Assignment 3

Solving System of Linear Equations

Jacob Foster

1 Introduction

For this assignment, I was required to write a program that can read a symmetric nxn matrix "A" from the contents of a Matrix Market (.mtx) file and convert it into Compressed Sparse Row (CSR) format. Once the matrix is in CSR format, it is then multiplied by a vector "x" in which the product of the matrix "A" and the vector "x" is a a 1xn matrix "b" with entries of all ones. An iterative solving method is implemented to solve for all entries of "x". Finally, the residual norm of the solution "x" is computed to check accuracy.

2 Results

Table 1: Results

Problem	Size		Non-zeros	CPU time (Sec)	Norm-Residual
	Row	*Column*			
LFAT5.mtx	14	14	30	0.029000	2.66e-14
LF10.mtx	18	18	50	0.070000	6.01e-13
ex3.mtx	1821	1821	27253	600.245000	29.9345
jnlbrng1.mtx	40000	40000	119600	1.598000	5.28e-9
ACTIVSg70K.mtx	69999	69999	154313	0.403000	nan
2cubes sphere.mtx	101492	101492	874378	-	-
tmt sym.mtx	726713	726713	2903837	-	-
StocF-1465.mtx	1465137	1465137	11235263	-	-

*Note that LFAT5 and LF10 were solved using the paramter max_iterations = 100000, and jnlbrng1 was solved using max_iterations = 10000.

3 Report

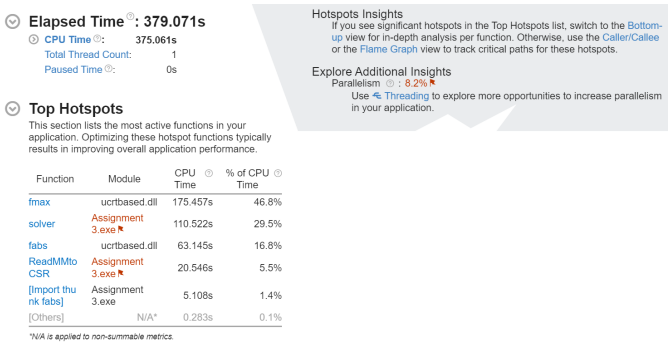
3.1 Solving Method Analysis

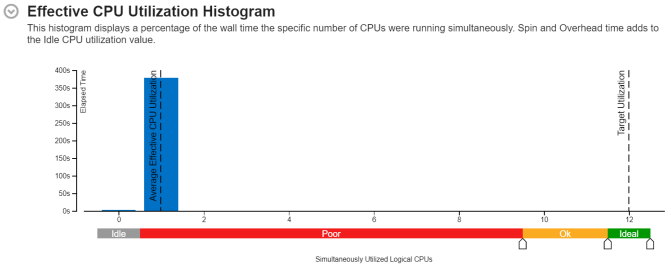
To solve the linear system I used an altered variation of the Jacobi iterative method. The Jacobi method works by rearranging each equation in the linear system to be in terms of the x value

along the main diagonal at that row. An initial guess of 0 is made for all x values in x, and the old values of x will be replaced with new values of x. As this process is repeated, the x values will converge to values infinitesimally close to the solution of the system. In this code, the x values stop converging when the difference in x values between iterations reaches the tolerance criteria, to ensure the loop will converge.

3.2 Vtune

Using the Vtune profiler, the code's performance on the jnlbrng1.mtx file can be analyzed and displayed this the following figures:





Looking at these graphs, it is very evident that the solving method I used was not very efficient. This is due to the solving method I chose to implement.Since I used an iterative method, solving matrices becomes increasingly harder as the size of the matrix increases. Furthermore, I was not happy with the original algorithms accuracy, so I implemented a new process that reassigns the solution vector an an extra n times to increase accuracy. This decreased the residual norm by two orders of magnitude, but also increased the amount of time used by the solver.

3.3 gcov

Using gcov to analyze my code, the following was outputted in the terminal:

```
File 'main.c'
Lines executed:93.55% of 31
Creating 'main.c.gcov'

File 'functions.c'
Lines executed:86.17% of 94
Creating 'functions.c.gcov'

Lines executed:88.00% of 125
```

This displays that there is excess code in my functions file that could be removed to further optimize my code, and that almost all the lines in my main function are used apart from some error handling.

