## Question 1

#### (a)

The Choleski decomposition was implemented according to the algorithm seen in class. The programming language used for the implementation was Python. A Matrix and MatrixElement classes were created to assist in coding the solution. The Matrix class had helper methods such as multiply (multiplies two matrices), transpose, subtract, clear, get (to get the value of an element), and set (to set the value of an element). Please see appendix for the commented code.

#### (b)

The real, symmetric and positive-definite matrices were constructed using the definition by multiplying a lower matrix L by its transpose to obtain A. This was done for n=2,3 and 4.

#### (c)

The program was tested successfully using the matrices from (b) and arbitrary x vectors. A method called testCholeski was written to take a lower matrix L and a vector x to generate the real, symmetric and positive-definite A () and the vector b (). These were then passed to the Choleski method to compute the solution. The program was made such that a visual comparison between the expected x and the resulting solution x was possible (allowing confirmation of the proper operation of the Choleski method).

#### (d)

The input file organizes data as follows:

Line 1: number of branches

Line 2: number of nodes

Next lines: J,R,E values for each branch

Next lines: incidence matrix (columns separated by commas, and rows by newlines)

The program that reads the file (called circuitSolver.py) works by first reading the number of branches and nodes from the input file. It then goes on to construct the matrices Y, E, J and A (initializing them to zero) based on the number of branches and nodes.

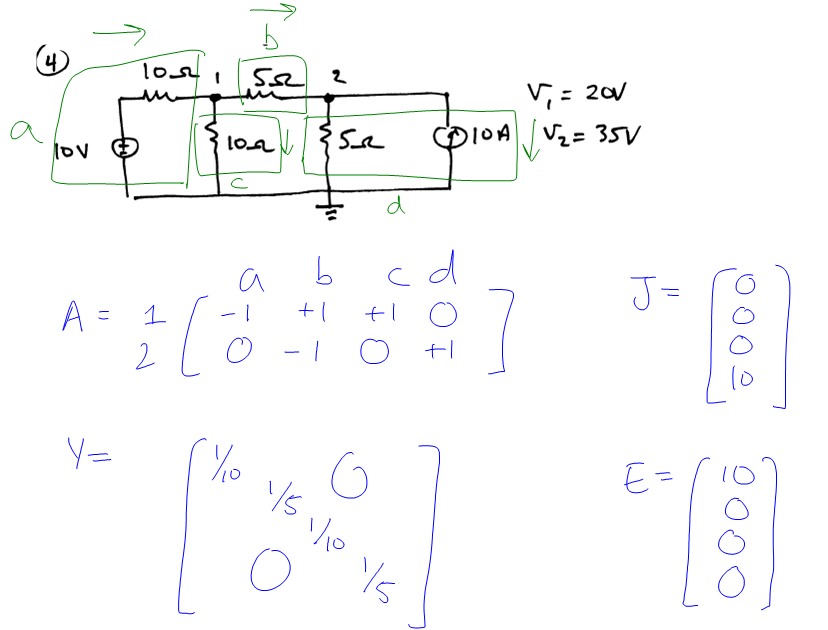
In the next step, the program reads the J,R,E values of each branch and sets the corresponding elements in Y,E and J. Finally, it traverses the incidence matrix row by row (line by line in the file) and similarly sets the corresponding values in the matrix A.

Once Y, E, J and A are properly setup, a call to the Choleski method is made with the real, symmetric and positive-definite matrix defined as:

and the b vector as:

All 5 suggested networks were tested with input files and the voltage results matched the the given solution.

The figure below shows a sample circuit whose branches have been framed and whose A, Y, J and E matrices built. The corresponding input file is show below.



4

2

0,10,10

0,5,0

0,10,0

10,5,0

-1,1,1,0

0,-1,0,1

The remaining circuits are placed in the appendix.

## Question 2

#### (a)

Devising the algorithm to generate the input file turned out to be quite complicated.

First, computing the number of nodes and branches in the grid wasn’t too hard.

Since we are adding a test voltage source around the grid, we will take the bottom right corner to be our ground (hence removing it from the number of nodes), and incrementing the number of branches by 1. This added source branch will also have a test resistance (as R=0 breaks the Choleski algorithm implemented).

The next stage in our input file is to describe each branch line by line. This results in printing 0,1,0 for each of the grid branches and 0,1,1 for the source branch (having a voltage of 1V and test resistance of 1ohm).

