MECHTRON 2MP3 - Porgramming for Mechatronics

Creating a Matrix Solver in C

Marco Tan, 400433483

1 Introduction

For this assignment, we created C code that reads a Matrix Market file, saves it as some CSRMatrix structure, and uses it to solve for $A * x = b$.

2 Algorithms

2.1 Jacobi method

One of the algorithms employed in my program was the Jacobi method, a method of solving for matrices iteratively. We set the solution matrix x to some initial guess. In my code, we set the values to be zero initially. Then, on each iteration, we calculate the next value of x with the following algorithm:

$$x_i{}^{(k+1)} = \frac{1}{a_{ii}}\big(b_i - \sum_{j \neq i} a_{ij} x_j{}^{(k)}\big), i = 1, 2, ..., n$$

This works for most cases. However, there are cases where the solution never converges. Such an example is when the matrix is singular, or there exists a zero somewhere down the diagonal. In my code, I also added a diagonal checker, which attempts to swap rows around until there are zeros down the diagonal.

Another way this method could fail is if the matrix is not diagonally dominant. In a diagonally dominant system, the following inequality is satisfied down the diagonal:

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$$

The Jacobi method fails when the system is not diagonally dominant and will diverge towards infinity.

2.2 Successive Over-Relaxation

The other algorithm employed in this program was Successive Over-Relaxation (SOR). It is a variation of the Guass-Seidel method (which itself is similar to the Jacobi method, but differs in that it forward-substitutes values to converge quicker towards the answer). How SOR works is by taking a weighted average of our current estimate and the next estimate based on some "relaxation factor" ω. This can cause our solution to converge faster, but larger values of $\omega \geq 1$, it could make the algorithm unstable and end up making it diverge instead.

This is the algorithm, notice the similarities to the Gauss-Seidel method:

$$x_i{}^{(k+1)} = (1-\omega)x_i{}^{(k)} + \frac{\omega}{a_{ii}}\big(b_i - \sum_{j<i} a_{ij}x_j{}^{(k+1)} - \sum_{j>i} a_{ij}x_j{}^{(k)}\big), i = 1, 2, ..., n$$

The benefit of choosing this algorithm over the Jacobi Method is that convergence is generally faster. Like stated before though, the method is potentially unstable for very large relaxation factors.

3 Results

Table 1: Results of Jacobi Method, Max 10,000 iterations, Residual Threshold of 1e-14

Problem	Size		Non-zeros	CPU time (sec)	Norm-Residual
	row	column			
LFAT5.mtx	14	14	30	0.105591	2.659256e-14
LF10.mtx	18	18	50	0.125620	3.783679e-02
ex3.mtx	1821	1821	27253	1.984293	-nan
jnlbrng1.mtx	40000	40000	119600	9.686789	1.253505e-12
ACTIVSg70K	69999	69999	154313	29.686923	-nan
2cubes_sphere.mtx	101492	101492	874378	84.701966	-nan
tmt_sym.mtx	726713	726713	2903837	334.341352	-nan
StocF-1465.mtx	1465137	1465137	11235263	934.029447	-nan

Table 2: Results of Successive Over-Relaxation, Max 10,000 iterations, Omega = 1.5, Residual Threshold of 1e-14

Problem	Size		Non-zeros	CPU time (sec)	Norm-Residual
	row	column			
LFAT5.mtx	14	14	30	0.093380	2.956614e-14
LF10.mtx	18	18	50	0.102590	4.467470e-13
ex3.mtx	1821	1821	27253	1.855408	5.937263e+01
jnlbrng1.mtx	40000	40000	119600	9.516753	2.021652e-12
ACTIVSg70K	69999	69999	154313	32.508544	-nan
2cubes_sphere.mtx	101492	101492	874378	89.707143	8.766432e-14
tmt_sym.mtx	726713	726713	2903837	355.608555	8.538759e+02
StocF-1465.mtx	1465137	1465137	11235263	1015.277404	7.764548e+03

I notice that the performance of the Jacobi Method is significantly worse than Successive Over-Relaxation. My hypothesis as to why this is the case is that the algorithm for SOR is similar to the Gauss-Seidel method, which uses forward substitution to speed up convergence. Further, the relaxation factor would converge our solution faster since it's based on a weighted average between the previous and current solution. Therefore, as we converge, the average between the previous and current solutions should get smaller and smaller, converging faster.