3.4 Plots

3.4.1 Graphing code

To create the matrix sparsity plots, I created a script in python that reads the Matrix Market file, expands it symmetrically, and then plots the indices of all of its entries on a Cartesian plane. This can be used to visualize the matrix and confirm symmetry. The script is attached below:

```
import matplotlib.pyplot as plt
from scipy.io import mmread

# Read the Matrix Market file
matrix = mmread('LFAT5.mtx')

# Extract the row and column indices
row_ind, col_ind = matrix.nonzero()

# Plot the non-zero entries
plt.scatter(row_ind, col_ind, marker='s', color='blue', label='Non-
    zero entries')
```

```
# Customize the plot
plt.title('Matrix Sparcity Plot')
plt.xlabel('Column Index')
plt.ylabel('Row Pointer')
plt.gca().invert_yaxis()  # Invert the y-axis to keep the flip
plt.grid(False)
plt.show()
```

3.4.2 LFAT5.mtx

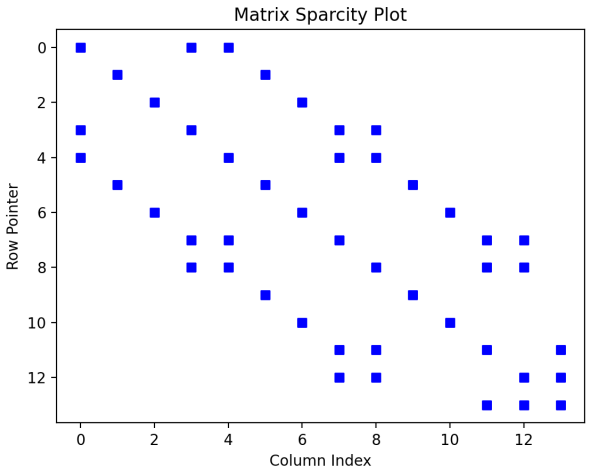


Figure 1: LFAT5 Sparsity Plot
This matrix converged and a solution with a minimal residual was found.

3.4.3 LF10.mtx

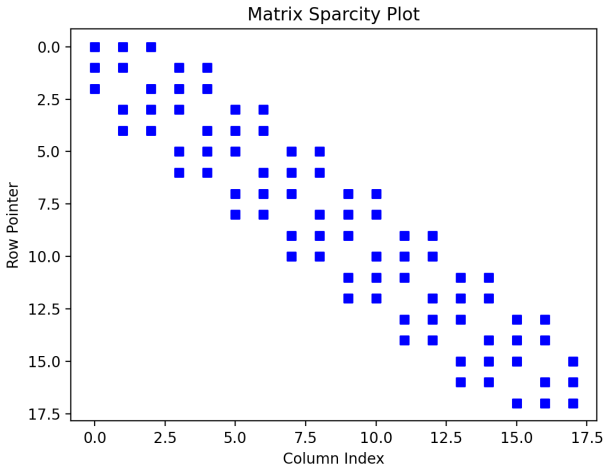


Figure 2: LF10 Sparsity Plot

This matrix converged and a solution with a minimal residual was found.

3.4.4 ex3.mtx

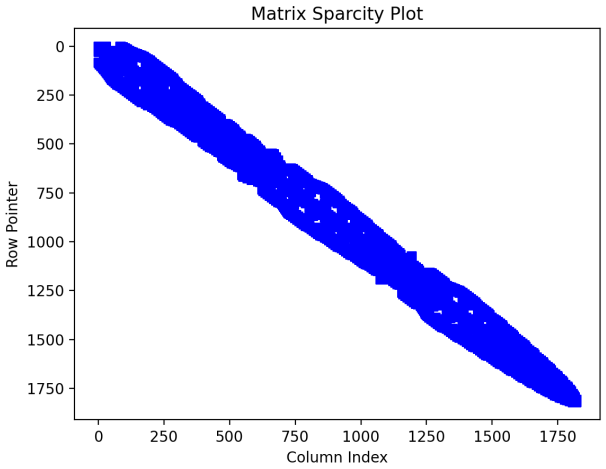


Figure 3: ex3 Sparsity Plot

This matrix converged and a solution with a minimal residual was found. With a larger number of iterations, it is fair to assume that the solution would converge. The number of iterations for this to be true is unreasonable for my computer, due to iterative limitations and code inefficiencies.

3.4.5 jnlbrng1.mtx

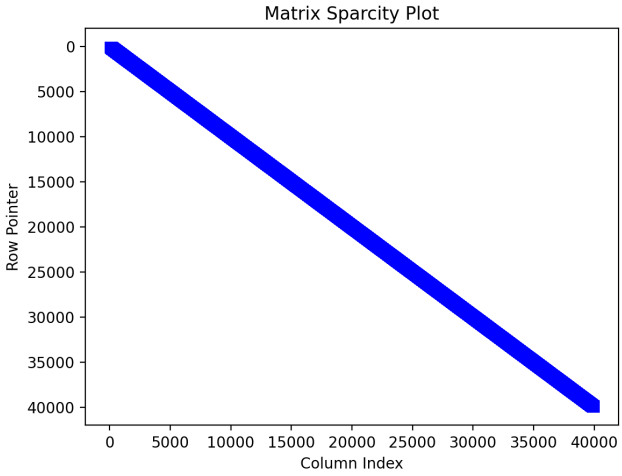


Figure 4: jnlbrng1 Sparsity Plot

This matrix did not converge due to iterative shortcomings.

