Assignment 3

Solving System of Linear Equations

Marcus De Maria

1 Introduction

I have to create a program to read a nxn matrix "A" from a Matrix Market (.mtx) file and convert it to Compressed Sparse Row (CSR) format for this project. The matrix is multiplied by a vector "x" once it has been converted to CSR format. The result of multiplying the matrix "A" by the vector "x" is a 1xn matrix "b" with all ones as entries. In order to verify accuracy, the residual norm of the solution "x" is finally calculated.

2 Results

For each of the required matrix mtx files, I compiled and ran my code with it to see how accurate my solver and other functions were. Some were much more accurate than others due to my solver, and others wouldn't run also do to the size of the matrix and ability of my current functions to run them.

Table 1: Results

Problem	Size		Non-zeros	CPU time (Sec)	Norm-Residual
	row	*column*			
LFAT5.mtx	14	14	30	0.000639	3.42e-12
LF10.mtx	18	18	50	0.011199	5.92e-11
ex3.mtx	1821	1821	27253	248.36	31.5
jnlbrng1.mtx	40000	40000	119600	1.75	1.27e-12
ACTIVSg70K.mtx	69999	69999	154313	0.392704	-nan
2cubes sphere.mtx	101492	101492	874378	13.33	-nan
tmt sym.mtx	726713	726713	2903837	-	-
StocF-1465.mtx	1465137	1465137	11235263	-	-

3 Report

3.1 Mathematical Breakdown

An iterative technique for solving systems of linear equations is the Jacobi method. On matrices that are diagonally dominant, it works best as it is able to approximate values for solutions over set amount of iterations and get increasingly more accurate. This approach of addressing the problem involves taking the matrix and dividing it into several equations. Each specific x value for each row and column is isolated and solved in terms of the other negative x values plus the b value.
X11 = X12 X13 + b
X22 = X21 X23 + b
X33 = X31 X32 + b

Each equation represents one row of the matrix and it's solution x, assume all rows of matrix A are populated. By isolating the diagonal entries, this approach finds the proper values. It solves for what they equal, and the result should be a matrix A with all diagonal values equal to 1. The solution will then be added in the X matrix.

In our case this Jacobi solver function also has additional parameters, max iterations and tolerance. These parameters are added as they essentially give us a stopping criteria for when to stop approximating values from iterations based on how many iterations we've done or what the difference in approximations is between new x value guesses, the tolerance of the difference. The while loop is run until either or both of these stopping criterias are met. It leads us to keep better efficiency while still ensuring a very accurate approximation for the type of matrices we are looking for. Initially, we choose X values at random for each of them, enter them into each equation, and then X11, X22, and X33 will all have new values as a result. After adding the new values for X11, X22, and X33 back into the equations and solving for them, we may utilize these new values to keep resolving and close in on a precise approximation as the new X values keep updating. If we do this enough or until stopping criteria is met, we will have a close enough approximation to the actual answer.

3.2 Vtune

Below is my Vtune printed simulation in the terminal for matrix jnlbrng1 as a sampler to test profiling.

```
vtune -collect hotspots ./main jnlbrng1.mtx
vtune: Warning: Microarchitecture performance insights will not be
   available. Make sure the sampling driver is installed and
   enabled on your system.
```

```
vtune: Collection started. To stop the collection, either press
   CTRL-C or enter from another console window: vtune -r /home/
   marcus/MECHTRON2MP3/2MP3/A3Fix/r002hs -command stop.
The matrix name: jnlbrng1.mtx
The dimension of the matrix: 40000 by 40000
Number of non-zeros: 119600
CPU time taken solve Ax=b: 3.175051
Residual Norm: 2.7507838848902152e-12
vtune: Collection stopped.
vtune: Using result path `/home/marcus/MECHTRON2MP3/2MP3/A3Fix/
   r002hs'
vtune: Executing actions 19 % Resolving information for `main'
vtune: Warning: Cannot locate debugging information for file `/home
   /marcus/MECHTRON2MP3/2MP3/A3Fix/main'.
vtune: Executing actions 75 % Generating a report
                                Elapsed Time: 69.111s
    CPU Time: 68.847s
        Effective Time: 68.847s
        Spin Time: 0s
        Overhead Time: 0s
    Total Thread Count: 1
    Paused Time: 0s

Top Hotspots
Function         Module     CPU Time  % of CPU Time(%)
---------------  ---------  --------  ----------------
ReadMMtoCSR      main        65.620s             95.3%
solver           main         2.984s              4.3%
fmax             libm.so.6    0.175s              0.3%
__isoc99_fscanf  libc.so.6    0.058s              0.1%
spmv_csr         main         0.009s              0.0%
Collection and Platform Info
    Application Command Line: ./main "jnlbrng1.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID
       =Ubuntu DISTRIB_RELEASE=22.04 DISTRIB_CODENAME=jammy
       DISTRIB_DESCRIPTION="Ubuntu 22.04.3 LTS"
    Computer Name: LAPTOP-1O0EUKP4
    Result Size: 5.0 MB
    Collection start time: 06:56:37 04/12/2023 UTC
```

```
    Collection stop time: 06:57:47 04/12/2023 UTC
    Collector Type: User-mode sampling and tracing
    CPU
        Name: Unknown
        Frequency: 3.194 GHz
        Logical CPU Count: 16
        Cache Allocation Technology
            Level 2 capability: not detected
            Level 3 capability: not detected

If you want to skip descriptions of detected performance issues in
   the report,
enter: vtune -report summary -report-knob show-issues=false -r <
   my_result_dir>.
Alternatively, you may view the report in the csv format: vtune -
   report
<report_name> -format=csv.
vtune: Executing actions 100 % done
```