As for why we were not able to converge for certain matrices, it's most likely because all these matrices are not *strictly* diagonally dominant. They might be diagonally dominant in specific rows, making these matrices weakly diagonally dominant, but a strictly diagonally dominant matrix would have guaranteed convergence.

3.1 VTune

Here is the VTune CLI output:

```
vtune: Collection stopped.
vtune: Using result path ‘/home/ionicargon/MECHTRON-2MP3/github/MT2MP3-MatrixSolver/r002hs
vtune: Executing actions 75 % Generating a report                              Elapsed Time
    CPU Time: 34.120s
        Effective Time: 34.120s
        Spin Time: 0s
        Overhead Time: 0s
    Total Thread Count: 1
    Paused Time: 0s

Top Hotspots
Function          Module     CPU Time  % of CPU Time(%)
---------------  ---------  --------  ----------------
solver_iter_SOR   main        17.470s             51.2%
spmv_csr          main        13.800s             40.4%
compute_residual  main         1.739s              5.1%
compute_norm      main         0.640s              1.9%
__GI__IO_fflush   libc.so.6    0.301s              0.9%
[Others]          N/A          0.170s              0.5%
Effective Physical Core Utilization: 14.8% (0.888 out of 6)
 | The metric value is low, which may signal a poor physical CPU cores
 | utilization caused by:
 |     - load imbalance
 |     - threading runtime overhead
 |     - contended synchronization
 |     - thread/process underutilization
 |     - incorrect affinity that utilizes logical cores instead of physical
 |       cores
 | Explore sub-metrics to estimate the efficiency of MPI and OpenMP parallelism
 | or run the Locks and Waits analysis to identify parallel bottlenecks for
 | other parallel runtimes.
 |
    Effective Logical Core Utilization: 7.9% (0.952 out of 12)
     | The metric value is low, which may signal a poor logical CPU cores
     | utilization. Consider improving physical core utilization as the first
     | step and then look at opportunities to utilize logical cores, which in
     | some cases can improve processor throughput and overall performance of
     | multi-threaded applications.
     |
```

```
Collection and Platform Info
    Application Command Line: ./bin/main "./test/jnlbrng1.mtx"
    Operating System: 5.15.133.1-microsoft-standard-WSL2 DISTRIB_ID=Ubuntu DISTRIB_RELEASE=
    Computer Name: DESKTOP-J8KNLS6
    Result Size: 4.7 MB
    Collection start time: 00:03:56 06/12/2023 UTC
    Collection stop time: 00:04:30 06/12/2023 UTC
    Collector Type: Driverless Perf per-process counting,User-mode sampling and tracing
    CPU
        Name: Intel(R) microarchitecture code named Coffeelake
        Frequency: 2.208 GHz
        Logical CPU Count: 12
        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
```

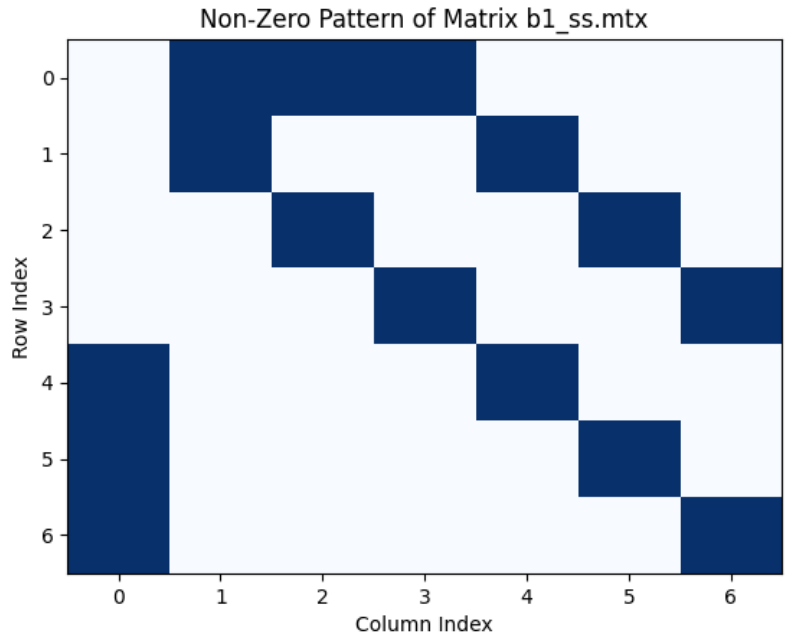
The VTune outputs for all the other files (except for the ones that I couldn't solve) were pretty similar.

