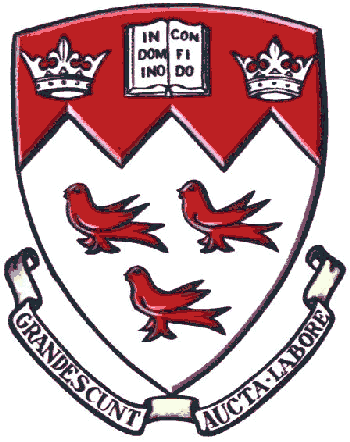
****

**Assignment 1**

**ECSE 543**

**Sharhad Bashar**

**260519664**

**Oct 17th, 2016**

**Question 1**

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

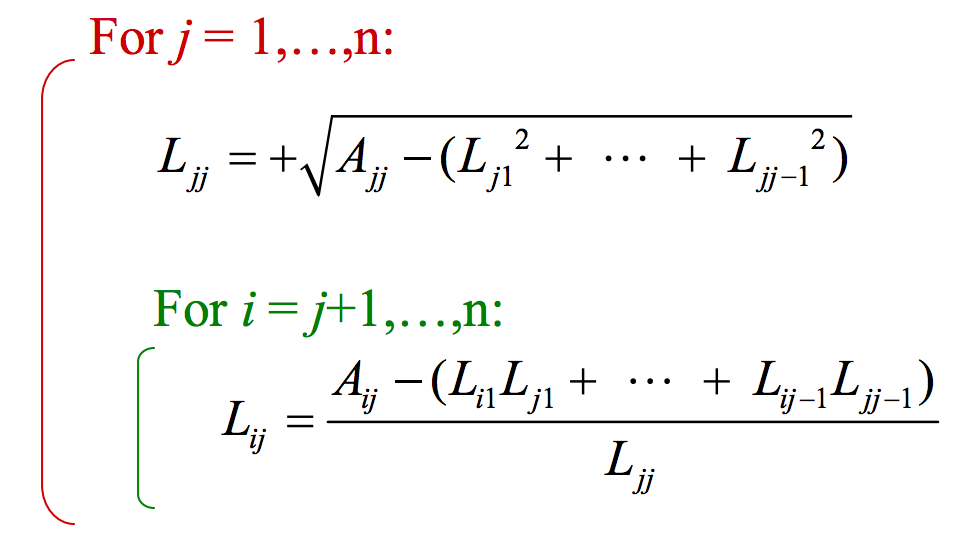
Code included under appendix.

Solving for x is done in three steps.

Step 1: A = LLT

Step 2: Ly = b

Step 3: LTx = 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: 

This equation gives us with L matrix of size n, and has a computational cost of O(n3)

**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:

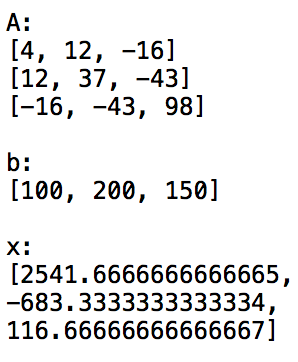
This equation returns y, which is a vector of size n. The computational cost of this step is O(n2)

**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:

This equation returns the answer to the equation Ax = b. x is a vector if size n, and costs O(n2) to compute.

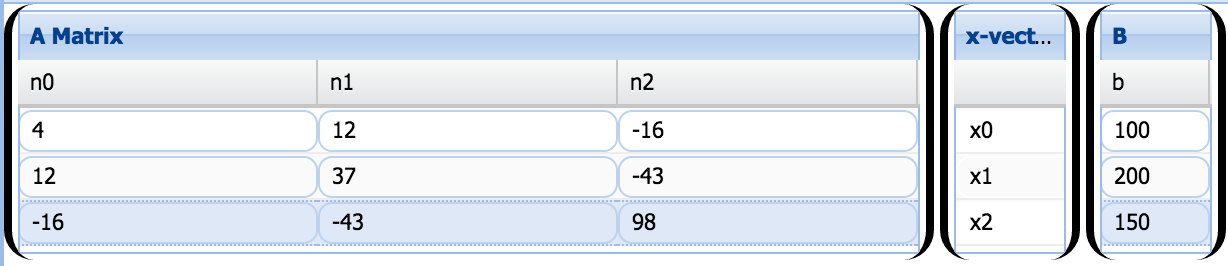
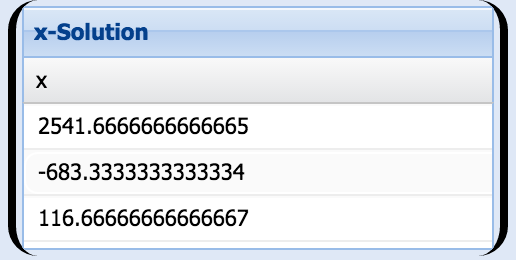
To test the code, the following A and b were inputted to the function:

After running the code, the following result was outputted:



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:



1. **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 = LLT. This is the equation used to create the symmetric positive definite matrix of order n.

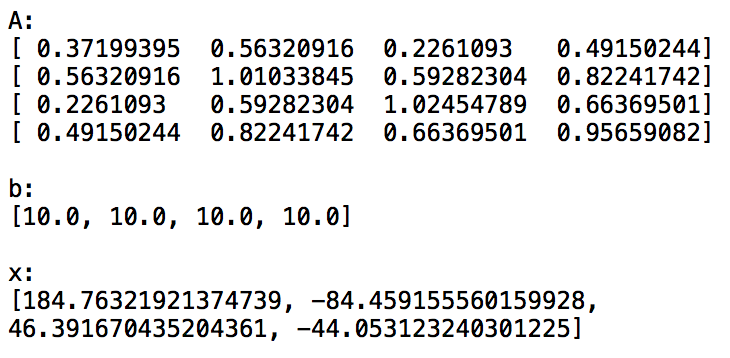
The code for this can be found in the file basicDefinition.py, in the function randomSPD. The input for the function is the matrix size n. This n is taken and used to create a lower triangular matrix L and fill it up with random values. Then the generated L is multiplied with its transpose, to give us the SPD matrix A. Since this function is used solely for testing, using numpy’s multiplication and transpose functions created the matrix.

1. **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.**

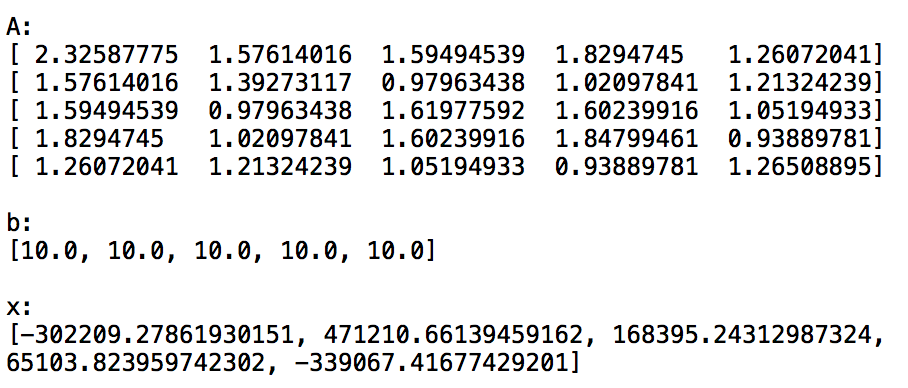
For the testing, we used values of n = 4,5,6,7,8.

These are the following results:

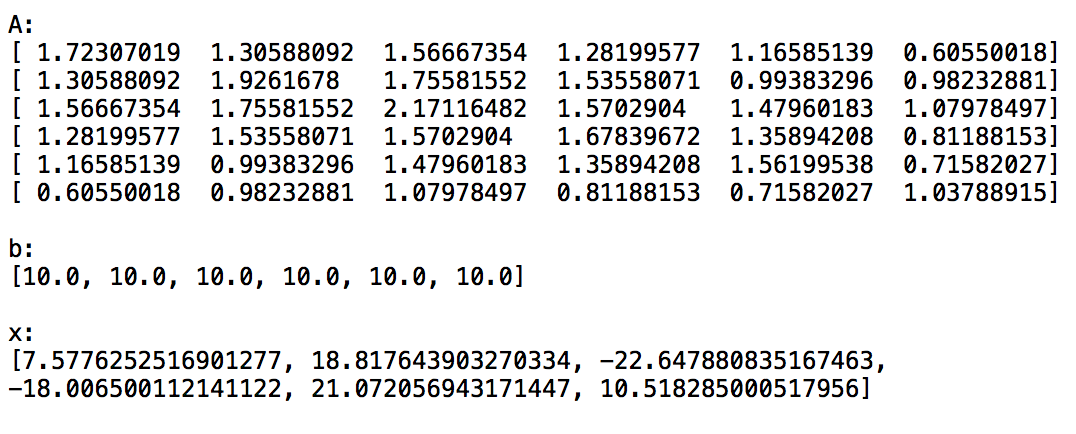
n = 4



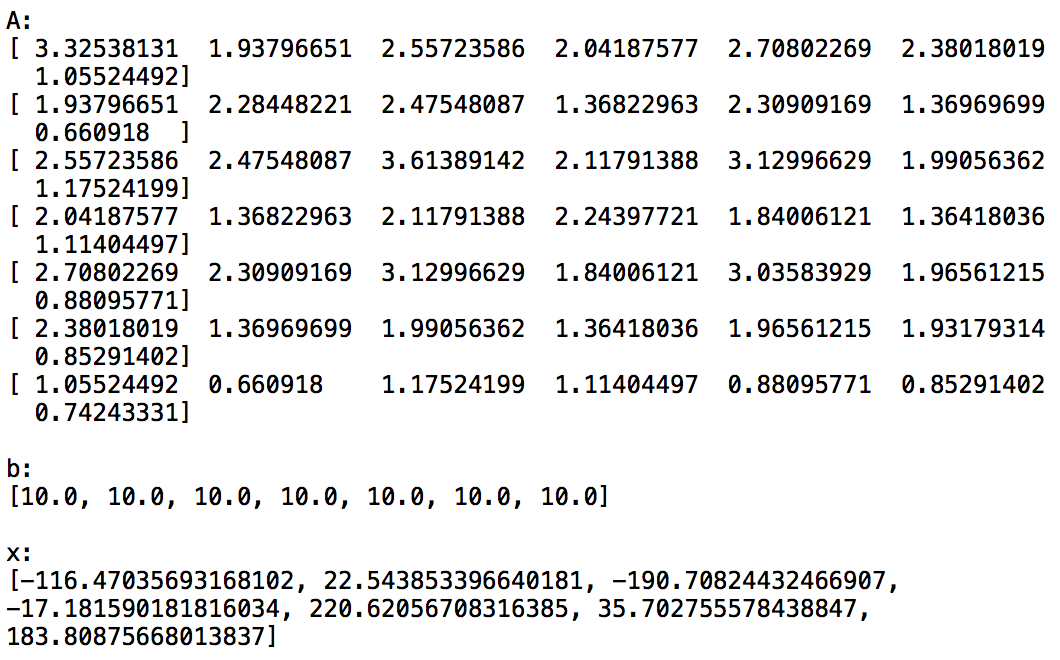
n = 5

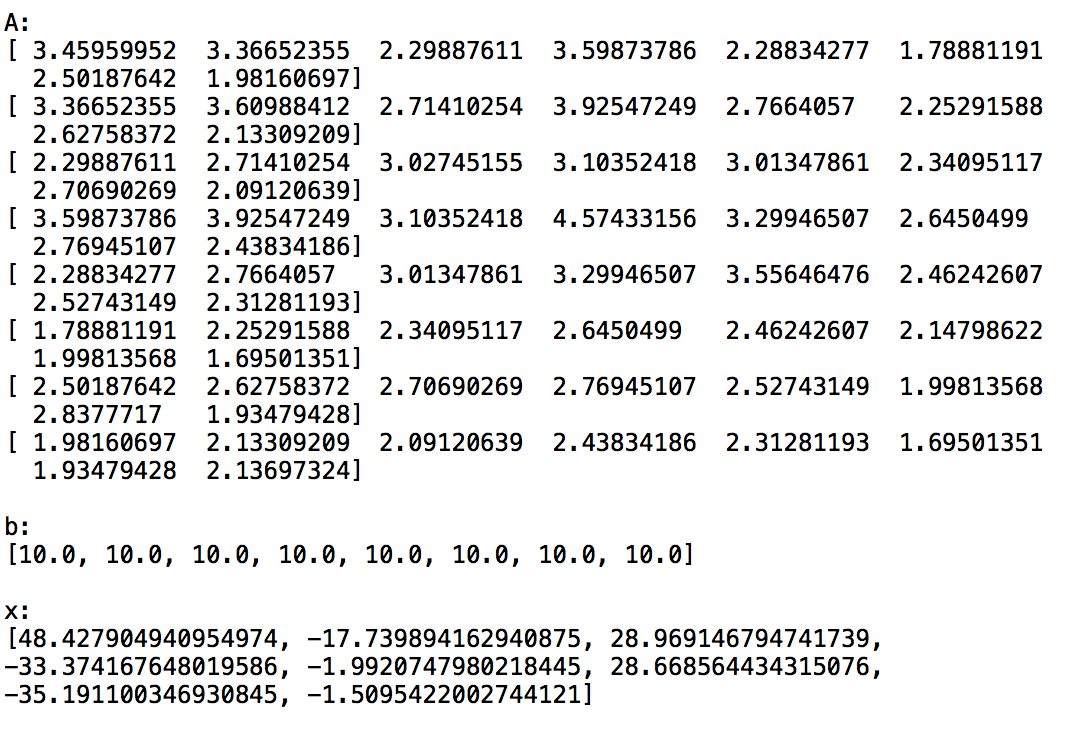


n = 6



n = 7



n = 8 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.

1. **Write a program that reads from a file a list of network branches (Jk, Rk, Ek) 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 Jk, Rk and Ek 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.

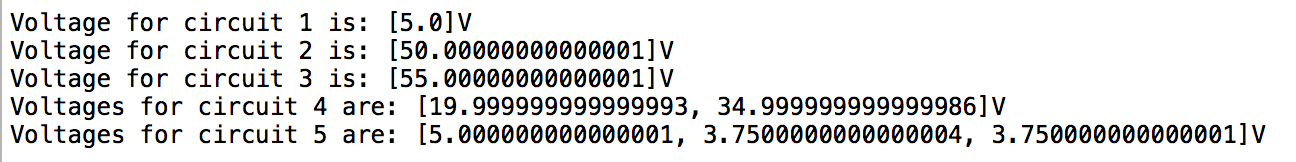
The incident matrix and the three vectors were fed into this equation to solve for voltages at the nodes:

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