The final and most complicated stage was generating the incidence matrix. This required a non-trivial algorithm. It consisted of building a list of branches where each branch is identified by coordinates (node1,node2). The nodes in the grid were numbered level by level (i.e. top vertex is 1, followed by 2 and 3 on the second level, then 4, 5 and 6 on the third level, etc.). The branches were generated based on this numbering scheme and likewise level by level. For each level, we would first save the branches linking the top level of nodes to the lower level of nodes, then follow that by the flat branches linking the nodes on the lower level. For example, in the case of N=2, we would have the following branches saved in a list in this order: (1,2),(1,3),(2,3),(2,4),(2,5),(3,5),(3,6),(4,5),(5,6).

This list of branches then allowed us to build the incidence matrix with dimensions Each column represented a branch from our list of branches, and had all rows set to 0 except the two nodes defining the branch (eg. branch (3,5) would have row 1 set to +1 and row 5 set to -1). Careful consideration was taken in removing the last row (representing the ground node) and adding an extra column for the source voltage.

Once the input generated and the circuit solved, the only voltage of interest would be the one at node 1. This voltage is used in conjunction with the test resistance and the source voltage to determine the of the grid. Voltage division is used as follows to determine the input resistance of the mesh:

|  |  |  |
| --- | --- | --- |
| N |  |  |
| 2 | 0.52632 | 1.1111 |
| 3 | 0.58824 | 1.4286 |
| 4 | 0.62614 | 1.6748 |
| 5 | 0.65228 | 1.8759 |
| 6 | 0.67169 | 2.0459 |
| 7 | 0.68683 | 2.1932 |
| 8 | 0.69907 | 2.323 |
| 9 | 0.70924 | 2.4393 |
| 10 | 0.71786 | 2.5443 |

#### (b)

The Choleski algorithm implemented in Question 1 has three nested loops for the decomposition and elimination steps, and two nested loops for the back substitution step. This results in a total running time of , where n x n is the dimension of the real, symmetric and positive definite matrix A in Ax=b.

In our triangle grid of resistors, there were nodes, resulting in a matrix A of size . This means the Choleski algorithm for our scenario would run in where N is the number of resistors on each side of the grid.

The running time of the Choleski method was calculated using the time.clock() method in python. The starting time was registered before the execution of the algorithm, and likewise the ending time at the end of the algorithm. The elapsed time was then computed as the difference between the two.

The running time was thus collected for N=2 upto 10, and the following graphs show the results along with a comparison with the theoretical expectations.

As can be seen from the plots above, the timings observed for the practical implementation are consistent with the theory.

#### (c)

A method called getHalfBandwidth was written to determine the half-bandwidth of a given matrix. It works by examining every single row in the symmetric matrix and determining the number of elements from the first non-zero value to the diagonal element. The maximum such half-bandwidth is returned as the half-bandwidth of the matrix.

The Choleski algorithm was then optimized to exploit this property in sparse matrices by changing the looping indices in two instances. First, in the decomposition/elimination step, the middle nested loop was made to run from i=j+1 to the half-bandwidth (instead of n). Then, in the back substitution step, the inner loop was likewise changed to run from i=j+1 to the half-bandwidth.

These modifications resulted in significant improvement in the running time. As shown in Figure \_\_\_\_ when compared to previous experimental results, the running time went from values in the range of a tenth of a second to a hundredth of a second.

Theoretically, this improvement would have a running time of as seen in the notes. In our scenario, we recall that n was . Moreover, as seen in the table below, . Therefore, . This indeed represents a significant improvement over our previous algorithm that ran in

|  |  |
| --- | --- |
| N | b |
| 2 | 4 |
| 3 | 5 |
| 4 | 6 |
| 5 | 7 |
| 6 | 8 |
| 7 | 9 |
| 8 | 10 |
| 9 | 11 |
| 10 | 12 |

|  |  |  |
| --- | --- | --- |
| N | Normal Choleski Time (s) | Half Bandwidth Time (s) |
| 2 | 8.33E-04 | 5.56E-04 |
| 3 | 2.86E-03 | 1.02E-03 |
| 4 | 9.60E-03 | 2.15E-03 |
| 5 | 4.74E-02 | 2.47E-03 |
| 6 | 5.66E-02 | 4.30E-03 |
| 7 | 8.22E-02 | 5.04E-03 |
| 8 | 1.56E-01 | 8.13E-03 |
| 9 | 4.00E-01 | 8.61E-03 |
| 10 | 5.67E-01 | 1.66E-02 |

#### (d)

## Question 3

# Appendix

### Question 1 (d)