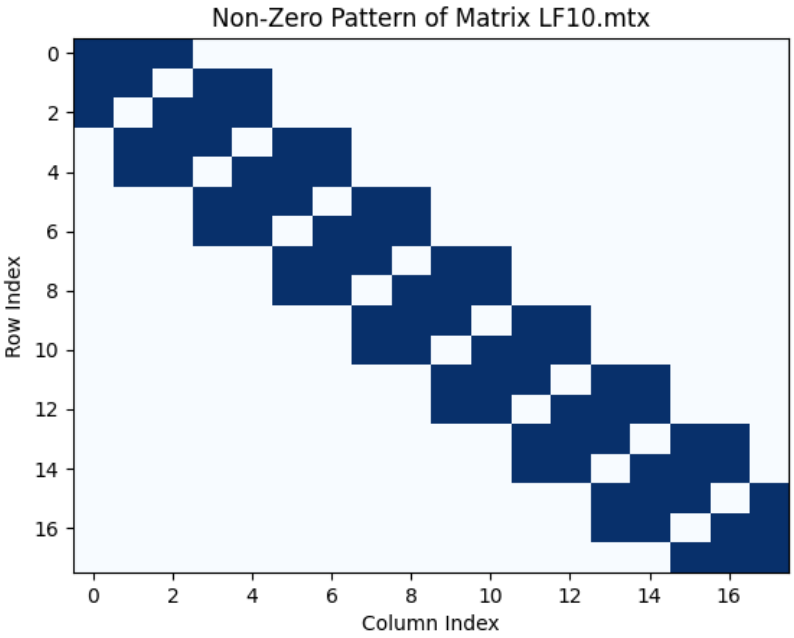
As we can see, the bulk of the hotspots are in the SOR solving function (it's the Jacobi function when I compile it as such) and the function `spmv_csr`, the function that performs matrix multiplications. There really isn't any other way of optimizing these functions short of parallelizing them.