3.4.6 ACTIVSg70K.mtx

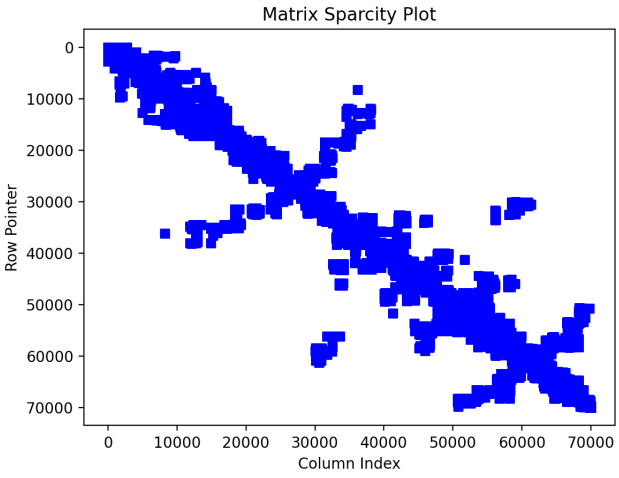


Figure 5: ACTIVSg70K Sparsity Plot

This matrix did not converge due to iterative shortcomings.

3.4.7 2cubes_sphere.mtx

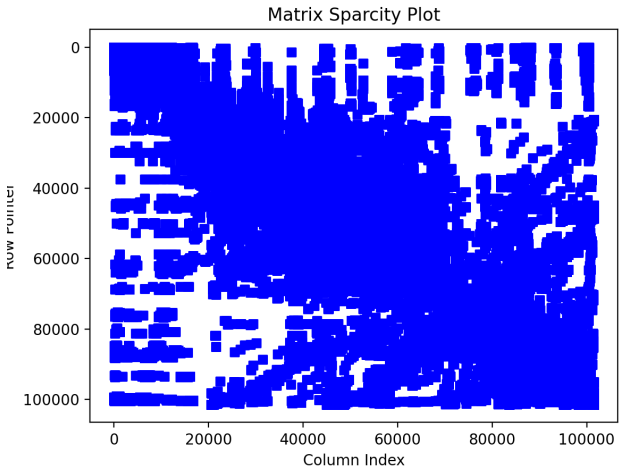


Figure 6: 2cubes_sphere Sparsity Plot

This matrix did not converge due to iterative shortcomings.

3.4.8 tmt_sym.mtx

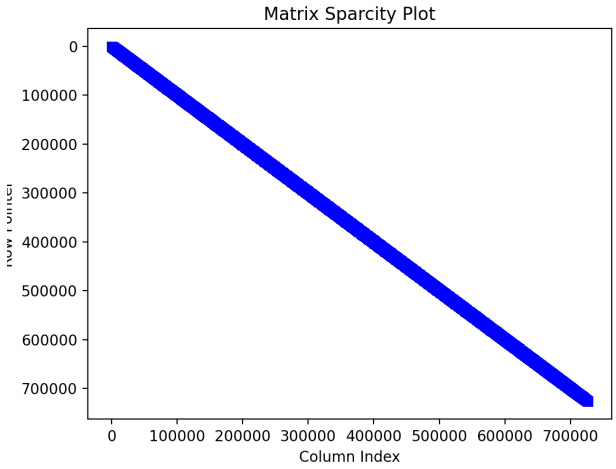


Figure 7: tmt_sym Sparsity Plot

This matrix did not converge due to iterative shortcomings.

3.4.9 StocF-1465.mtx

Unfortunately, this matrix was to large for my program to process without crashing my computer.This matrix did not converge due to iterative shortcomings.

3.5 Makefile

The makefile attached in this assignment's submission is essential to running the genetic sequencing algorithm in an autonomous manner. To run the makefile, firstly type "make clean" in the terminal to run and provide a clean reset, followed by "make" to run the file. The technical function of the file can be broken into sections as such. The variables CC, CFLAGS (-Wall, -Wextra, and -std=c99), SOURCES, OBJECTS, EXECUTABLE, and LIBS represent the GCC compiler, the compiling flags, the source code files, the object files, the executed file, and the imported libraries respectively. The executable file target declares the the default target as all, and is influenced by the object files target. To ensure the makefile successfully executes the executable function, it must be built with the same compiler as outlined in the file and link to all libraries and object files. Furthermore, it must follow the ".o: .c" format since it was programmed in the C language. This ensures that the compiler is able to successfully interpret the executable file. The last section of the code, "Clean", removes any excess object files that were previous compiled and ensure an uninterrupted execution. The makefile for this code sequence is attached below.

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99 -ggdb -fprofile-arcs -ftest-
   coverage
SOURCES = functions.c main.c
EXECUTABLE = main
LIBS = -lm

all: $(EXECUTABLE)

$(EXECUTABLE): $(SOURCES)
 $(CC) $(SOURCES) -o $(EXECUTABLE) $(LIBS)

%.o: %.c
 $(CC) $(CFLAGS) -c $< -o $@

clean:
 rm -f $(EXECUTABLE)
```