AYAT 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 basicDefninition.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:



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 |

**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.**

1. **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

1. 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.

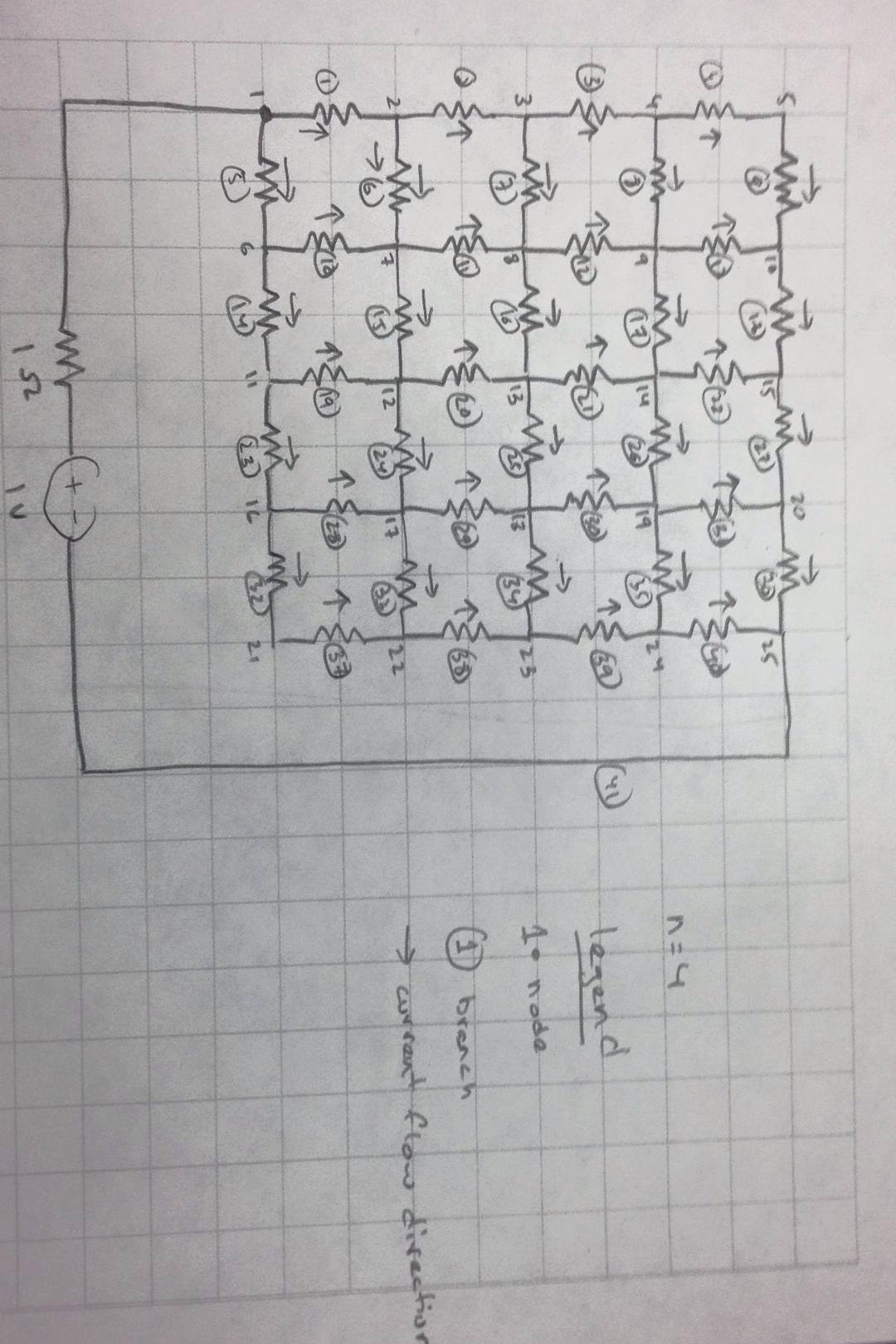
**EVector** returned a vector of the voltage sources present in the circuit, and since there was only one, it was a vector of zeros except for the very last value, which is 1.

