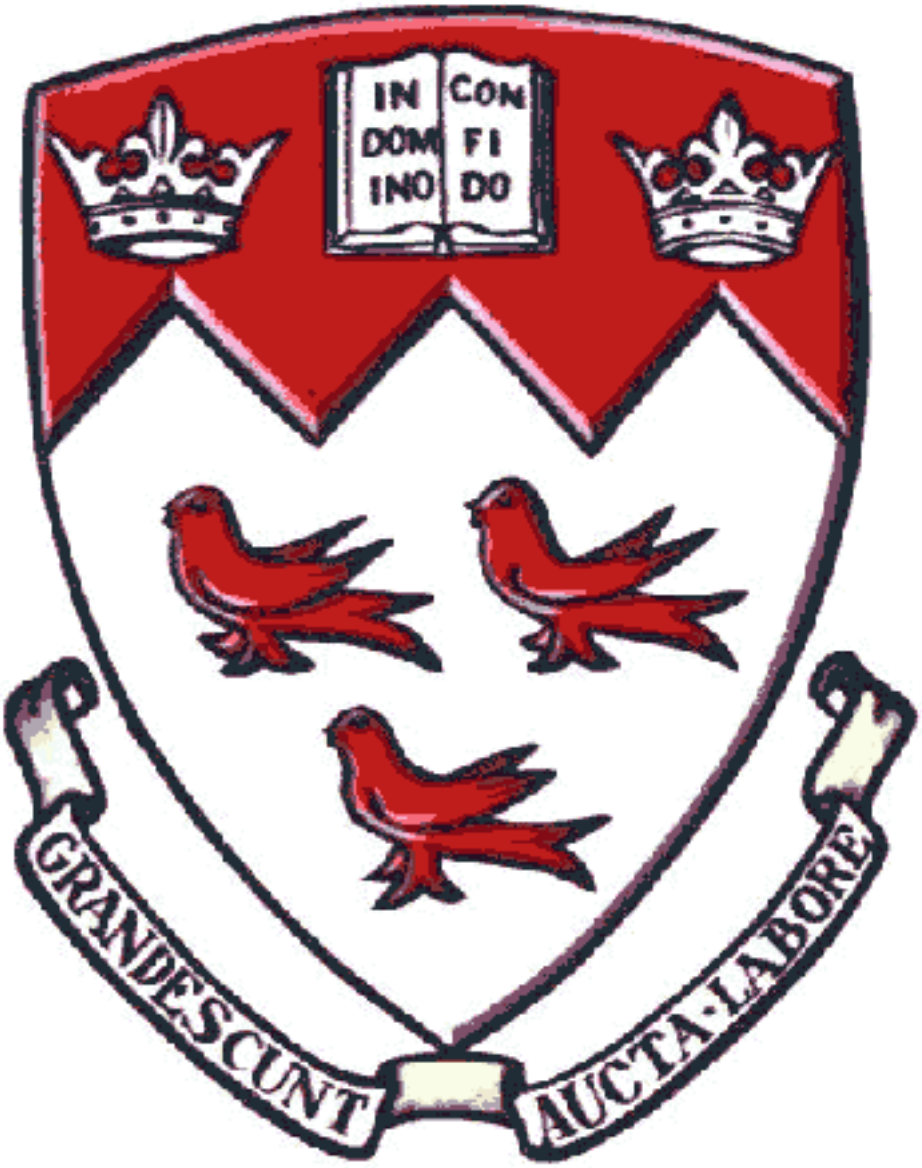
Sharhad Bashar
260519664
ECSE 543



Assignment 1
ECSE 543
Sharhad Bashar
260519664
Oct 17th, 2016

Sharhad Bashar
260519664
ECSE 543

Question 1

a) Write a program to solve the matrix equation Ax=b by Choleski decomposition. A is a real, symmetric, positive-definite matrix of order n.

Code included under appendix.
Solving for x is done in three steps.
Step 1: $A = LL^T$
Step 2: $Ly = b$
Step 3: $L^Tx = y$

Step 1 is the calculation of a lower triangular matrix L from the SPD matrix A of size n. This is done in a function called cholesky in the file basicDefinitions.py. The following equation is used to calculate the values at each indices of L:

For $j = 1,\ldots,n$:

$$L_{jj} = +\sqrt{A_{jj} - (L_{j1}^{\,2} + \cdots + L_{jj-1}^{\,2})}$$

For $i = j+1,\ldots,n$:

$$L_{ij} = \frac{A_{ij} - (L_{i1}L_{j1} + \cdots + L_{ij-1}L_{jj-1})}{L_{jj}}$$

This equation gives us with L matrix of size n, and has a computational cost of $O(n^3)$

Step 2, we solve for y in the equation Ly = b. This step is known as Forward Elimination, and it's done in the function forwElim in the file basicDefinitions.py. The equation used for calculating y is given by:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} L_{ij}y_j}{L_{ii}}$$

This equation returns y, which is a vector of size n. The computational cost of this step is $O(n^2)$

Sharhad Bashar
260519664
ECSE 543

Step 3, we solve for the output x. This step is known as Back Substitution, and it is done in the backSub function of basicDefinition.py. The equation is similar to the one used for Forward Elimination. It is given as follows:

$$x_i = \frac{y_i - \sum_{j=i+1}^{n} L_{ji} x_j}{L_{ii}}$$

This equation returns the answer to the equation Ax = b. x is a vector if size n, and costs $O(n^2)$ to compute.

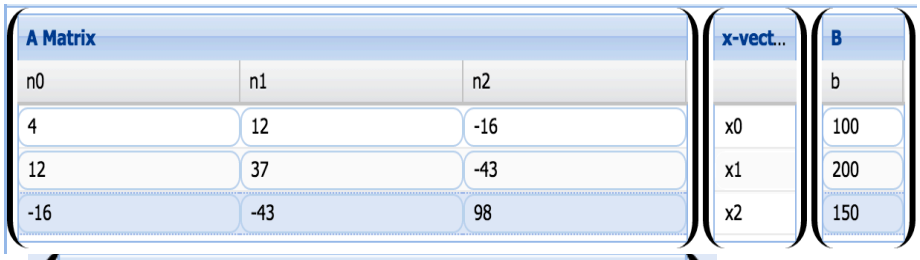
To test the code, the following A and b were inputted to the function:

$$A = \begin{matrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{matrix} \qquad b = \begin{matrix} 100 \\ 200 \\ 150 \end{matrix}$$

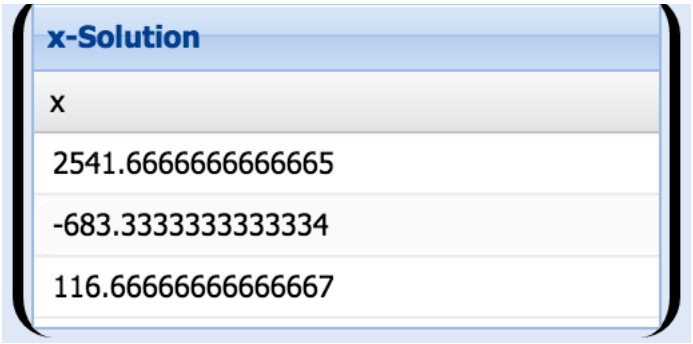
After running the code, the following result was outputted:

```
A:
[4, 12, -16]
[12, 37, -43]
[-16, -43, 98]

b:
[100, 200, 150]

x:
[2541.6666666666665,
-683.3333333333334,
116.66666666666667]
```

This result was compared to an online Ax = b solver from the website:
http://www.mathstools.com/section/main/system_equations_solver
And the results from there are as follows, showing a perfect match, and confirming that the decomposition is functioning:



A Matrix			x-vect...	B
n0	n1	n2		b
4	12	-16	x0	100
12	37	-43	x1	200
-16	-43	98	x2	150



x-Solution
x
2541.6666666666665
-683.3333333333334
116.66666666666667

Sharhad Bashar
260519664
ECSE 543

b) Construct some small matrices (n = 2, 3, 4, ...,10)to test the program. Remember that the matrices must be real, symmetric and positive-definite. Explain how you chose/created the matrices.

From step 1 of part **a)** we can see that $A = LL^T$. This is the equation used to create the symmetric positive definite matrix of order n.
The code for this can be found in the file basicDefinition.py, in the function randomSPD. The input for the function is the matrix size n. This n is taken and used to create a lower triangular matrix L and fill it up with random values. Then the generated L is multiplied with its transpose, to give us the SPD matrix A. Since this function is used solely for testing, using numpy's multiplication and transpose functions created the matrix.

c) Test the program you wrote in (a) with each small matrix you built in (b) in the following way: invent an x, multiply it by A to get b, then give A and b to your program and check that it returns x correctly.

