matrix saved to {file_name}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Usage: python script.py <dimension>")
        sys.exit(1)

    try:
        dimension = int(sys.argv[1])
        if dimension <= 0:
            raise ValueError
        generate_matrix_and_save(dimension)
```

```
    except ValueError:
        print("Please provide a valid positive integer for the matrix dimension.")
```

To further enhance the code, we are not allowing for the user to specify an output file where the solution will get saved, if no output file is specified, the solution gets printed on the terminal.

```
// main.c
// Author: Mohamed Ghonim
// Created: 02/18/2024
// Last Modified: 02/18/2024
// Functionality: Perform LU decomposition and solve a system of linear equations
using forward and backward substitution.
// Version: 0.3 : readin the matrix from a file passed as an argument to the program
//              : added a function to free the allocated memory
//              : the matrix size is parameterized, and passed as the first line in
the matrix file

// Version: 0.4 : allow write the solution to a file passed as an argument to the
program
//              : if no file is passed, the solution will be printed to the standard
output
//              : example usage: matrices/py_generated/5000x5000.txt
matrices_solution/5000x5000.txt
```

Running the program serially:

Currently, we have a functional serial program that takes in a matrix of dimensions N and solves it using LU decomposition and saves the solution to a text file or prints it on the screen.

Next steps:

At this point, we need to focus on parallelizing the program, so that we can use pthreads to run it on multiple cores.
We also need to implement a way to measure the time it takes to find a solution, so that we can use that to collect benchmarks and compare the serial performance to the parallel performance.
We will then use perl or python to automate running the program once serially, then in parallel on multiple processors from 1 to 16, and produce an output file in the benchmark directory with the time it takes to find the solutions, as well as the speedup achieved.

References:

[1] K. Hartnett and substantive Quanta Magazine moderates comments to facilitate an informed, “New algorithm breaks speed limit for solving linear equations,” Quanta Magazine, https://www.quantamagazine.org/new-algorithm-breaks-speed-limit-for-solving-linear-equations-20210308/ (accessed Jan. 28, 2024).

[2] T. J. Dekker, W. Hoffmann, and K. Potma, “Parallel algorithms for solving large linear systems,” *Journal of Computational and Applied Mathematics*, vol. 50, no. 1–3, pp. 221–232, 1994. doi:10.1016/0377-0427(94)90302-6

[3] W. by: Q. Chunawala, “Fast algorithms for solving a system of linear equations,” Baeldung on Computer Science, https://www.baeldung.com/cs/solving-system-linear-equations (accessed Jan. 28, 2024).

[4] D. Kaya and K. Wright, “Parallel algorithms for LU decomposition on a shared memory multiprocessor,” *Applied Mathematics and Computation*, vol. 163, no. 1, pp. 179–191, Apr. 2005. doi:10.1016/j.amc.2004.01.027

[5] E. E. Santos and M. Muralcetharan, “Analysis and Implementation of Parallel LU-Decomposition with Different Data layouts,” *University of California, Riverside*, Jun. 2000.

Appendix – 1 (Step by step solution of example 1 using LU Decomposition)

$$A = \begin{pmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -2 & 2 & 3 \end{pmatrix}$$

$$= \begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & -1 \\ 4 & 1 & 2 \\ -2 & 2 & 3 \end{pmatrix} \quad \begin{matrix} II-(2)I \\ III-(-1)I \end{matrix}$$

$$= \begin{pmatrix} 1 & & \\ 2 & 1 & \\ -1 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & -1 \\ 0 & -5 & 4 \\ 0 & 5 & 2 \end{pmatrix} \quad III-(-1)II$$

$$= \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 3 & -1 \\ 0 & -5 & 4 \\ 0 & 0 & 6 \end{pmatrix}$$

L matrix U matrix

Solve $Ly = b$ for y

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}, \quad y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix}$$

$$\therefore \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} 5 \\ 6 \\ 8 \end{pmatrix}$$

① $y_1 = 5$

② $2y_1 + y_2 = 6 \qquad \therefore y_2 = 6 - 2(5) = -4$

③ $-y_1 - y_2 + y_3 = 8 \qquad \therefore y_3 = 8 + (5) + (-4) = \underline{\underline{9}}$

$$\therefore y = \begin{pmatrix} 5 \\ -4 \\ 9 \end{pmatrix}$$

Solve $Ux = y$ for x

$$\begin{pmatrix} 2 & 3 & -1 \\ 0 & -5 & 4 \\ 0 & 0 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 5 \\ -4 \\ 9 \end{pmatrix}$$

① $2x_1 + 3x_2 - x_3 = 5 \qquad \therefore 2x_1 = 5 + \frac{3}{2} - 3(2) = \frac{1}{4}$ *

② $-5x_2 + 4x_3 = -4 \qquad \therefore 5x_2 = 4 - 4\left(\frac{3}{2}\right), \; x_2 = 2$

③ $6x_3 = 9 \qquad \therefore x_3 = \frac{3}{2}$

Now we know that $x_1 = x$, $x_2 = y$, $x_3 = z$

$x_1 = \frac{1}{4}$

$x_2 = 2$

$x_3 = \frac{3}{2}$

Let's substitute back in the eluations.

① $2x_1 + 3x_2 - x_3 = 5$

$$2\left(\frac{1}{4}\right) + 3(2) - \left(\frac{3}{2}\right) = 5 \quad \#$$

② $4x_1 + x_2 + 2x_3 = 6$

$$4\left(\frac{1}{4}\right) + (2) + 2\left(\frac{3}{2}\right) = 6 \quad \#$$

③ $-2x_1 + 2x_2 + 3x_3 = 8$

$$-2\left(\frac{1}{4}\right) + 2(2) + 3\left(\frac{3}{2}\right) = 8 \quad \#$$

Appendix –2 (Basic Example code to solve a 3x3 matrix using LU Decomposition)

