Table of Contents

[Getting Started: 2](#_Toc159168336)

[Step-by-step walk through of the algorithm. 3](#_Toc159168337)

[Example (1) 3](#_Toc159168338)

[Step1: System of Linear Equations 3](#_Toc159168339)

[Step2: Transforming into Matrix Form: 3](#_Toc159168340)

[Step3: LU Decomposition 3](#_Toc159168341)

[Step4: Solving Ly = b for y 4](#_Toc159168342)

[Step5: Solving Ux = y for x 4](#_Toc159168343)

[Step6: Verifying the answers: 5](#_Toc159168344)

[Example (2): 5](#_Toc159168345)

[Step1: System of Linear Equations 5](#_Toc159168346)

[Step2: Transforming into Matrix Form: 5](#_Toc159168347)

[Step3: LU Decomposition 5](#_Toc159168348)

[Step4: Solving Ly = b for y 6](#_Toc159168349)

[Step5: Solving Ux = y for x 6](#_Toc159168350)

[Step6: Verifying the answers: 7](#_Toc159168351)

[Coding a simple serial program to do this calculation: 7](#_Toc159168352)

[Testing the program: 8](#_Toc159168353)

[Version 0.1: 8](#_Toc159168354)

[Version 0.2: 9](#_Toc159168355)

[Version 0.3: 9](#_Toc159168356)

[Next steps: 10](#_Toc159168357)

[Appendix – 1 (Step by step solution of example 1 using LU Decomposition) 10](#_Toc159168358)

[Appendix –2 (Basic Example code to solve a 3x3 matrix using LU Decomposition) 13](#_Toc159168359)

# 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.

A screenshot of a computer program

Description automatically generated

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:

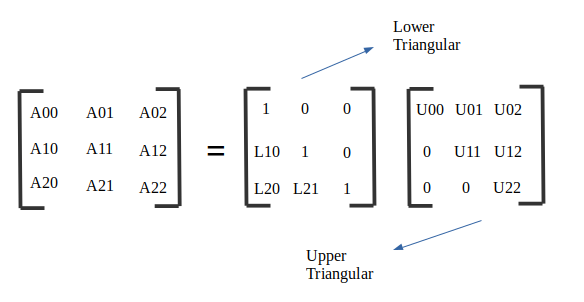
­­­

## Step2: Transforming into Matrix Form:

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

## 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.



In our example:

(Step by step derivations of the LU matrices in the appendix)

## 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.

## Step5: Solving Ux = y for x

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

## Step6: Verifying the answers:

# Example (2):

## Step1: System of Linear Equations

## Step2: Transforming into Matrix Form:

## Step3: LU Decomposition

## 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.

## Step5: Solving Ux = y for x

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

## Step6: Verifying the answers:

# 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:

A screenshot of a computer program

Description automatically generated

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).

A black screen with white text and numbers

Description automatically generated

# Testing the program:

### Version 0.1:

First example:

The first example we tried was this matrix:

And the answer was:

When we give the program this matrix:

A black background with colorful text

Description automatically generated with medium confidence

We get:

A screen shot of a computer

Description automatically generated

Which is the expected solution.

Second example:

And the answer was:

When we give the program this matrix:



We get:

A screen shot of a computer

Description automatically generated

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.

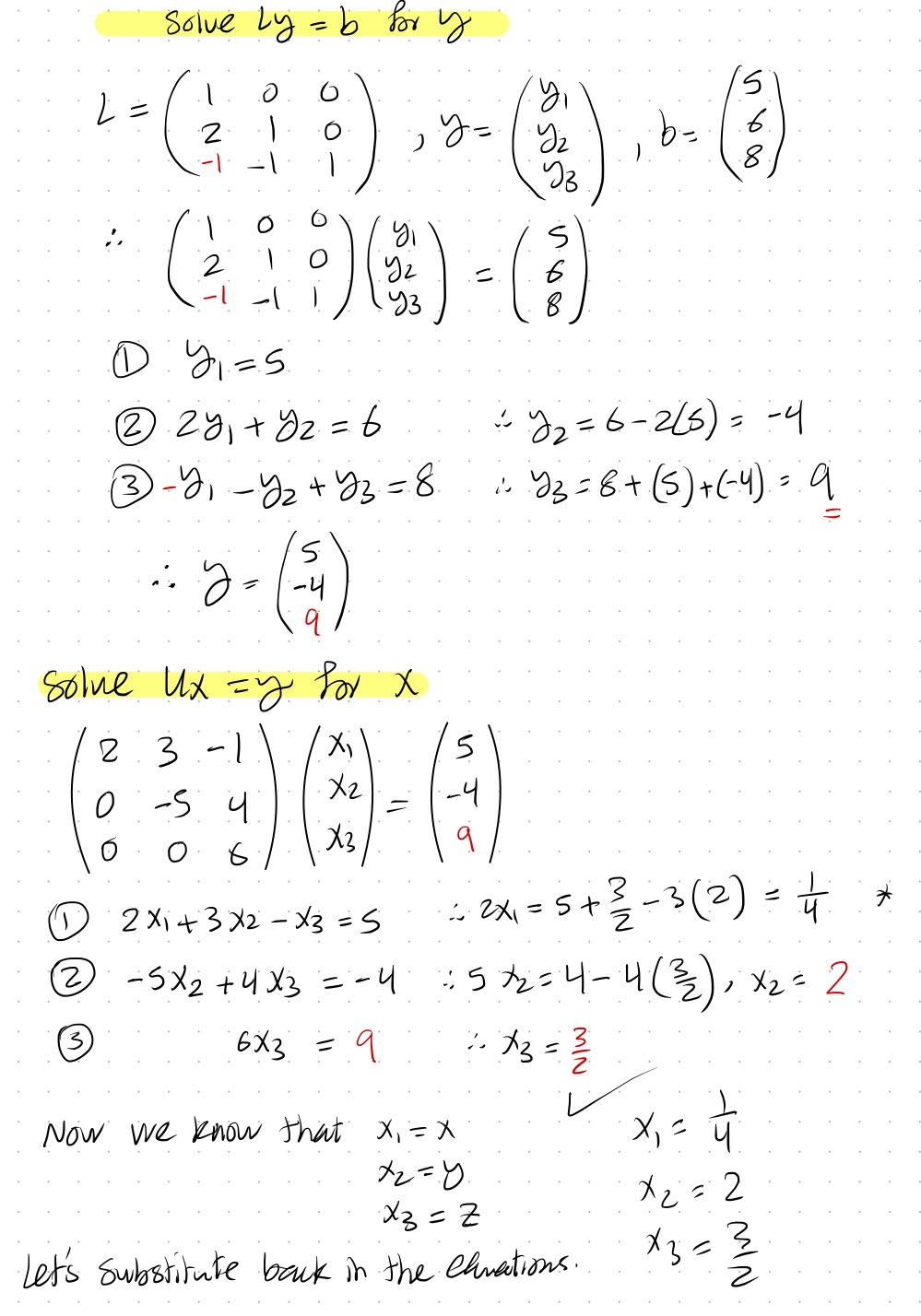
We also need to implement a way to measure the time it takes to find a solution, so that we can use that to collect benchmarks and compare the serial performance to the parallel performance.

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.

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

A group of math equations

Description automatically generated with medium confidence



A white paper with black text

Description automatically generated

# 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;

}