For the testing, we used values of n = 4,5,6,7,8.
These are the following results:
n = 4

```
A:
[ 0.37199395  0.56320916  0.2261093   0.49150244]
[ 0.56320916  1.01033845  0.59282304  0.82241742]
[ 0.2261093   0.59282304  1.02454789  0.66369501]
[ 0.49150244  0.82241742  0.66369501  0.95659082]

b:
[10.0, 10.0, 10.0, 10.0]

x:
[184.76321921374739, -84.459155560159928,
46.391670435204361, -44.053123240301225]
```

n = 5

```
A:
[ 2.32587775  1.57614016  1.59494539  1.8294745   1.26072041]
[ 1.57614016  1.39273117  0.97963438  1.02097841  1.21324239]
[ 1.59494539  0.97963438  1.61977592  1.60239916  1.05194933]
[ 1.8294745   1.02097841  1.60239916  1.84799461  0.93889781]
[ 1.26072041  1.21324239  1.05194933  0.93889781  1.26508895]

b:
[10.0, 10.0, 10.0, 10.0, 10.0]

x:
[-302209.27861930151, 471210.66139459162, 168395.24312987324,
65103.823959742302, -339067.41677429201]
```

Sharhad Bashar
260519664
ECSE 543

n = 6

```
A:
[ 1.72307019  1.30588092  1.56667354  1.28199577  1.16585139  0.60550018]
[ 1.30588092  1.9261678   1.75581552  1.53558071  0.99383296  0.98232881]
[ 1.56667354  1.75581552  2.17116482  1.5702904   1.47960183  1.07978497]
[ 1.28199577  1.53558071  1.5702904   1.67839672  1.35894208  0.81188153]
[ 1.16585139  0.99383296  1.47960183  1.35894208  1.56199538  0.71582027]
[ 0.60550018  0.98232881  1.07978497  0.81188153  0.71582027  1.03788915]

b:
[10.0, 10.0, 10.0, 10.0, 10.0, 10.0]

x:
[7.5776252516901277, 18.817643903270334, -22.647880835167463,
-18.006500112141122, 21.072056943171447, 10.518285000517956]
```

n = 7

```
A:
[ 3.32538131  1.93796651  2.55723586  2.04187577  2.70802269  2.38018019
  1.05524492]
[ 1.93796651  2.28448221  2.47548087  1.36822963  2.30909169  1.36969699
  0.660918  ]
[ 2.55723586  2.47548087  3.61389142  2.11791388  3.12996629  1.99056362
  1.17524199]
[ 2.04187577  1.36822963  2.11791388  2.24397721  1.84006121  1.36418036
  1.11404497]
[ 2.70802269  2.30909169  3.12996629  1.84006121  3.03583929  1.96561215
  0.88095771]
[ 2.38018019  1.36969699  1.99056362  1.36418036  1.96561215  1.93179314
  0.85291402]
[ 1.05524492  0.660918    1.17524199  1.11404497  0.88095771  0.85291402
  0.74243331]

b:
[10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0]

x:
[-116.47035693168102, 22.543853396640181, -190.70824432466907,
-17.181590181816034, 220.62056708316385, 35.702755578438847,
183.80875668013837]
```

n = 8

```
A:
[ 3.45959952  3.36652355  2.29887611  3.59873786  2.28834277  1.78881191
  2.50187642  1.98160697]
[ 3.36652355  3.60988412  2.71410254  3.92547249  2.7664057   2.25291588
  2.62758372  2.13309209]
[ 2.29887611  2.71410254  3.02745155  3.10352418  3.01347861  2.34095117
  2.70690269  2.09120639]
[ 3.59873786  3.92547249  3.10352418  4.57433156  3.29946507  2.6450499
  2.76945107  2.43834186]
[ 2.28834277  2.7664057   3.01347861  3.29946507  3.55646476  2.46242607
  2.52743149  2.31281193]
[ 1.78881191  2.25291588  2.34095117  2.6450499   2.46242607  2.14798622
  1.99813568  1.69501351]
[ 2.50187642  2.62758372  2.70690269  2.76945107  2.52743149  1.99813568
  2.8377717   1.93479428]
[ 1.98160697  2.13309209  2.09120639  2.43834186  2.31281193  1.69501351
  1.93479428  2.13697324]

b:
[10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0, 10.0]

x:
[48.427904940954974, -17.739894162940875, 28.969146794741739,
-33.374167648019586, -1.9920747980218445, 28.668564434315076,
-35.191100346930845, -1.5095422002744121]
```

These values were cross-checked and confirmed with the online Ax = b matrix multiplier. Bigger values of n were also checked to confirm perfect operation.

d) Write a program that reads from a file a list of network branches (J_k, R_k, E_k) and a reduced incidence matrix, and finds the voltages at the nodes of the network. Use the code from part (a) to solve the matrix problem. Explain how the data is organized and read from the file. Test the program with a few small networks that you can check by hand. Compare the results for your test circuits with the analytical results you obtained by hand. Cleary specify each of the test circuits used with a labeled schematic diagram.

Five test circuits were provided for testing. Incident matrices as well as the J_k, R_k and E_k vectors were created, and inputed in a .csv file called testCircuit_1D.csv. the function readCell in basicDefinition.py was created to read the values of the .csv file. The incident matrix and the three vectors were fed into this equation to solve for voltages at the nodes:

$$(AYA^T)V_n = A(J - YE)$$

Y is a diagonal matrix where the diagonal values are the inversed values of the values in the R vector. For this equation, matrix multiplication, addition, subtraction and transpose were created from scratch. These functions are named: matrixMult, matrixAddorSub, matTranspose respectively, and can be found in the file basicDefinition.py

Sharhad Bashar
260519664
ECSE 543

AYA^T gave us an SPD matrix of size n, where as A(J-YE) gave us a vector of size n. These are the A and b of our Ax = b function.
These values are fed into a function called voltageSolver in basicDefnition.py, which returns the voltages at the nodes
The voltage at the nodes of each of the circuits were provided, and we only had to match our answers to the values given to us. The results are shown below:

```
Voltage for circuit 1 is: [5.0]V
Voltage for circuit 2 is: [50.00000000000001]V
Voltage for circuit 3 is: [55.00000000000001]V
Voltages for circuit 4 are: [19.999999999999993, 34.999999999999986]V
Voltages for circuit 5 are: [5.000000000000001, 3.7500000000000004, 3.750000000000001]V
```

All the node voltages matched the test circuit values.

Below is the .cs files with all the incident matrix, E,J and R vectors

Circuit #	A						E	J	R
1	-1	1					10	0	10
							0	0	10
2	1	1					0	10	10
							0	0	10
3	-1	1					10	0	10
							0	10	10
4	-1	1	1	0			10	0	10
	0	0	-1	1			0	0	10
							0	0	5
							0	10	5
5	-1	1	0	0	1	0	10	0	20
	0	-1	1	1	0	0	0	0	10
	0	0	0	-1	-1	1	0	0	30
							0	0	30
							0	0	10
							0	0	30

Sharhad Bashar
260519664
ECSE 543

Question 2

Take a regular N by N finite-difference mesh and replace each horizontal and vertical line by a 1 kOhm resistor. This forms a linear, resistive network.

a) Using the program you developed in question 1, find the resistance, R, between the node at the bottom left corner of the mesh and the node at the top right corner of the mesh, for N= 2, 3, ..., 15.(You will probably want to write a small program that generates the input file needed by the network analysis program. Constructing the incidence matrix by hand for a 225-node network can be tedious.)