**JVector** returned a vector of zeros, since there was no current sources

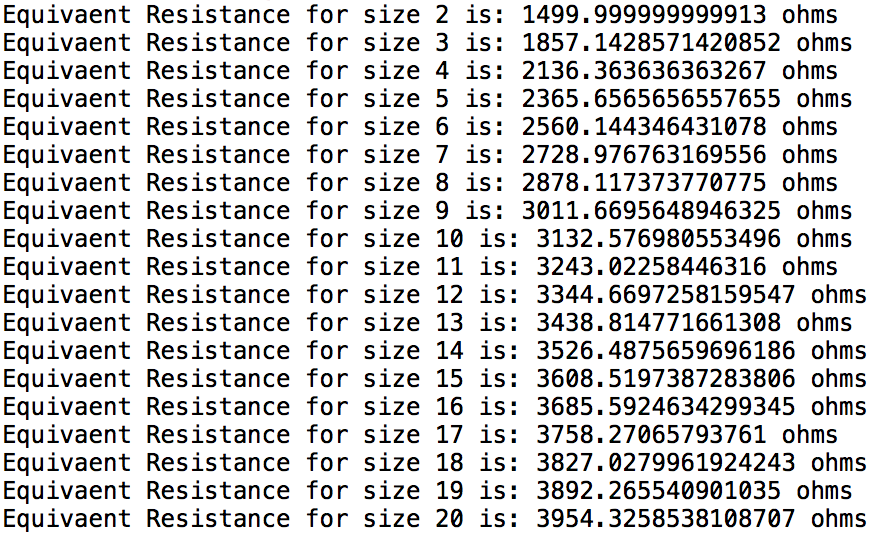
**RVector** was a vector of resistances, which in this case was a vector of 1000, except for the last one which is 1, to take into account the 1 ohm resistance that connects the source to the mesh.

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



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



1. **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(n3). For the mesh, n = total number of nodes, which is N2.

So the overall time complexity should be O(N6). 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.

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

**Question 3**

1. 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

1. 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 |

1. 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 |

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.

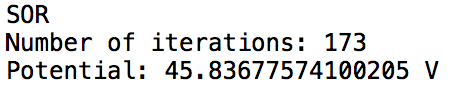
As expected, as h becomes smaller, number of iterations increase exponentially

1. 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 |

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

1. A new code was created in question\_3E.py, which was used to calculate the potential. The results are as follows:



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

######################################################################################

#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)]]

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),readCell(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

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

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

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

#######################################################################################################

#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')