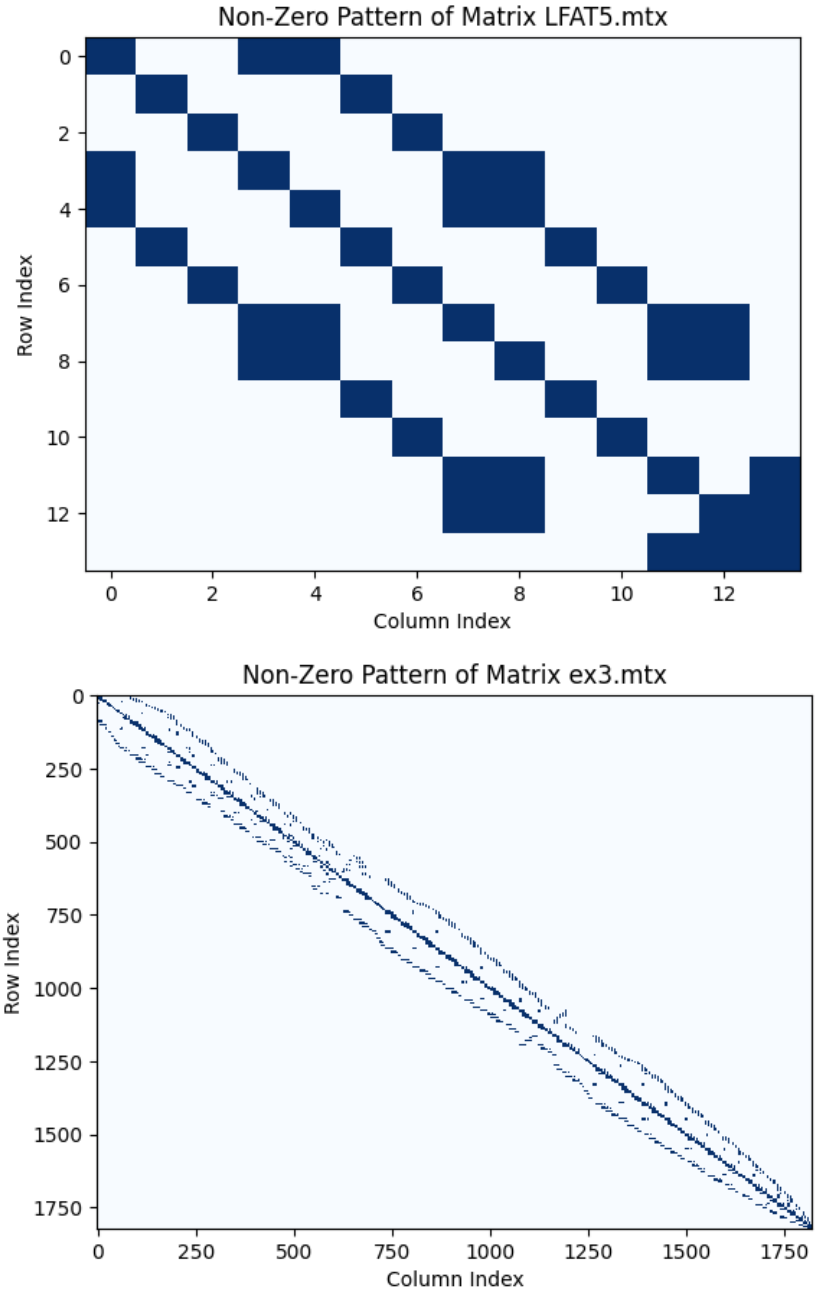
Furthermore, the poor core usage indicator shows that this program could use the benefits of parallelism to run code in concurrent tasks, which would speed up our code significantly.

4 Matrix Visualization

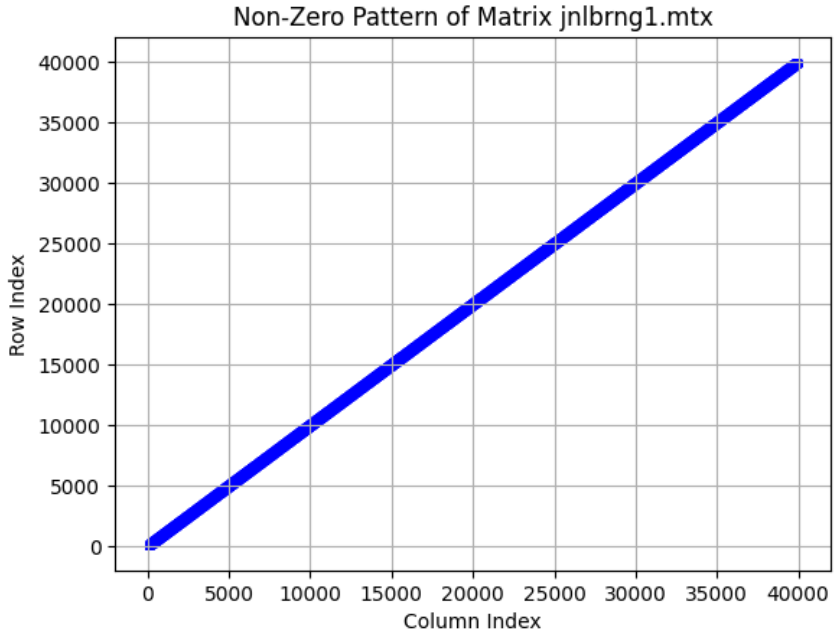
These matrices were visualized using `numpy` and `matplotlib` in Python 3.12. I could not link the Python properly to the C program because WSL does not come installed with an X server, so any GUI program does not work properly in WSL. Further, matrices that were larger than `ex3.mtx` required exponentially more memory than I have, so I couldn't plot them.