The file generateMesh.py has three functions:

- genMesh
- EVector
- JVector
- RVector

The inputs for each of these functions are n, which is the number of meshes in one line of the circuit.

genMesh created the incident matrix in the following steps:

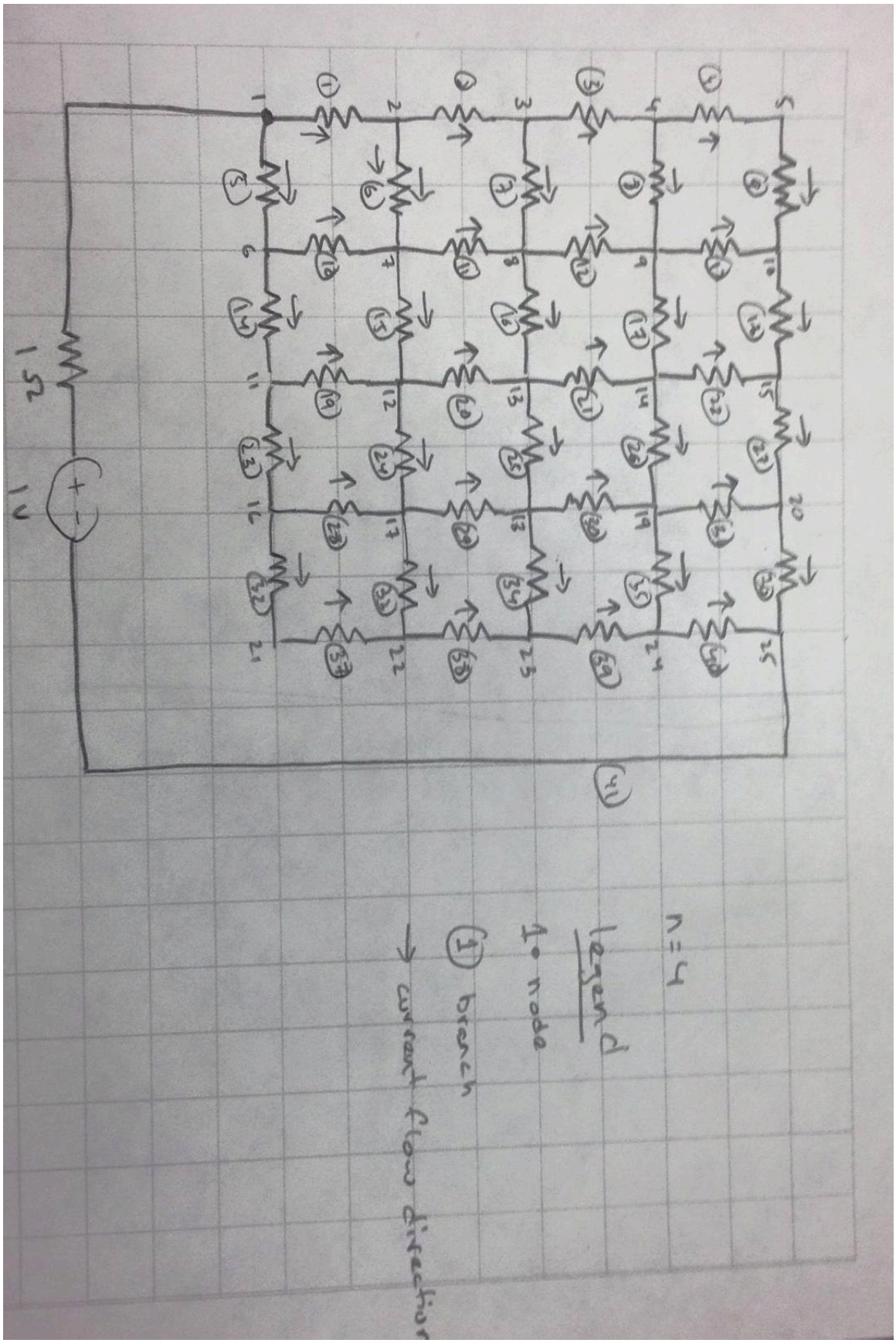
1) Calculate the total number of nodes and branches in the circuits for a given n
2) Create a matrix of size (total nodes) X (total branches + 1) and fill it up with zeros.
3) Then five variables were created that were used to number the nodes and branches connected to the nodes
4) Then the matrix was populated with -1 or 1 in the appropriate indicies, using the following method:

   If current is leaving a node through a certain connected branch then Matrix[node][branch] = 1

   And if current is entering a node through a certain branch connected to it, then Matrix[node][branch] = -1
5) Finally, a voltage source of 1V was connected from the bottom left node to the top right node throught a 1 ohm resistor, and Matrix[0][totalbranch + 1] = -1 and Matrix[totalnode][totalbranch + 1] = 1 was set.

EVector returned a vector of the voltage sources present in the circuit, and since there was only one, it was a vector of zeros except for the very last value, which is 1.
JVector returned a vector of zeros, since there was no current sources
RVector was a vector of resistances, which in this case was a vector of 1000, except for the last one which is 1, to take into account the 1 ohm resistance that connects the source to the mesh.
The incident matrix and 3 vectors are fed into the voltageSolver function, and it solves for the voltage across the mesh. Then we use a simple voltage divider in question_2.py to solve for the equivalent resistance of the mesh.
The following picture (on the picture) shows how the nodes and branches were numbered, as well as the flow of current

Sharhad Bashar
260519664
ECSE 543



Sharhad Bashar
260519664
ECSE 543

The equivalent resistance for values of N = 2,…,20 are shown below:

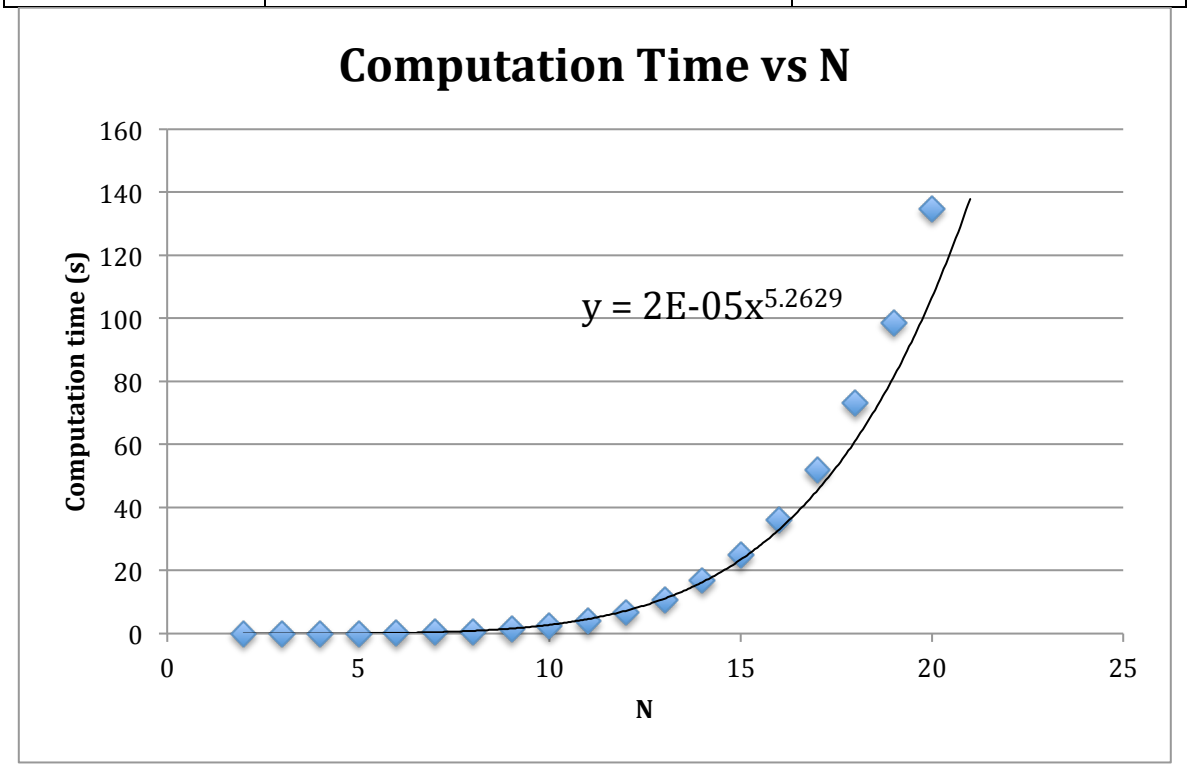
```
Equivaent Resistance for size 2 is: 1499.999999999913 ohms
Equivaent Resistance for size 3 is: 1857.1428571420852 ohms
Equivaent Resistance for size 4 is: 2136.363636363267 ohms
Equivaent Resistance for size 5 is: 2365.6565656557655 ohms
Equivaent Resistance for size 6 is: 2560.144346431078 ohms
Equivaent Resistance for size 7 is: 2728.976763169556 ohms
Equivaent Resistance for size 8 is: 2878.117373770775 ohms
Equivaent Resistance for size 9 is: 3011.6695648946325 ohms
Equivaent Resistance for size 10 is: 3132.576980553496 ohms
Equivaent Resistance for size 11 is: 3243.02258446316 ohms
Equivaent Resistance for size 12 is: 3344.6697258159547 ohms
Equivaent Resistance for size 13 is: 3438.814771661308 ohms
Equivaent Resistance for size 14 is: 3526.4875659696186 ohms
Equivaent Resistance for size 15 is: 3608.5197387283806 ohms
Equivaent Resistance for size 16 is: 3685.5924634299345 ohms
Equivaent Resistance for size 17 is: 3758.27065793761 ohms
Equivaent Resistance for size 18 is: 3827.0279961924243 ohms
Equivaent Resistance for size 19 is: 3892.265540901035 ohms
Equivaent Resistance for size 20 is: 3954.3258538108707 ohms
```

b) In theory, how does the computer time taken to solve this problem increase with N, for large N. Are the timings you observe for your practical implementation consistent with this? Explain your observations.

