Tiffany Wang

260684152

ECSE 543 – Assignment 1

The assignment was fully completed in python.

Question 1

1. Choleski

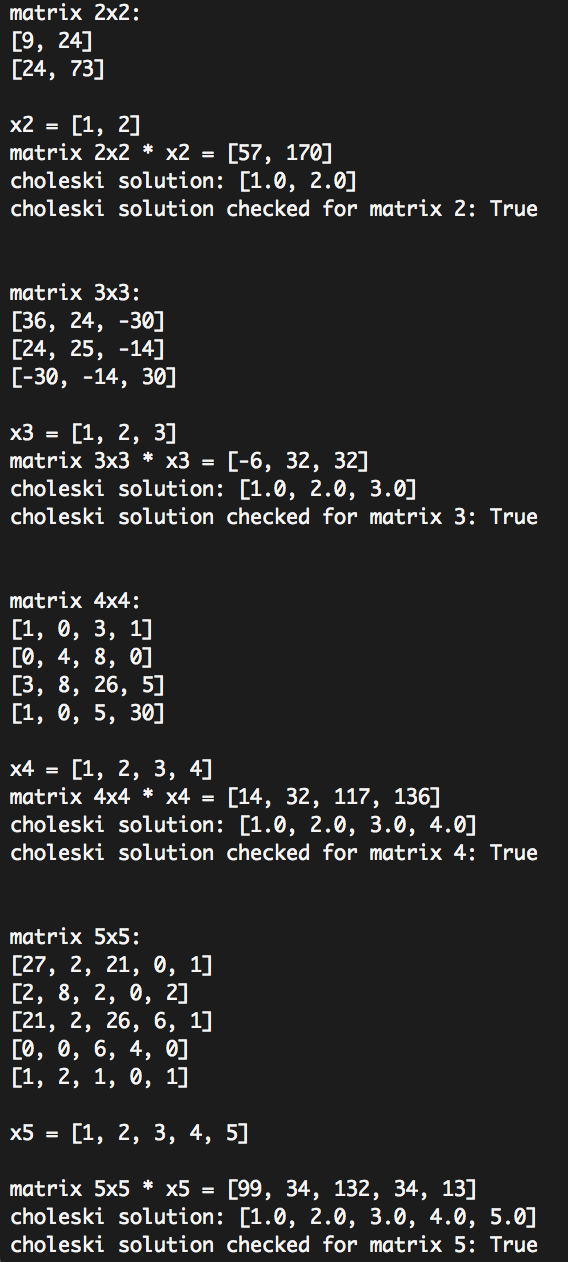
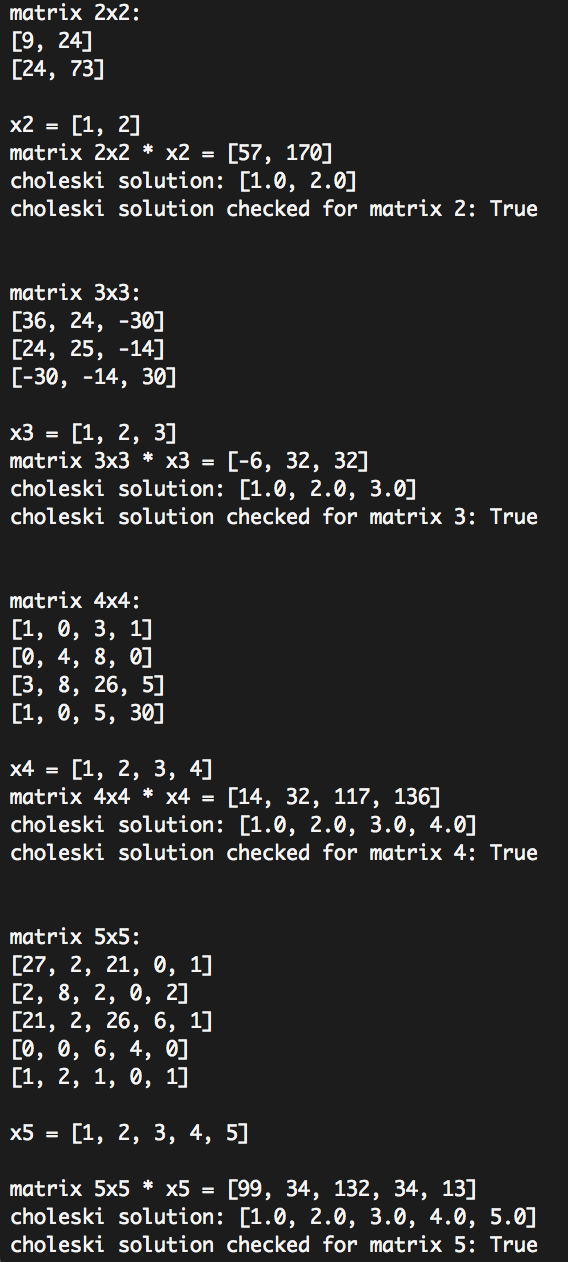
Several helper methods were used to program choleski(). These methods include the following:

* choleskiDecomposition(matrix): this function takes in the real, symmetric, positive-definite matrix *A* and modified it into a lower triangle matrix *L*, where *L 🞄 LT = A*. The “look-ahead modification” method was implemented in order to save runtime.
* solvingLowerMatrix(matrix, vector): taking in a lower matrix *L* and a vector *b*, the function returns the solution to *L🞄 y =b*.
* solvingTransposeLwerMatrix(matrix, vector): similar to solverLowerMatrix(), this function solves the equation *LT🞄 x = b*, where *LT* is an upper triangle matrix.
* Within choleski(): the read, symmetric, positive definite matrix *A* is first decomposed into its lower triangle matrix *L.* Then using solving solvingLowerMatrix(), we find the corresponding vector *y* to *L🞄 y =b*. Finally, the solution *A🞄x = b*, it found by solving *LT🞄 x = y* with solvingTransposeLwerMatrix().

1. The real, symmetric, positive-definite matrices were built through computing *A = L 🞄 LT*, where *L*, is a singular lower matrix*.*  I made sure that *A* satisfied the condition *ZT 🞄 A 🞄 Z ≠ 0*. As the results do not already satisfy the conditions, I tweaked the values and found real, symmetric, positive-definite matrices through trial and error (especially for the larger size matrices).

5

1. I wrote a vector comparator function to compare the choleski() result and the original *x* vector.



1. In this part, I assumed a standard format of the circuit text file of my choice. Below is an example with an explanation of the deployed format:

consider ‘#’ demarks the beginning of a comment.

|  |
| --- |
| 0 0 0 0 0 0 #first line: J – each entry is separated by a space  20 10 10 30 30 30 #second line: R  10 0 0 0 0 0 #third line: E  #empty line  -1 1 1 0 0 0 #incident matrix A  0 -1 0 1 1 0  0 0 -1 -1 0 1 |

The function parseCircuit() reads the lines of the circuit files, parse the data into J, R, E, A tuples, and returns an array containing all these circuit network information. Then, it is easy to retrieve the data by simply calling:

circuitNetwork = parseCircuit(filename)

J = circuitNetwork[0]

R = circuitNetwork[1]

E = circuitNetwork[2]

A = circuitNetwork[3]

I implemented my matrix manipulation codes to solve following equation to find the node voltages *vn.*

where A is the incidence matrix, Y is the admittance matrix, J is the branch current vector, and E is the branch voltage vector.