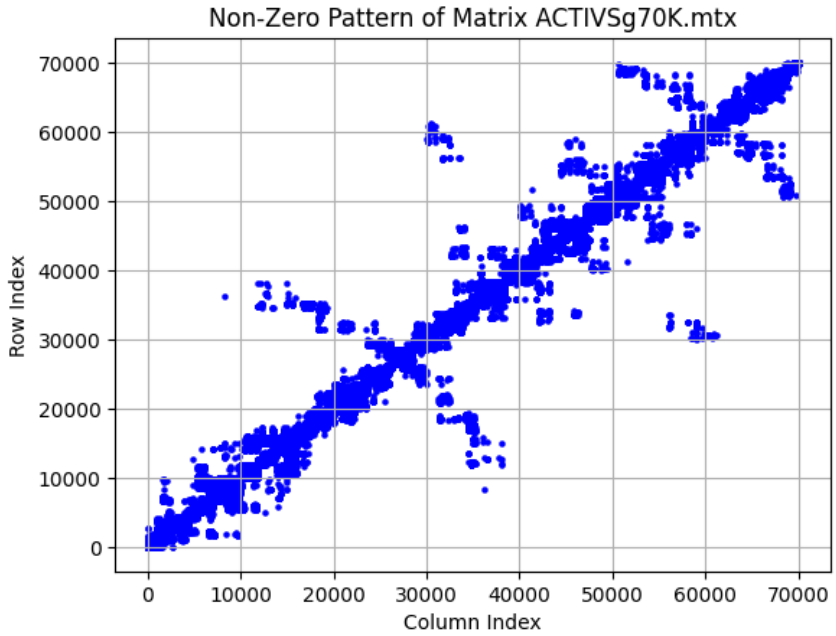


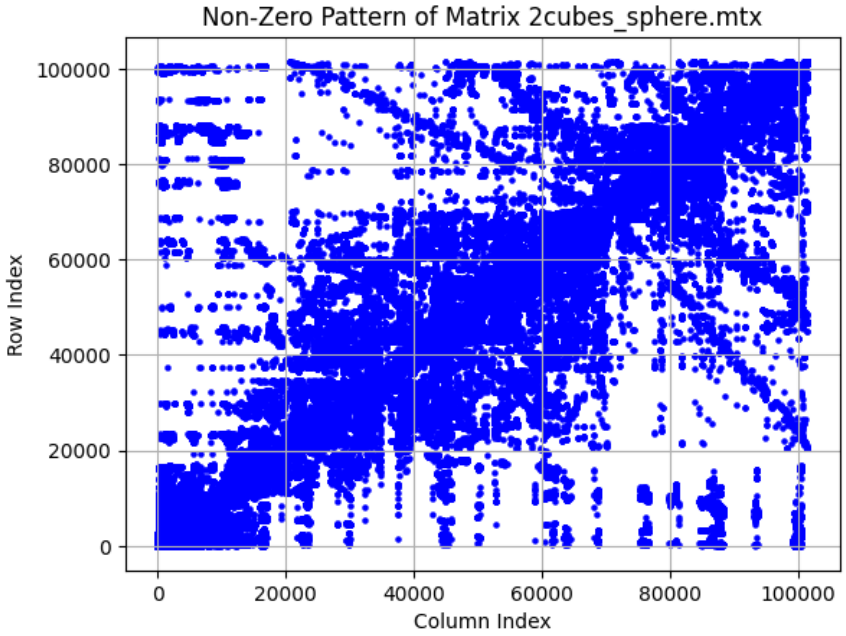


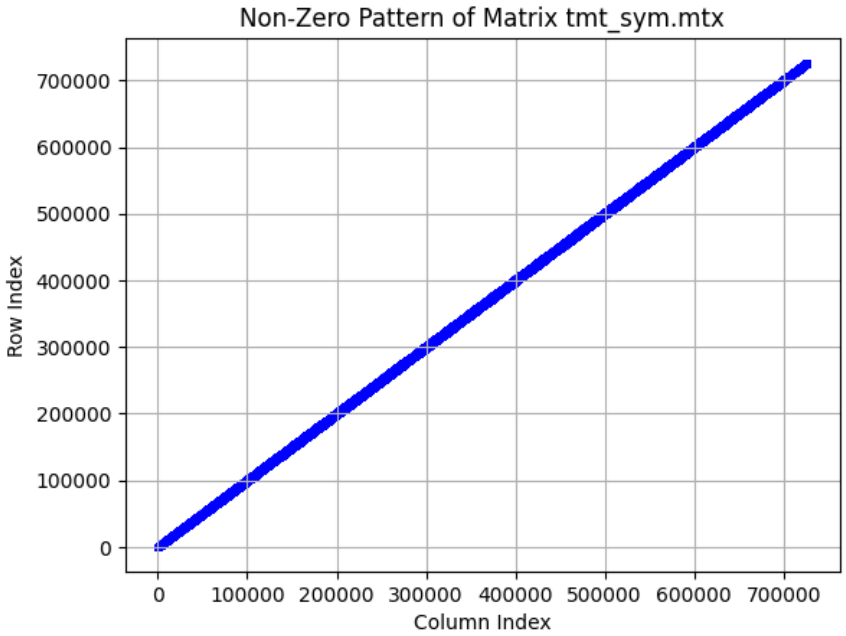


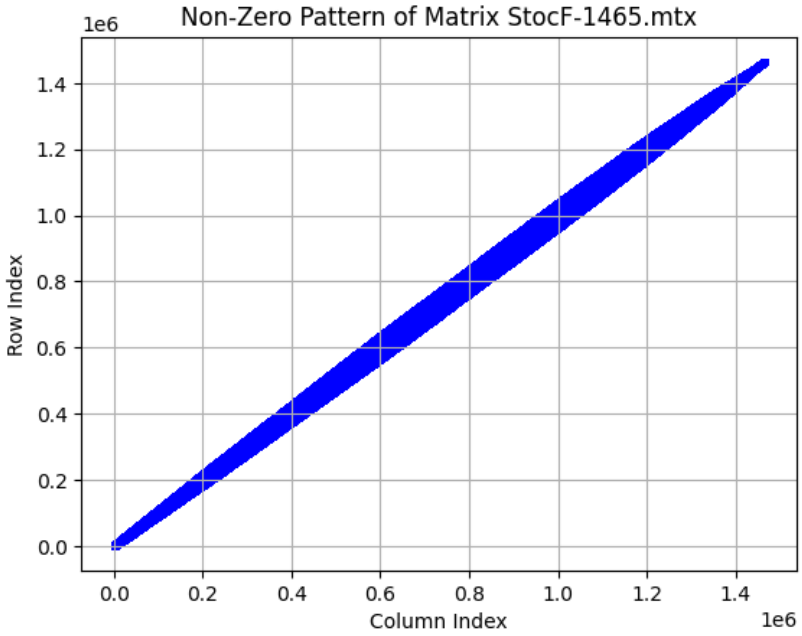
The visualizations of the larger files are achieved with a scatter plot as the binary matrices take up too much memory:


Non-Zero Pattern of Matrix jnlbrng1.mtx
Row Index
Column Index
0
5000
10000
15000
20000
25000
30000
35000
40000


Non-Zero Pattern of Matrix ACTIVSg70K.mtx
Row Index
Column Index
0
10000
20000
30000
40000
50000
60000
70000


Non-Zero Pattern of Matrix 2cubes_sphere.mtx
Row Index
Column Index
0
20000
40000
60000
80000
100000


Non-Zero Pattern of Matrix tmt_sym.mtx
Row Index
Column Index
0
100000
200000
300000
400000
500000
600000
700000