Theoretically, the computational time should match the time complexity of the cholesky decomposition: $O(n^3)$. For the mesh, n = total number of nodes, which is N^2. So the overall time complexity should be $O(N^6)$. In order to test this, the clock is started just before the voltageSolver function is called, and stopped right after we get the voltage across the mesh. The table below shows the time taken for the values of N:

N	Equivalent Resistance (ohms)	Calculation time (s)
2	1499.9999	0.001046000000314961
3	1857.1428	0.00531399999999848
4	2136.3636	0.01922299999978349
5	2365.6566	0.0606130000001030
6	2560.1443	0.149779999999736
7	2728.9768	0.3355590000001029
8	2878.1174	0.698327999999946
9	3011.6696	1.321473999999852
10	3132.5769	2.37526099999968
11	3243.0226	4.07159700000011
12	3344.6697	6.74055599999974
13	3438.8248	10.66287800000009
14	3526.4876	16.77557900000010
15	3608.5297	25.030382999999
16	3685.5925	36.00026100000014

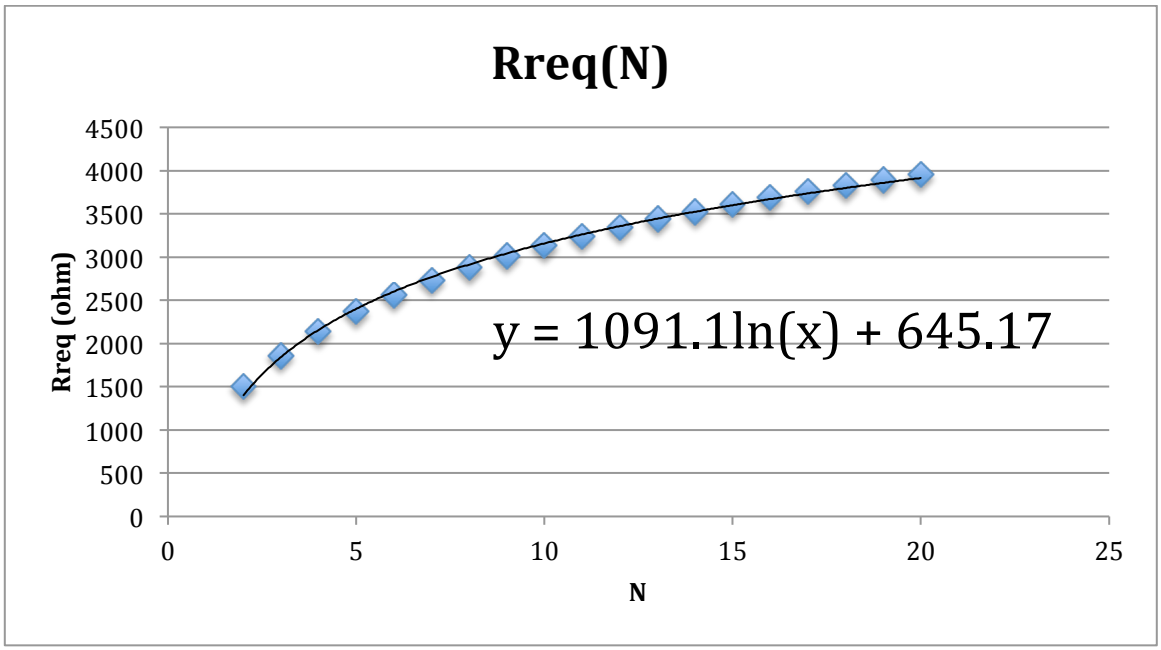
17	3758.2707	51.929596000000
18	3827.2655	73.1550969999998
19	3892.2655	98.7310130000000
20	3954.3259	134.9233400000002



The observed data has a magnitude of power 5, which is close to the theory.

c) The function that represents the curve is a log function, because as N increases, the equivalent resistance starts to plateau, indicating a log function.

Sharhad Bashar
260519664
ECSE 543



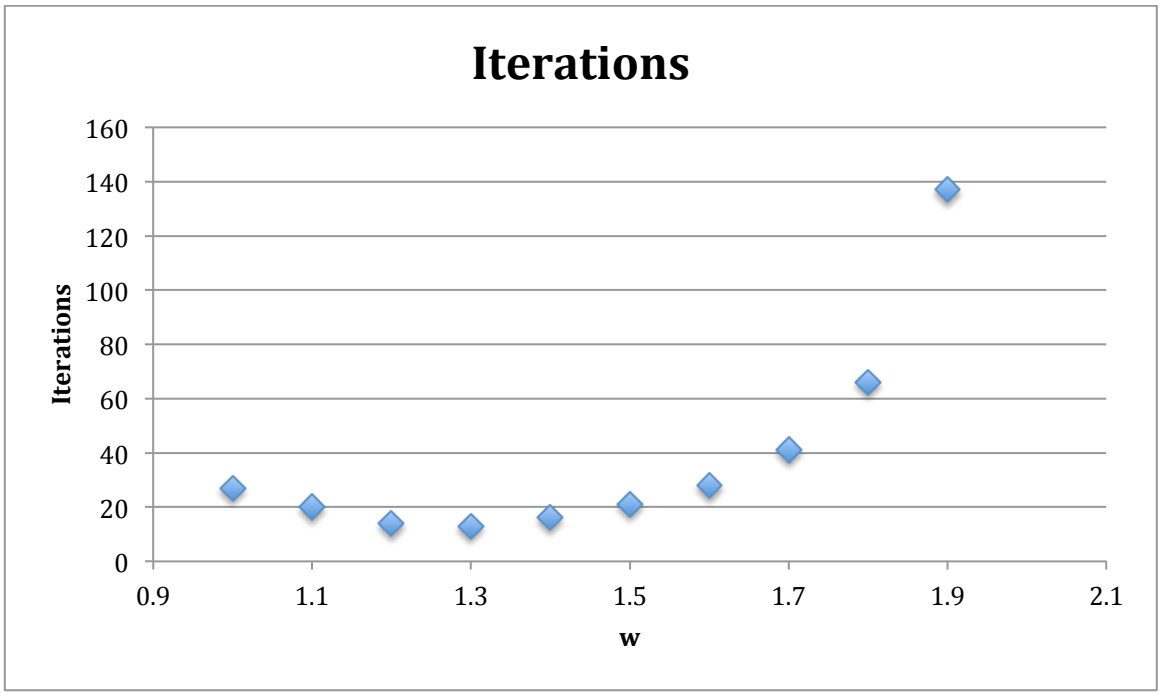
Question 3

a) The code is written in q3_Function.py has all the functions required to calculate the potential using both SOR and Jacobian:
computeMaxRes which computes the residue of each point, and updates the max residue value of the mesh. If this value is less than 10^{-5}, we stop the number of iterations.
genMesh generates the matrix using Dirchlet and Neuman conditions.
SOR calculates the potential using SOR method.
Jacobian does the same using jacobian method.
numIteration is the function that gives the number of iterations.
getPot gives the potential of the (x,y) coordinates

b) The potential at the point (x,y) = (0.06,0.04) is 42.46533V, using h = 0.02. The number of iterations for w are:

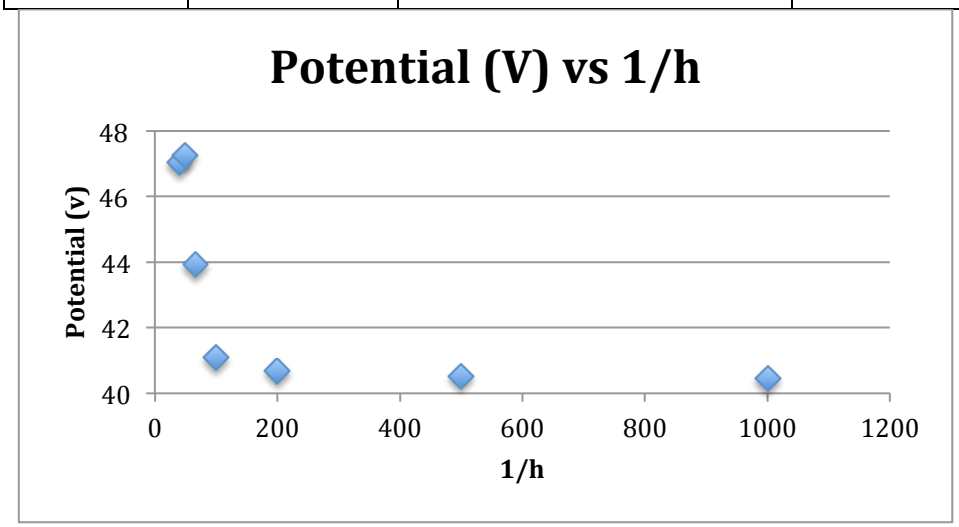
w	Iterations
1.0	27
1.1	20
1.2	14
1.3	13
1.4	16
1.5	21
1.6	28
1.7	41
1.8	66
1.9	137

Sharhad Bashar
260519664
ECSE 543



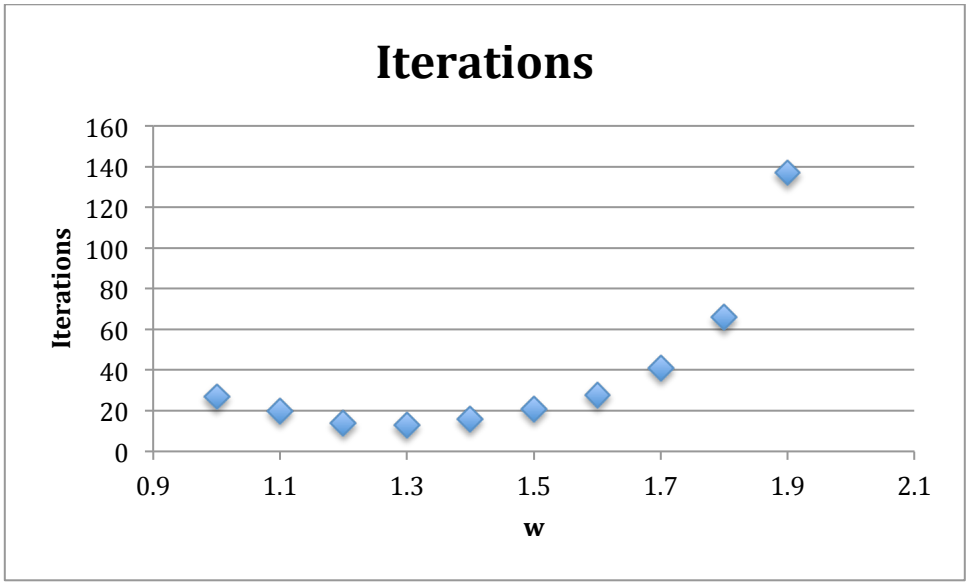
c) For my circuit, the least number of iterations occur at w = 1.25

h	1/h	Iterations	Potential (V)
0.025	40	10	47.035
0.02	50	11	47.265
0.015	66.67	20	43.913
0.01	100	57	41.081
0.005	200	213	40.689
0.002	500	1123	40.516
0.001	1000	3791	40.447



Sharhad Bashar
260519664
ECSE 543

As the value of h decreases, the number of iterations increases, and we get closer to a stable value for the potential. From my graphs, the value of potential approaches 40.446 V.

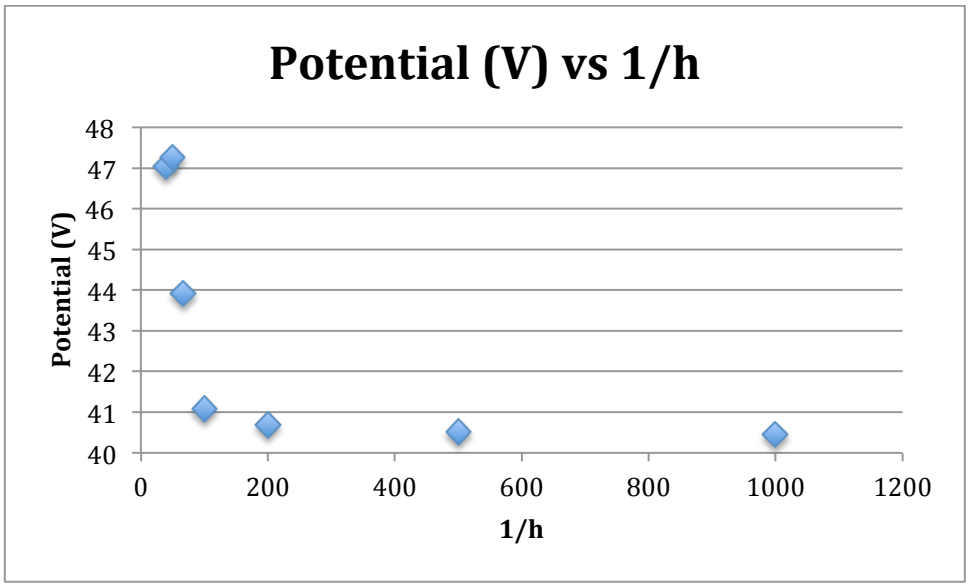


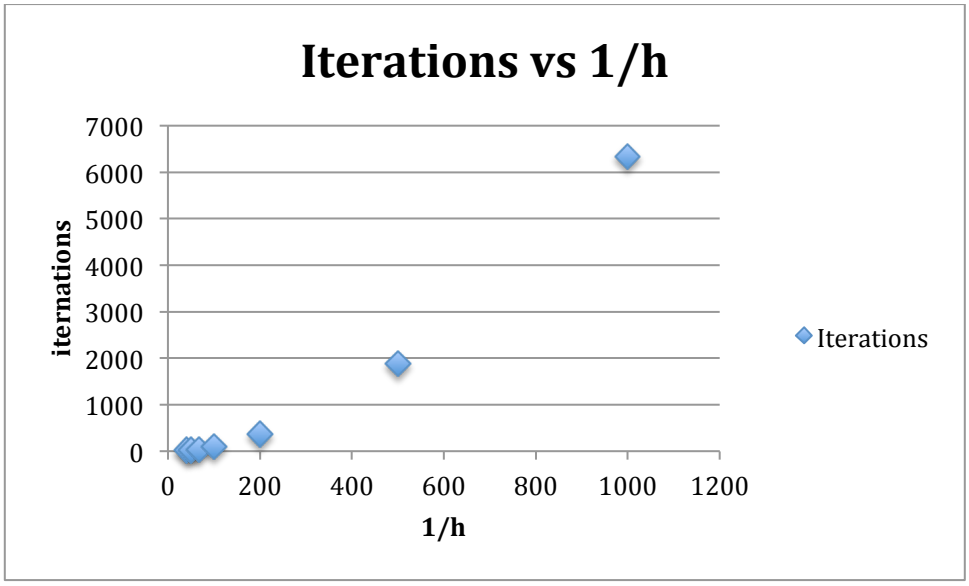
As expected, as h becomes smaller, number of iterations increase exponentially

d) Using the Jacobian method, gives the following values:

h	1/h	Iterations	Potential (V)
0.025	40	20	47.035
0.02	50	27	47.265
0.015	66.67	40	43.913
0.01	100	100	41.081
0.005	200	362	40.689
0.002	500	1883	40.516
0.001	1000	6338	40.447

Sharhad Bashar
260519664
ECSE 543





The same trend is seen in both SOR and Jacobian. As expected, the Jacobian takes much more time.

e) A new code was created in question_3E.py, which was used to calculate the potential. The results are as follows:

```
SOR
Number of iterations: 173
Potential: 45.83677574100205 V
```

Sharhad Bashar
260519664
ECSE 543

APPENDIX

```
# -*- coding: utf-8 -*-
"""
Sharhad Bashar
260519664
ECSE 543
Assignment 1
Question 1
basicDefinitions.py
"""

import math
from scipy import random
import numpy as np
import csv

#############################################################
############################
#Function that checks if a matrix is 1D or 2D
def is1Dor2D (A):
  while True:
    try:
      length = len(A[0]) #If true, A is 2D
      return A
      break
    except TypeError: #else A is 1D
      return [A]
      break
#############################################################
############################
#function that creates floats from lists
def list2float(A):
  length = len(A)
  floatA = [0 for x in range(length)]
  for i in range (length):
    numVal = ''
    stringVal = list(str(A[i]))
    for j in range (1,len(stringVal)-1,1):
      numVal = numVal + stringVal[j]
    floatVal = float(numVal)
    floatA [i] = floatVal
  return floatA
#############################################################
############################
```