The results are as follows:

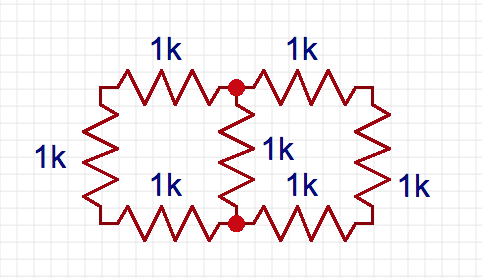
|  |  |  |
| --- | --- | --- |
| Circuit diagram | circuit file | Program Result |
| ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.21.11 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.20.51 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.28.40 |
| ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.29.51 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.32.36 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.28.11 |
| ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.30.07 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.33.44 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.28.07 |
| ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.30.12 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.34.09 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.28.02 |
| ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.30.17 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.34.30 | ../../../../../../Desktop/Screen%20Shot%202017-10-18%20at%208.27.51 |

The results of the choleski circuits analysis are the same as what’s expected (results from hand analysis). Therefore, I conclude that my program works properly.

Question 2

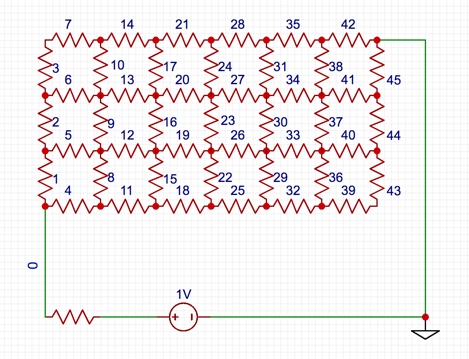
In this problem, I considered a regular *N* x *2N* finite difference mesh circuit to be the following:

e.g. with N = 1



1. In order to calculate the equivalent resistance between the top right and bottom corner, I added an additional mesh connecting the two nodes with a test voltage of 1V and an input resistance of 1kΩ as well.

For a N = 3 circuit, the test circuit would be



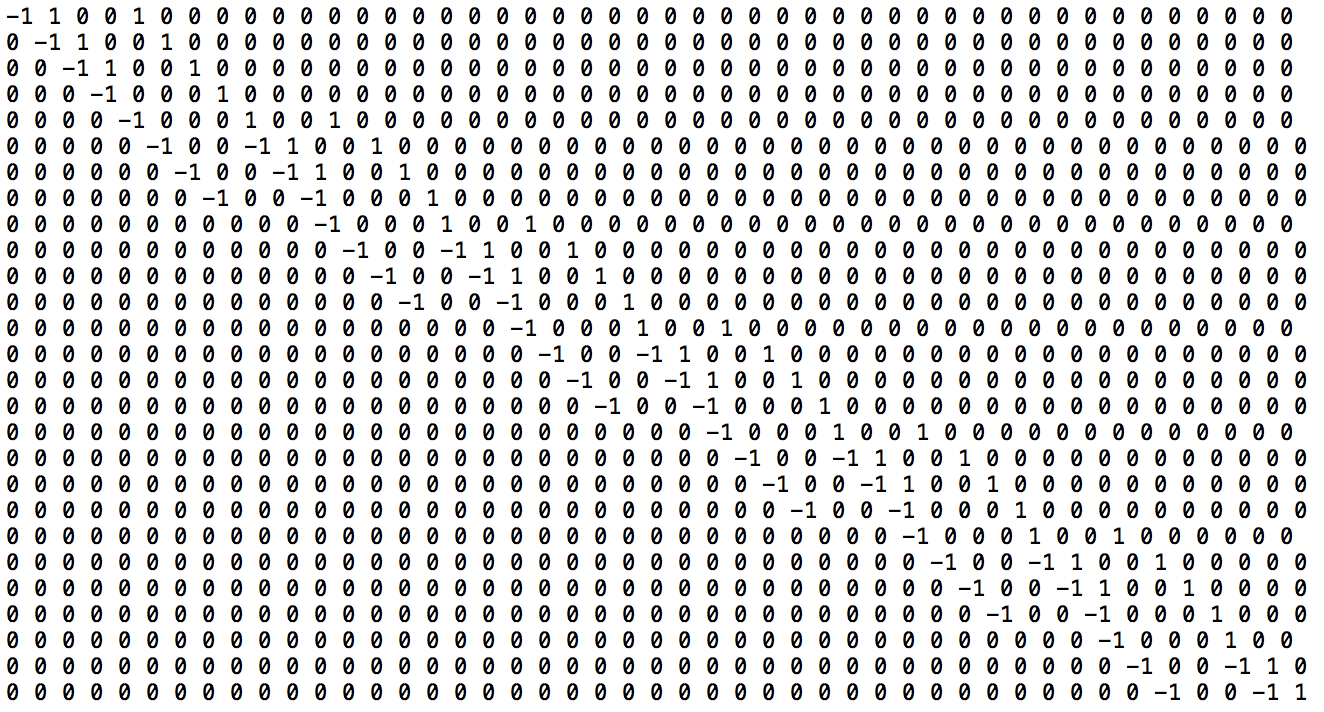
Once we find the node voltage at the bottom left corner of the grid, we can use voltage divider to find *Req* of the grid.