Non-Zero Pattern of Matrix StocF-1465.mtx
1e6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Row Index
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1e6
Column Index

Here is the code used to visualize these matrices:

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

def visualize_matrix_market(file_path):
    matrix = mmread(file_path)

    non_zero_indices = matrix.nonzero()
    rows, cols = non_zero_indices

    plt.scatter(cols, rows, s=5, c='blue', marker='o')
    plt.title(f'Non-Zero Pattern of Matrix {file_path}')
    plt.xlabel('Column Index')
    plt.ylabel('Row Index')
    plt.grid(True)
    plt.show()

matrix_market_file_path = 'your_matrix_market_file.mtx'
visualize_matrix_market(matrix_market_file_path)
```

The code for the larger matrices was similar:

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

def visualize_matrix_market(file_path):
    matrix = mmread(file_path)

    binary_matrix = np.zeros_like(matrix.toarray())
    binary_matrix[matrix.nonzero()] = 1

    plt.imshow(binary_matrix, cmap='Blues', interpolation='none', aspect='auto'
                                            )
    plt.title(f'Non-Zero Pattern of Matrix {file_path}')
    plt.xlabel('Column Index')
    plt.ylabel('Row Index')
    plt.show()

matrix_market_file_path = 'your_matrix_market_file.mtx'
visualize_matrix_market(matrix_market_file_path)
```

5 Makefile

The following code is my Makefile:

```
CC=gcc
CFLAGS=-lm -Ofast
DFLAGS=-Wall -Wextra -W -g -O0 -lm

FLAGSSOR= -DSOR -DMAX_ITER=10000 -DTHRESHOLD=1e-14 -DOMEGA=1.5  -DDIAGONAL_CHECK
FLAGSJacobi= -DJACOBI -DMAX_ITER=10000 -DTHRESHOLD=1e-14 -DDIAGONAL_CHECK
FLAGSPrint= -DPRINT=1

FLAGS= $(DFLAGS) $(FLAGSSOR) # $(FLAGSPrint) #-DUSER_INPUT

SDIR=src
IDIR=include
ODIR=obj
BDIR=bin

_DEPS = functions.h
DEPS = $(patsubst %,$(IDIR)/%,$(_DEPS))

_OBJ = main.o functions.o
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))

all: check_dir $(BDIR)/main

$(ODIR)/%.o: $(SDIR)/%.c $(DEPS)
$(CC) -c -o $@ $< -I$(IDIR) $(FLAGS)

$(BDIR)/main: $(OBJ)
$(CC) -o $@ $^ -I$(IDIR) $(FLAGS)


.PHONY: clean check_dir


clean:
rm -f $(ODIR)/*.o $(BDIR)/main
```

```
if [ -d "./r000hs/" ]; then rm -rf ./r000hs/; fi
if [ -f solution.txt ]; then rm solution.txt; fi
if [ -f smvp_output.txt ]; then rm smvp_output.txt; fi

check_dir:
if [ ! -d "./$(ODIR)/" ]; then mkdir $(ODIR); fi
if [ ! -d "./$(BDIR)/" ]; then mkdir $(BDIR); fi
```

This Makefile is similar to the one I had for assignment 2, but with a few differences:

- I have a set of flags (`FLAGSJacobi`, `FLAGSSOR`) for choosing between different methods. Originally, my code worked for the two methods, the Jacobi Method and Successive Over-Relaxation. We can use these variables in the `FLAGS` variable to change the method we compile with.
  - `MAX_ITER` defines the maximum number of iterations.
  - `THRESHOLD` is how low our norm of the residual needs to reach before we end early.
  - For SOR, `OMEGA` is our relaxation factor $\omega \geq 1$
  - `DIAGONAL_CHECK` will check through the diagonal for zeros and perform row swaps to try to fill in zeros.
- the `check_dir` checks to see if the build and object directories exist before building the program. If they don't exist, they are automatically created.
- `DPRINT` selects the kind of printing we have in the program. 1 prints the raw CSR matrix while 2 pretty prints the matrix, generating an ASCII matrix in the terminal. If this flag isn't defined, then the code outputs to files by default.