Sharhad Bashar
260519664
ECSE 543

```
#Function that transposes a Matrix
def matTranspose(A):
    A = is1Dor2D(A)
    rowsA = len(A)
    colsA = len(A[0])
    C = [[0 for rows in range (rowsA)] for cols in range(colsA)]
    for i in range (colsA):
        for j in range (rowsA):
            C[i][j] = A[j][i]
    return C
##############################################################
############################
#Function that adds or subtracts two matricies
def matrixAddorSub(A,B,operation):
    A = is1Dor2D(A)
    B = is1Dor2D(B)
    rowsA = len(A)
    colsA = len(A[0])
    rowsB = len(B)
    colsB = len(B[0])
    if (rowsA == rowsB and colsA == colsB):
        C = [[0 for row in range(colsA)] for col in range(rowsA)]
        if (operation == 'a'):
            for i in range (rowsA):
                for j in range (colsA):
                    C[i][j] = A[i][j]+B[i][j]
        elif (operation == 's'):
            for i in range (rowsA):
                for j in range (colsA):
                    C[i][j] = A[i][j]-B[i][j]
    return C
##############################################################
############################
#Function that multiplies two matricies
def matrixMult (A, B):
    A = is1Dor2D(A)
    B = is1Dor2D(B)
    rowsA = len(A)
    colsA = len(A[0])
    rowsB = len(B)
    colsB = len(B[0])
    if (rowsA == colsB or colsA == rowsB):
        C = [[0 for row in range(colsB)] for col in range(rowsA)]
        for i in range(rowsA):
            for j in range(colsB):
```

```
            for k in range(colsA):
            # Create the result matrix
            # Dimensions would be rows_A x cols_B
               C[i][j] += A[i][k] * B[k][j]
    else:
     print ("Cannot multiply the two matrices. Incorrect dimensions.")
     return
    return C
#############################################################
############################
#Function that creates a diagonal matrix
def diogMat (A):
    length = len(A)
    floatA = [0 for x in range(length)]
    for i in range (length):
        numVal = ''
        stringVal = list(str(A[i]))
        for j in range (1,len(stringVal)-1,1):
            numVal = numVal + stringVal[j]
        floatVal = float(numVal)
        floatA [i] = floatVal
    diognalMatrix = [[0 for x in range(length)] for y in range(length)]
    for i in range (length):
        diognalMatrix[i][i] = 1/floatA[i]
    return diognalMatrix
#############################################################
############################
#Function that creates random A, given a length input
def randomSPD(length):
    A = [[0 for x in range(length)] for y in range(length)]
    L = random.rand(length,length)
    A = np.dot(L,L.T)
    return A
#############################################################
############################
#Function that performs the Cholesky decomposition and returns L
def cholesky(A,length):
    global sum
    L = [[0 for x in range(length)] for y in range(length)]
    for i in range(length):
        for k in range(i + 1):
            sum = 0
            for j in range(k):
                sum += L[i][j] * L[k][j]
            if (i == k):
```

```
            L[i][k] = math.sqrt(abs(A[i][i] - sum))
        else:
            L[i][k] = (A[i][k]-sum) / L[k][k]
    return L
#############################################################
############################
#Function that solves y in Ly = b
def forwElim (L,b,length):
    b = list2float(b)
    global sum
    y = [0 for x in range (length)]
    for i in range (length):
        sum = 0.00
        for j in range (i):
            sum += L[i][j] * y[j]
        y[i] = (b[i]-sum)/L[i][i]
    return y
#############################################################
############################
#Function that solves for x in L^Tx = y
def backSub (L,y,length):
    global sum
    X = [0 for x in range(length)]
    for i in range (length - 1, -1, -1):
        sum = 0
        for j in range (i + 1, length, 1):
            sum += L[j][i] * X[j]
        X[i] = (y[i]-sum) / L[i][i]
    return X
#############################################################
############################
#Solves the (AYA^T)Vn = A(J_YE) Equation
def voltageSolver(incidentMatrix, E, J, R):
    #Equation to solve: (A*Y*A^T)Vn = A(J-Y*E)
    #step_1 = Y*E
    #Step_2 = J-Step_1
    #Step_3 = A*Step_2
    #Step_4 = A^T
    #Step_5 = Y*Step_4
    #Step_6 = A*Step_5
    #Gives B
    Y = diogMat(R)
    Step_1 = matrixMult(Y,E)
    Step_2 = matrixAddorSub(J,Step_1,'s')
    Step_3 = matrixMult(incidentMatrix,Step_2)
```

```
    #Gives A
    Step_4 = matrixMult(incidentMatrix,Y)
    Step_5 = matTranspose(incidentMatrix)
    Step_6 = matrixMult(Step_4,Step_5)

    #computes the volatage
    length = len(Step_6)
    Step_7a = cholesky(Step_6,length)
    Step_7b = forwElim(Step_7a,Step_3, length)
    Step_7c = backSub(Step_7a,Step_7b, length)
    return Step_7c
##############################################################
############################
#Reads values from the csv file
def readCell(x, y):
    with open('testCircuit_1D.csv', 'r') as data:
        reader = csv.reader(data)
        yCount = 0
        for n in reader:
            if (yCount == y):
                rawCellValue = n[x]
                cellValue = float(rawCellValue)
                return cellValue
            yCount += 1
##############################################################
##############################
# -*- coding: utf-8 -*-
"""
Sharhad Bashar
ECSE 543
Assignment 1
OCt 17th, 2016
question_1.py
"""
from basicDefinitions import cholesky, forwElim, backSub, readCell, randomSPD,
voltageSolver
##############################################################
##############################################################
###
#Sets up the A,E,J,R Matricies for the five example circuits provided to us
incidentMatrix_1 = [readCell(1, 1),readCell(2, 1)]
E_1 = [[readCell(7, 1)],[readCell(7, 2)]]
J_1 = [[readCell(8, 1)],[readCell(8, 2)]]
R_1 = [[readCell(9, 1)],[readCell(9, 2)]]
```

Sharhad Bashar
260519664
ECSE 543

```
incidentMatrix_2 = [readCell(1, 4),readCell(2, 4)]
E_2 = [[readCell(7, 4)],[readCell(7, 5)]]
J_2 = [[readCell(8, 4)],[readCell(8, 5)]]
R_2 = [[readCell(9, 4)],[readCell(9, 5)]]

incidentMatrix_3 = [readCell(1, 7),readCell(2, 7)]
E_3 = [[readCell(7, 7)],[readCell(7, 8)]]
J_3 = [[readCell(8, 7)],[readCell(8, 8)]]
R_3 = [[readCell(9, 7)],[readCell(9, 8)]]

incidentMatrix_4 = [[readCell(1,10),readCell(2,10),readCell(3,10),readCell(4,10)],
                    [readCell(1,11),readCell(2,11),readCell(3,11),readCell(4,11)]]
E_4 = [[readCell(7,10)],[readCell(7,11)],[readCell(7,12)],[readCell(7,13)]]
J_4 = [[readCell(8,10)],[readCell(8,11)],[readCell(8,12)],[readCell(8,13)]]
R_4 = [[readCell(9,10)],[readCell(9,11)],[readCell(9,12)],[readCell(9,13)]]

incidentMatrix_5 =
[[readCell(1,15),readCell(2,15),readCell(3,15),readCell(4,15),readCell(5,15),readCel
l(6,15)],

[readCell(1,16),readCell(2,16),readCell(3,16),readCell(4,16),readCell(5,16),readCell
(6,16)],

[readCell(1,17),readCell(2,17),readCell(3,17),readCell(4,17),readCell(5,17),readCell
(6,17)],]
E_5 =
[[readCell(7,15)],[readCell(7,16)],[readCell(7,17)],[readCell(7,18)],[readCell(7,19)],
[readCell(7,20)]]
J_5 =
[[readCell(8,15)],[readCell(8,16)],[readCell(8,17)],[readCell(8,18)],[readCell(8,19)],
[readCell(8,20)]]
R_5 =
[[readCell(9,15)],[readCell(9,16)],[readCell(9,17)],[readCell(9,18)],[readCell(9,19)],
[readCell(9,20)]]
##########################################################################
##########################################################################
###
#for a test run
#A = [[4,12,-16],[12,37,-43],[-16,-43,98]]
#b = [100,200,150]
#x = [2541.667, -683.334, 116.667]
##########################################################################
############################
#Main
A,b,L,y,x = 0,0,0,0,0 #initialize the values to zero
```