The meshes are numbered in the left to right, bottom to top fashion. The vertical meshes are numbered first, then are the horizontal ones. The nodes are numbers in the same way, the grid would for this example would be:

This will render a total of *(N+1) 🞄 (2N + 1)* nodes and *4N2 + 3N + 1* meshes.

These parameters are taken into account when writing the circuit files. As for the incidence matrix, the way the nodes and meshes are numbers make it easier to determine the current flow with regards to each node. In this analysis, the current flow is always from left to right, bottom to top. Please refer to the code in FDMatrixGenerator() function in circuit.py for more detailed understanding.

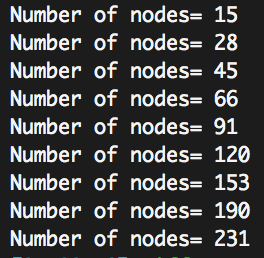
Still using the same circuit example, the generated incidence matrix *A* would be:

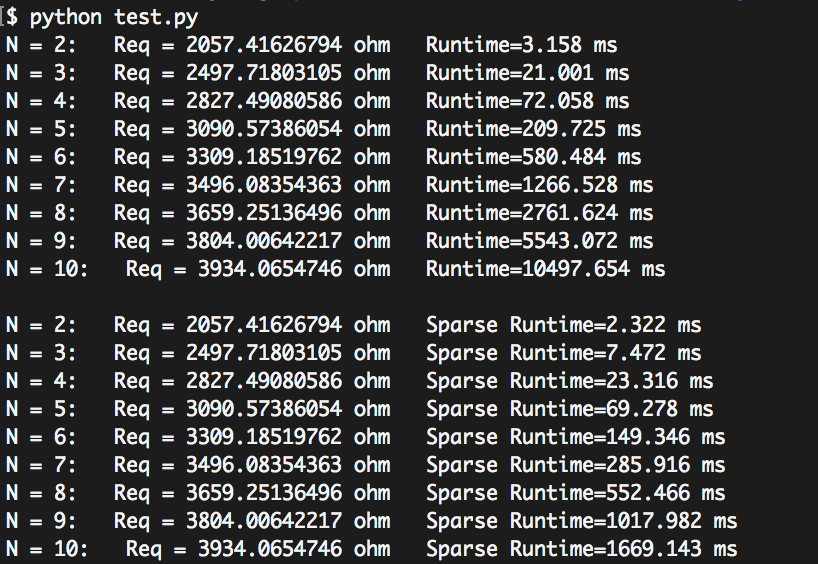


as expected, it is a sparse banded matrix with a maximum of 4 non-zero entries per row.

Please refer to the appendix to all incidence matrices from N = 2 to 10.

1. The *Req* for N = 2 …10 and the runtime required are the following:





In theory, the computing time for this operation is *O(n3)* with *n* being the number of nodes.