It doesn't give a very good output as the MMtoCSR function isn't very efficient when it comes to super large matrices, and my solver is Jacobi which is also much better for smaller matrices. When running any of the smaller matrices, not enough info is generated due to their size, so this is the best sample we can do to test my efficiency.

.
.
.
.
.
.
.
.
.
.

3.3 gcov

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

```
Creating 'main.c.gcov'

File 'functions.c'
Lines executed:95.96% of 99
Creating 'functions.c.gcov'

Lines executed: 95.39% of 130
```

After running my program with gcov my terminal output was able to determine how accurate and efficient my code was. It can be seen that almost the entirety of my code is ran through my program each time it is used. Some reasons for parts not running are the checks for symmetry or invalid memory allocation, along with if statements that help to tell if the files are being read correctly.

3.4 Plots

3.4.1 Graphing code

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

# Read the Matrix Market file
matrix = mmread('jnlbrng1.mtx').toarray()

# Extract the row and column indices of non-zero entries
row_ind, col_ind = (matrix != 0).nonzero()

# Create a figure with a slightly blue background
fig, ax = plt.subplots()
ax.set_facecolor('#e6f7ff')  # Light blue background color

# Plot the non-zero entries as squares
square_size = 1  # Adjust the square size as needed
scatter = ax.scatter(col_ind, row_ind, marker='s', color='blue',
   label='Non-zero entries', s=square_size)

# Customize the plot
```

```
ax.set_title('Matrix Sparsity Plot with Squares', color='navy',
    fontsize=16)
ax.set_xlabel('Column Index', color='navy')
ax.set_ylabel('Row Index', color='navy')
ax.invert_yaxis()  # Invert the y-axis to keep the flip
ax.grid(True, linestyle='--', alpha=0.7)
ax.legend()

# Set the aspect ratio to be equal
ax.set_aspect('equal')

# Set the color of tick labels
ax.tick_params(axis='x', colors='navy')
ax.tick_params(axis='y', colors='navy')

# Save the plot as an image file
downloads_path = os.path.join(os.path.expanduser("~"), "Downloads")

# Check if the directory exists, if not, create it
if not os.path.exists(downloads_path):
    os.makedirs(downloads_path)

save_path = os.path.join(downloads_path, 'matrix_sparsity_plot.png'
    )
plt.savefig(save_path)

#Downloads the plot into directory instead of opening image (unable
    to directly open image on this version)
plt.close()
```

For matrix graphing, I was able to make use of pythons built in libraries that can read a matrix in a mtx file and produce a graph as a png file under a directory in VSCode, which in my case was the downloads directory. The file has information of the location of every row and column from the mtx file along with where non-zero entries are. This information is placed into a new file, converted into a png, and produced/stored in a different directory based on where you want it to go.

3.4.2 Matrices and Graphs

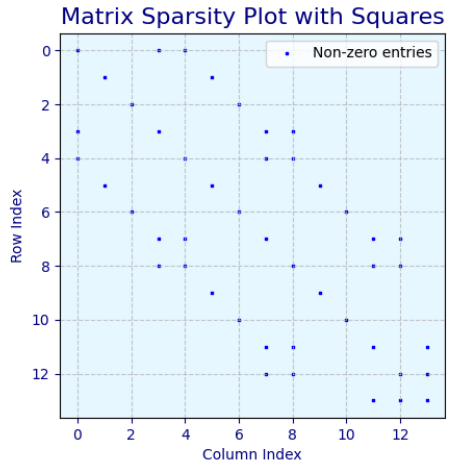

Figure 1: LFAT5

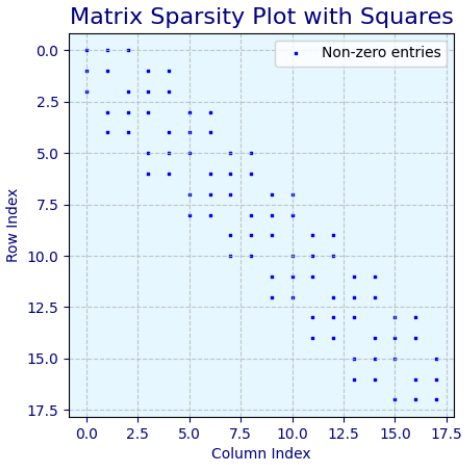

Figure 2: LF10

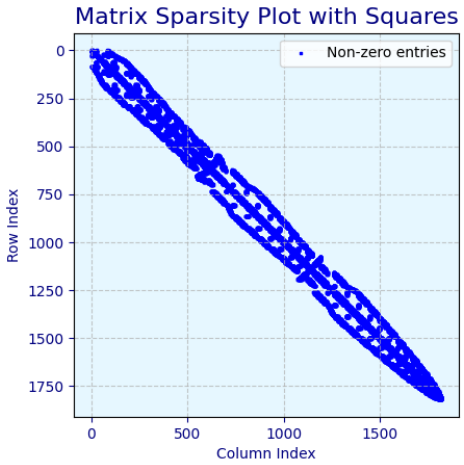

Figure 3: ex3

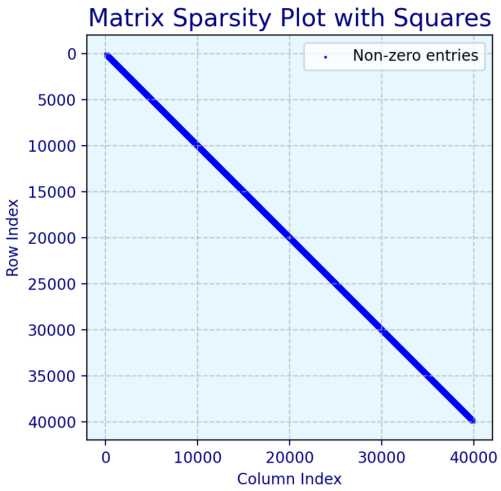

Figure 4: jnlbrng1