Sharhad Bashar
260519664
ECSE 543

```
lengthInput = int(input("Please enter the length of A: "))
A = randomSPD(lengthInput) # creates a random SPD matrix of any requested
length
b = []
for i in range (lengthInput):
   count = i + 1
   if (count == 1):
      abbv = 'st'
   elif (count == 2):
      abbv = 'nd'
   elif (count == 3):
      abbv = 'rd'
   else:
      abbv = 'th'
   bVal = input("Please enter the " + str(count) + abbv + " value of b: ")
   b.append(float(bVal))

############################################################
############################
#function calls
chol = cholesky(A, lengthInput)
y = forwElim(chol, b, lengthInput)
x = backSub(chol,y,lengthInput)
############################################################
############################
#prints the input A
print ("")
print('A: ')
for i in range (lengthInput):
   print(A[i])
print("")
#prints the input b
print("b:")
print(b)
print("")
#prints the output x
print ('x:')
print (x)
print ('')
############################################################
############################
#1_D
print('Voltage for circuit 1 is: ' + str(voltageSolver(incidentMatrix_1,E_1,J_1,R_1)) +
'V')
```

```
print('Voltage for circuit 2 is: ' + str(voltageSolver(incidentMatrix_2,E_2,J_2,R_2)) +
'V')
print('Voltage for circuit 3 is: ' + str(voltageSolver(incidentMatrix_3,E_3,J_3,R_3)) +
'V')
print('Voltages for circuit 4 are: ' + str(voltageSolver(incidentMatrix_4,E_4,J_4,R_4))
+ 'V')
print('Voltages for circuit 5 are: ' + str(voltageSolver(incidentMatrix_5,E_5,J_5,R_5))
+ 'V')
##############################################################
##############################

# -*- coding: utf-8 -*-
"""
Sharhad Bashar
ECSE 543
Assignment 1
OCt 17th, 2016
question_2.py
"""

import time
from basicDefinitions import voltageSolver
from generateMesh import genMesh, EVector, JVector, RVector

#N = int(input("Please enter the size of mesh: ")) #Lets the user enter the desired
dimention of the mesh
for N in range(2,21,1):
    incidentMatrix = genMesh(N) #Generates the incident matrix for the input N
    #Generates the E, J and R vectors
    E = EVector(N)
    J = JVector(N)
    R = RVector(N)
    #Solves for the voltage across the mesh
    startTime = time.clock() #Starts the clock
    V = voltageSolver(incidentMatrix,E,J,R)
    endTime = time.clock() #Stops the clock
    #Vm = Vs*(Req/1+Req)
    Rreq = V[0]/(1-V[0]) #Solves for the equivalent resistance using voltage dividor

    timeTaken = endTime - startTime #Gives the execution time
    print("Equivaent Resistance for size "+ str(N) +" is: " + str(Rreq) + " ohms")
    print("Execution time: " + str(timeTaken) +"s")
    print('')

# -*- coding: utf-8 -*-
```

Sharhad Bashar
260519664
ECSE 543

```
"""
Sharhad Bashar
ECSE 543
Assignment 1
OCt 17th, 2016
generateMesh.py
"""
#################################################################
#################################
#Generates the incident matrix
def genMesh(meshDim):
    N = (meshDim + 1) #N is the number of nodes in one line of the mesh
    totalNodes = N ** 2 #totla nodes in the mesh circuit
    totalBranches = 2 * N * (N - 1) #total branches in the mesh circuit
    #Createing incident matrix, and filling it with 0's
    incMat = [[0 for rows in range (totalBranches + 1)] for cols in range (totalNodes)]
    #i is the horizontal rows, j is the vertical rows
    for i in range (1,(N + 1),1):
        for j in range (1,(N + 1),1):
            node = N * (j - 1) + i  #Numbering the nodes
            bUp = node + (N - 1) * (j - 1) #Branch above the node
            bDown = bUp - 1 #Branch below the node
            bLeft = bUp - N #Branch to the left of the node
            bRight = bUp + N - 1 #Branch to the right of the node
            #Populating the Incident Matrix
            #Leaving node = +1
            #Entering node = -1
            #Taking into account of the voltage source connected to the bottom left and
            #top right of the mesh
            incMat[0][totalBranches] = -1
            incMat[totalNodes - 1][totalBranches] = 1
            #Rest of the Mesh
            if (j == 1): #left most vertical branch
                incMat[node - 1][bRight - 1] = 1
                if (i == 1):
                    incMat[node - 1][bUp - 1] = 1
                elif (i == N):
                    incMat[node - 1][bDown - 1] = -1
                else:
                    incMat[node - 1][bUp - 1] = 1
                    incMat[node - 1][bDown - 1] = -1
            elif (j == N): #right most vertical branch
                incMat[node - 1][bLeft - 1] = -1
                if(i == 1):
                    incMat[node - 1][bUp - 1] = 1
```

```
            elif (i == N):
                incMat[node - 1][bDown - 1] = -1
            else:
                incMat[node - 1][bUp - 1] = 1
                incMat[node - 1][bDown - 1] = -1
        else:
            incMat[node - 1][bLeft - 1] = -1
            incMat[node - 1][bRight - 1] = 1
            if (i == 1):
                incMat[node - 1][bUp - 1] = 1
            elif (i == N):
                incMat[node - 1][bDown - 1] = -1
            else:
                incMat[node - 1][bUp - 1] = 1
                incMat[node - 1][bDown - 1] = -1
    incMatR = [[0 for rows in range (totalBranches + 1)] for cols in range (totalNodes -
1)]
    for i in range (totalBranches + 1):
        for j in range (totalNodes - 1):
            incMatR[j][i] = incMat[j][i]
    return incMatR

##############################################################
################################
#create E vector
def EVector(meshDim):
    N = (meshDim + 1) #N is the number of nodes in one line of the mesh
    totalBranches = 2 * N * (N - 1)
    E = [[0.00 for rows in range(1)]for rows in range (totalBranches + 1)]
    E[totalBranches][0] = 1.00 #voltage source of 1V in the last branch
    return E
##############################################################
################################
#create J vector
def JVector(meshDim):
    N = (meshDim + 1) #N is the number of nodes in one line of the mesh
    totalBranches = 2 * N * (N - 1)
    J = [[0.00 for rows in range(1)]for rows in range (totalBranches + 1)]
    return J
##############################################################
################################
#create R vector
def RVector(meshDim):
    N = (meshDim + 1) #N is the number of nodes in one line of the mesh
    totalBranches = 2 * N * (N - 1)
```

```
    R = [[1000 for rows in range(1)]for rows in range (totalBranches + 1)]
    R[totalBranches][0] = 1 #i ohm resistance connecting the source to the mesh
    return R
##################################################################
#################################

# -*- coding: utf-8 -*-
"""
Sharhad Bashar
ECSE 543
Assignment 1
Oct 17th, 2016
question_3.py
"""
from q_3Functions import genMesh, numIteration, getPot
##################################################################
############
#Fixed value of h, w changing
x = 0.06
y = 0.04
h = 0.02

print('SOR')
for w in range (10,20,1):
    w = float(w/10)
    initialMesh = (genMesh(h))
    finalMesh = numIteration(initialMesh,h,w,'s')
    print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
    print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
##################################################################
############
print('##################################################################
##')
#w = 1.25 is the value that gives least number of iterations
w = 1.25
h = 0.025
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
```

```
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

h = 0.02
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

h = 0.015
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

h = 0.01
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
```

```
print('')

h = 0.005
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

h = 0.002
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

h = 0.001
print('SOR')
print('h: ' + str(h))
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'s')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')
print('Jacobian')
initialMesh = (genMesh(h))
finalMesh = numIteration(initialMesh,h,w,'j')
print ('Potential: ' + str(getPot(finalMesh,x,y,h)) + ' V')
print('')

# -*- coding: utf-8 -*-
"""
Sharhad Bashar
ECSE 543
```