Therefore, the expected runtime of the program for each N would be the following:

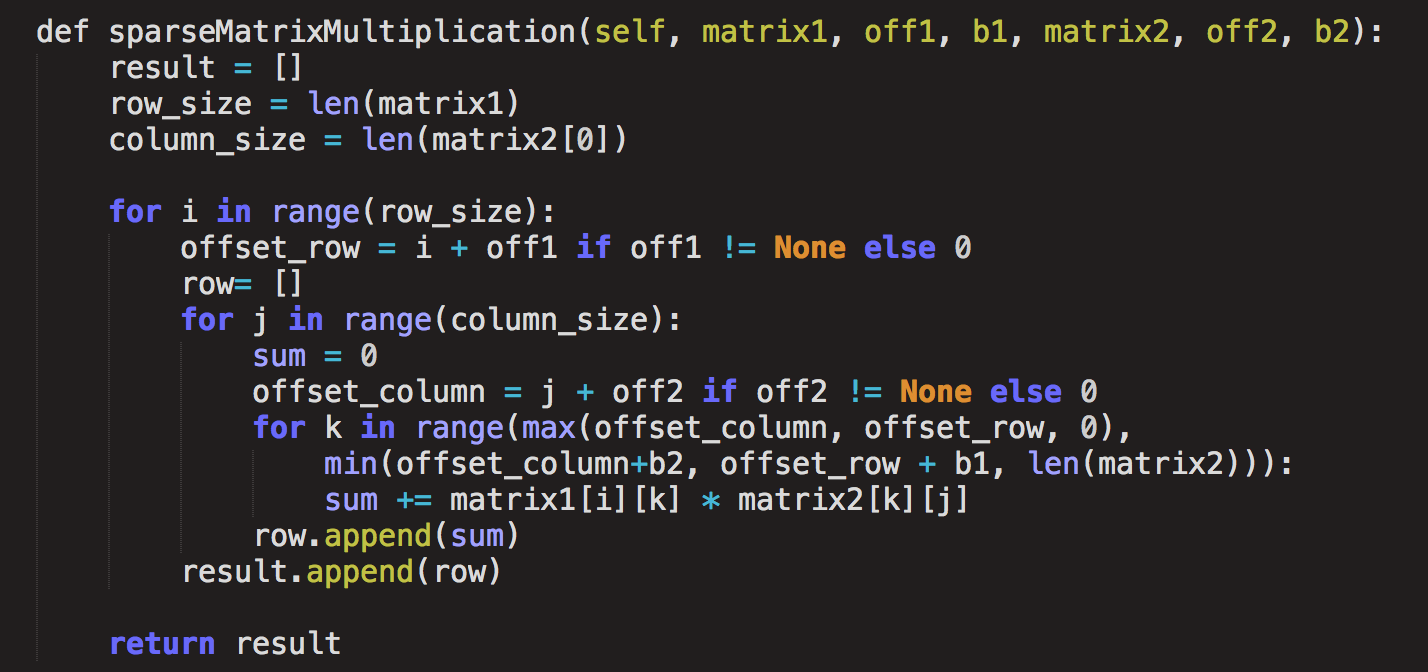
|  |  |  |
| --- | --- | --- |
| N | nodes | *O(n3)* |
| 2 | **15** | 3 375 |
| 3 | **28** | 21 952 |
| 4 | **45** | 91 125 |
| 5 | **66** | 287 496 |
| 6 | **91** | 753 571 |
| 7 | **120** | 1 728 000 |
| 8 | **153** | 3 581577 |
| 9 | **190** | 6 859 000 |
| 10 | **231** | 12 326 391 |

When comparing the outcomes, the runtime of my program is consistent with the theoretical results. They are of the same order, with an error of at most 16% (at N = 10). However, we can note that the practical implementation runtime diverges from the theoretical trend and becomes faster a N grows larger.

1. The incidence matrices *A* and the admittance matrices *Y* are greatly sparse and banded.

The band of the incidence matrix *An*x*m* is: *bandA = m-n,* and that of *Y* is just 1. Therefore, a lot of runtime can be saved in functions including matrix multiplication and choleski decomposition.

* Matrix multiplication



The function only multiplies and adds the components that are within the band of the both matrices. In the case of multiplying *Y🞄AT*, the sparseness comes in handy because the function only computes one multiplication in each row.

* Decomposition of matrix *A* to lower matrix *L*