```
Traceback (most recent call last):
  File "/home/marcus/MECHTRON2MP3/2MP3/A3Fix/graphs.py", line 6, in
      <module>
    matrix = mmread('jnlbrng1.mtx').toarray()
  File "/home/marcus/.local/lib/python3.10/site-packages/scipy/
     sparse/_coo.py", line 329, in toarray
    B = self._process_toarray_args(order, out)
  File "/home/marcus/.local/lib/python3.10/site-packages/scipy/
     sparse/_base.py", line 1267, in _process_toarray_args
    return np.zeros(self.shape, dtype=self.dtype, order=order)
numpy.core._exceptions._ArrayMemoryError: Unable to allocate 11.9
   GiB for an array with shape (40000, 40000) and data type float64
```

Unfortunately, I run into this error with my compiler for any of the larger matrices as I do not have enough space to generate the graph for them on my current VScode setup.

3.5 Issues With Matrix Solving

As outlined in some of the above sections, the Jacobi solver that I incorporated into my function to estimate the matrices solved values and compare to the actual with residual calculations don't always line up. For some matrices, especially larger ones, the residual is either very large or reported as inf/-nan, meaning it wasn't even a number. Part of this is the nature of the larger matrices to not converge or converge over such a long period of time that we'd need way too many iterations to solve them.

Since Jacobi's method is an iterative method that works better for diagonally dominant and smaller matrices, the nature of having to iterate for long periods of time for super large matrices makes it unrealistic to solve some of the larger matrices with an abnormally long CPU time and highly innaccurate results due to how many actual guesses of x are needed and reiterated before it can become somewhat accurate. This explains how the smaller matrices generally have a really low residual value and fast CPU time to solve, as Jacobi's method is meant to excel with smaller matrices that require less to solve and less guess values until a reasonable solution is attained.

3.6 Makefile

```
CC = gcc
CFLAGS = -Wall -Wextra -std=c99
SOURCES = functions.c main.c
OBJECTS = $(SOURCES:.c=.o)
EXECUTABLE = myprogram
LIBS = -lm

all: $(EXECUTABLE)

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

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

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

Make sure that the makefile is installed in the terminal before executing it; if not, follow the installation instructions. After that, you may run it by typing "make" into the terminal. To restart everything, use "make clean" and then "make" once more. This Makefile contains instructions on how to construct a C program using the make build automation tool.

The compiler (gcc) is identified by CC, the compilation flags (-Wall, -Wextra, and -std=c99), the source code files are listed in SOURCES, the corresponding object files are specified in OBJECTS, the final executable name is EXECUTABLE, and the math library (-lm) is included in LIBS. All of these variables are defined at the outset.

The (EXECUTABLE) target determines the default target, which is all. In turn, the (OBJECTS) target is dependent upon the "(EXECUTABLE)" target. The executable creation rule

states that the compiler to be used is the one designated by the user ((CC)), and that the object files and libraries must be linked with the executable.

A pattern rule for creating object files from C source files is the following rule,.o:.c. It uses the compiler and previously provided flags to compile each source file.

In summary, by defining rules for linking object files, compiling source files, and cleaning up output files, this Makefile automates the compilation of C programs. Based on dependencies and timestamps, the make tool applies these rules to decide what needs to be rebuilt.
...