```
// Author: Mohamed Ghonim
// Created: 02/18/2024
// Last Modified: 02/18/2024
// Functionality: Perform LU decomposition and solve a system of linear equations
using forward and backward substitution.
// Version: 0.1


#include <stdio.h>
#define N 3 // Size of the matrix (3x3)

// Function to perform LU decomposition
void luDecomposition(double A[N][N], double L[N][N], double U[N][N]) {
    int i, j, k;
    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++) {
            if (j < i)
                L[j][i] = 0;
            else {
                L[j][i] = A[j][i];
                for (k = 0; k < i; k++) {
                    L[j][i] = L[j][i] - L[j][k] * U[k][i];
                }
            }
        }
        for (j = 0; j < N; j++) {
            if (j < i)
                U[i][j] = 0;
            else if (j == i)
                U[i][j] = 1;
            else {
                U[i][j] = A[i][j] / L[i][i];
                for (k = 0; k < i; k++) {
                    U[i][j] = U[i][j] - ((L[i][k] * U[k][j]) / L[i][i]);
                }
            }
        }
    }
}

// Function to solve the equation Ly = b
void forwardSubstitution(double L[N][N], double b[N], double y[N]) {
    for (int i = 0; i < N; i++) {
        y[i] = b[i];
        for (int j = 0; j < i; j++) {
```

```
            y[i] -= L[i][j] * y[j];
        }
        y[i] = y[i] / L[i][i];
    }
}

// Function to solve the equation Ux = y
void backwardSubstitution(double U[N][N], double y[N], double x[N]) {
    for (int i = N - 1; i >= 0; i--) {
        x[i] = y[i];
        for (int j = i + 1; j < N; j++) {
            x[i] -= U[i][j] * x[j];
        }
        // No division by U[i][i] since U[i][i] = 1
    }
}

int main() {
    double A[N][N] = {{2, 3, -1}, {4, 1, 2}, {-2, 2, 3}};
    double b[N] = {5, 6, 8};
    double L[N][N] = {0};
    double U[N][N] = {0};
    double y[N] = {0};
    double x[N] = {0};

    luDecomposition(A, L, U);
    forwardSubstitution(L, b, y);
    backwardSubstitution(U, y, x);

    printf("Solution: \n");
    for (int i = 0; i < N; i++) {
        printf("x[%d] = %f\n", i, x[i]);
    }

    return 0;
}
```