Sharhad Bashar
260519664
ECSE 543

Assignment 1
Oct 17th, 2016
q_3Functions.py
"""

```
#using the top right quater of the cable, due to symmetry
import math
#########################################################################################
#Generates the initial mesh, taking into considering the boundary conditions
def genMesh (h):
   cableHeight = 0.1
   cableWidth = 0.1
   coreHeight = 0.02
   coreWidth = 0.04
   corePot = 110.0
   nodeHeight = (int)(cableHeight/h + 1)
   nodeWidth = (int)(cableWidth/h + 1)
   #Create the mesh, with Dirchlet conditions
   mesh = [[corePot if x <= coreWidth/h and y <= coreHeight/h else 0.0 for x in
range(nodeWidth)] for y in range(nodeHeight)]
   #update the mesh to take into account the Neuman conditions
   rateofChangeX = 110*h/(cableWidth - coreWidth)
   rateofChangeY = 110*h/(cableHeight - coreHeight)
   for x in range ((int)(coreWidth/h) + 1, nodeWidth - 1):
      mesh[0][x] = mesh[0][x - 1] - rateofChangeX
   for y in range ((int)(coreHeight/h) + 1, nodeHeight - 1):
      mesh[y][0] = mesh[y - 1][0] - rateofChangeY
   return mesh
#########################################################################################
#The Equation that calculates SOR
def SOR(mesh,h,w):
   cableHeight = 0.1
   cableWidth = 0.1
   coreHeight = 0.02
   coreWidth = 0.04
   nodeHeight = (int)(cableHeight/h + 1)
   nodeWidth = (int)(cableWidth/h + 1)
   for y in range (1,nodeHeight - 1):
      for x in range (1,nodeWidth - 1):
         if (x > (int)(coreWidth/h) or y > (int)(coreHeight/h)):
            mesh[y][x] = (1 - w) * mesh[y][x] + (w/4) * (mesh[y][x-1] + mesh[y][x+1] +
mesh[y-1][x] + mesh[y+1][x])
   return mesh
```

Sharhad Bashar
260519664
ECSE 543

```
#############################################################################################
#The Equation that calculates Jacobian
def jacobian(mesh,h):
   cableHeight = 0.1
   cableWidth = 0.1
   coreHeight = 0.02
   coreWidth = 0.04
   nodeHeight = (int)(cableHeight/h + 1)
   nodeWidth = (int)(cableWidth/h + 1)
   for y in range (1,nodeHeight - 1):
      for x in range (1,nodeWidth - 1):
         if (x > (int)(coreWidth/h) or y > (int)(coreHeight/h)):
            mesh[y][x] = (1/4) * (mesh[y][x-1] + mesh[y][x+1] + mesh[y-1][x] +
mesh[y+1][x])
   return mesh

#############################################################################################
#Equation that computes the residue
def computeMaxRes(mesh,h):
   cableHeight = 0.1
   cableWidth = 0.1
   coreHeight = 0.02
   coreWidth = 0.04
   nodeHeight = (int)(cableHeight/h + 1)
   nodeWidth = (int)(cableWidth/h + 1)
   maxRes = 0
   for y in range(1, nodeHeight - 1):
      for x in range(1, nodeWidth - 1):
         if (x > coreWidth/h or y > coreHeight/h):
            #calculate the residue of each free point
            res = mesh[y][x-1] + mesh[y][x+1] + mesh[y-1][x] + mesh[y+1][x] - 4 *
mesh[y][x]
            res = math.fabs(res)
            if (res > maxRes):
               #Updates variable with the biggest residue amongst the free point
               maxRes = res
   return maxRes
#############################################################################################
#Function that computes the number of iterations
def numIteration (initialMesh,h,w,method):
   minRes = 0.0001
   iteration = 1
```

```
    if (method == 's'):
        mesh = SOR(initialMesh,h,w)
        while (computeMaxRes(mesh,h) >= minRes):
            mesh = SOR(mesh,h,w)
            iteration += 1
    elif (method == 'j'):
        mesh = jacobian(initialMesh,h)
        while (computeMaxRes(mesh,h) >= minRes):
            mesh = jacobian(mesh,h)
            iteration += 1
    print ('Number of iterations: '+ str(iteration))
    return(mesh)
############################################################
##########################################
#Function that returns the potential at a free node
def getPot(mesh, x, y, h):
    cableHeight = 0.1
    cableWidth = 0.1
    nodeHeight = (int)(cableHeight/h + 1)
    nodeWidth = (int)(cableWidth/h + 1)
    xNode = int(nodeWidth - x/h - 1)
    yNode = int(nodeHeight - y/h - 1)
    return mesh[yNode][xNode]
############################################################
##########################################

# -*- coding: utf-8 -*-
"""
Sharhad Bashar
ECSE 543
Assignment 1
Oct 17th, 2016
question_3E.py
"""
import math
############################################################
##########################################
#Generates the initial mesh, taking into considering the boundary conditions
def genMesh (verLine,horLine):
    cableHeight = 0.1
    cableWidth = 0.1
    coreHeight = 0.02
    coreWidth = 0.04
    corePot = 110.0
    #Create the mesh, with Dirchlet conditions
```

```
   mesh = [[corePot if x <= coreWidth and y <= coreHeight else 0.0 for x in verLine]
for y in horLine]
   #update the mesh to take into account the Neuman conditions
   rateofChangeX = 110/(cableWidth - coreWidth)
   rateofChangeY = 110/(cableHeight - coreHeight)
   for x in range (len(verLine)):
      if (verLine[x] > coreWidth):
         mesh[0][x] = 110 - rateofChangeX * (verLine[x] - coreWidth)
   for y in range (len(horLine)):
      if (horLine[y] > coreHeight):
         mesh[y][0] = 110 - rateofChangeY * (horLine[y] - coreHeight)
   return mesh
##########################################################
##########################################
#The Equation that calculates SOR
def SOR(mesh,verLine,horLine):
   coreHeight = 0.02
   coreWidth = 0.04
   for y in range (1,len(horLine) - 1):
      for x in range(1,len(verLine) - 1):
         if (verLine[x] > coreWidth or horLine[y] > coreHeight):
            a1 = verLine[x] - verLine[x-1]
            a2 = verLine[x+1] - verLine[x]
            b1 = horLine[y+1] - horLine[y]
            b2 = horLine[y] - horLine[y-1]
            mesh[y][x] = (mesh[y][x-1]/(a1 * (a1 + a2)) + mesh[y][x+1]/(a2 * (a1 + a2))
+ \
                  mesh[y-1][x]/(b1 * (b1 + b2)) + mesh[y+1][x]/(b2 * (b1 + b2))) / \
                  (1/(a1 * a2) + 1/(b1 * b2))
   return mesh
##########################################################
##########################################
#Equation that computes the residue
def computeMaxRes(mesh,horLine,verLine):
   coreHeight = 0.02
   coreWidth = 0.04
   maxRes = 0
   for y in range (1,len(horLine) - 1):
      for x in range (1,len(verLine) - 1):
         if (verLine[x] > coreWidth or horLine[y] > coreHeight):
            a1 = verLine[x] - verLine[x-1]
            a2 = verLine[x+1] - verLine[x]
            b1 = horLine[y+1] - horLine[y]
            b2 = horLine[y] - horLine[y-1]
```

```
            res = (mesh[y][x-1]/(a1 * (a1 + a2)) + mesh[y][x+1]/(a2 * (a1 + a2)) +
mesh[y-1][x]/(b1 * (b1 + b2)) + mesh[y+1][x]/(b2 * (b1 + b2))) - (1/(a1 * a2) +
1/(b1 * b2))*mesh[y][x]
            res = math.fabs(res)
            if (res > maxRes):
               #Updates variable with the biggest residue amongst the free point
               maxRes = res
   return maxRes
##########################################################
##########################################
def numIteration (initialMesh,horLine,verLine):
   minRes = 0.0001
   mesh = SOR(initialMesh,horLine,verLine)
   iteration = 1
   while (computeMaxRes(mesh,horLine,verLine) >= minRes):
      mesh = SOR(mesh,horLine,verLine)
      iteration += 1
   print ('Number of iterations: '+ str(iteration))
   return(mesh)
##########################################################
##########################################
def getPot(mesh, x, y,verLine,horLine):
   xNode = verLine.index(x)
   yNode = horLine.index(y)
   return mesh[yNode][xNode]

horLine = [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]
verLine = [0, 0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.1]

x = 0.06
y = 0.04
print('SOR')
initialMesh = genMesh(horLine,verLine)
finalMesh = numIteration(initialMesh,horLine,verLine)
print ('Potential: ' + str(getPot(finalMesh,x,y,verLine,horLine)) + ' V')
```