Assignment 1

Mubaslat, Laith

McGill University

Assignment 1

Problem 1

A python file for basic function for creating Matrices, Transposing them, doing Matrix Multiplication and other procedures was created and place in the python file "basicfunctions.py"

a) The code used to solve this problem can be found in the Appendix. The first step is to perform Cholesky decomposition on the matrix A into the product of a lower triangular matrix and its transpose. This is possible under the assumption that A is a positive definite. Thus, the code for the decomposition would exit and print an error message in case A was not positive definite. The code identifies that by a number of ways including the existence of a diagonal element with a value of zero or a negative element in the matrix A.

   Following the Cholesky decomposition the problem is separate into two parts one exploiting the lower triangular matrix L to find y and then using that value for y with the transpose of L to compute x.

b) 10 matrices were constructed. With the target of the matrices in question being positive definite they had to be constructed in a special way. Lower triangular real matrices with positive diagonal entries (invertible) where constructed. These Matrices were then transposed. Accordingly the product of these matrices LL^T is positive definite (which also means that it's non singular).

c) A vector x of size n was created for each matrix An found earlier. The product An.x yields the vector b. later that vector b and matrix A were used to compute a xhat which was compared and found to be identical to the values used in making b earlier.

```
In [58]: A2
Out[58]: [[4, 6], [6, 25]]

In [59]: A3
Out[59]: [[4, 6, 8], [6, 25, 32], [8, 32, 77]]

In [60]: A4
Out[60]: [[4, 6, 8, 10], [6, 25, 32, 39], [8, 32, 77, 92], [10, 39, 92, 174]]
```

```
In [61]: A5
Out[61]:
[[4, 6, 8, 10, 12],
 [6, 25, 32, 39, 46],
 [8, 32, 77, 92, 107],
 [10, 39, 92, 174, 200],
 [12, 46, 107, 200, 330]]
```

```
In [62]: A6
Out[62]:
[[4, 6, 8, 10, 12, 14],
 [6, 25, 32, 39, 46, 53],
 [8, 32, 77, 92, 107, 122],
 [10, 39, 92, 174, 200, 226],
 [12, 46, 107, 200, 330, 370],
 [14, 53, 122, 226, 370, 559]]
```

```
In [63]: A7
Out[63]:
[[4, 6, 8, 10, 12, 14, 16],
 [6, 25, 32, 39, 46, 53, 60],
 [8, 32, 77, 92, 107, 122, 137],
 [10, 39, 92, 174, 200, 226, 252],
 [12, 46, 107, 200, 330, 370, 410],
 [14, 53, 122, 226, 370, 559, 616],
 [16, 60, 137, 252, 410, 616, 875]]
```

```
In [64]: A8
Out[64]:
[[4, 6, 8, 10, 12, 14, 16, 18],
 [6, 25, 32, 39, 46, 53, 60, 67],
 [8, 32, 77, 92, 107, 122, 137, 152],
 [10, 39, 92, 174, 200, 226, 252, 278],
 [12, 46, 107, 200, 330, 370, 410, 450],
 [14, 53, 122, 226, 370, 559, 616, 673],
 [16, 60, 137, 252, 410, 616, 875, 952],
 [18, 67, 152, 278, 450, 673, 952, 1292]]
```

```
In [65]: A9
Out[65]:
[[4, 6, 8, 10, 12, 14, 16, 18, 20],
 [6, 25, 32, 39, 46, 53, 60, 67, 74],
 [8, 32, 77, 92, 107, 122, 137, 152, 167],
 [10, 39, 92, 174, 200, 226, 252, 278, 304],
 [12, 46, 107, 200, 330, 370, 410, 450, 490],
 [14, 53, 122, 226, 370, 559, 616, 673, 730],
 [16, 60, 137, 252, 410, 616, 875, 952, 1029],
 [18, 67, 152, 278, 450, 673, 952, 1292, 1392],
 [20, 74, 167, 304, 490, 730, 1029, 1392, 1824]]
```

```
In [66]: A10
Out[66]:
[[4, 6, 8, 10, 12, 14, 16, 18, 20, 22],
 [6, 25, 32, 39, 46, 53, 60, 67, 74, 81],
 [8, 32, 77, 92, 107, 122, 137, 152, 167, 182],
 [10, 39, 92, 174, 200, 226, 252, 278, 304, 330],
 [12, 46, 107, 200, 330, 370, 410, 450, 490, 530],
 [14, 53, 122, 226, 370, 559, 616, 673, 730, 787],
 [16, 60, 137, 252, 410, 616, 875, 952, 1029, 1106],
 [18, 67, 152, 278, 450, 673, 952, 1292, 1392, 1492],
 [20, 74, 167, 304, 490, 730, 1029, 1392, 1824, 1950],
 [22, 81, 182, 330, 530, 787, 1106, 1492, 1950, 2485]]
```

```
In [98]: x2
Out[98]: [4, 15]

In [99]: x3
Out[99]: [4, 15, 26]

In [100]: x4
Out[100]: [4, 15, 26, 33]

In [101]: x5
Out[101]: [4, 15, 26, 33, 40]

In [102]: X2hat
Out[102]: [[4.0], [15.0]]

In [103]: X3hat
Out[103]: [[4.0], [15.0], [26.0]]

In [104]: X4hat
Out[104]: [[4.0], [15.0], [26.0], [33.0]]

In [105]: X5hat
Out[105]: [[4.0], [15.0], [26.0], [33.0], [40.0]]
```

```
In [116]: x6
Out[116]: [4, 15, 26, 33, 40, 51]

In [117]: x7
Out[117]: [4, 15, 26, 33, 40, 51, 70]

In [118]: x8
Out[118]: [4, 15, 26, 33, 40, 51, 70, 77]

In [119]: x9
Out[119]: [4, 15, 26, 33, 40, 51, 70, 77, 92]

In [120]: X6hat
Out[120]: [[4.0], [15.0], [26.0], [33.0], [40.0], [51.0]]

In [121]: X7hat
Out[121]: [[4.0], [15.0], [26.0], [33.0], [40.0], [51.0], [70.0]]

In [122]: X8hat
Out[122]: [[4.0], [15.0], [26.0], [33.0], [40.0], [51.0], [70.0], [77.0]]

In [123]: X9hat
Out[123]: [[4.0], [15.0], [26.0], [33.0], [40.0], [51.0], [70.0], [77.0], [92.0]]
```

```
x10
[4, 15, 26, 33, 40, 51, 70, 77, 92, 87]

X10hat
[[4.0], [15.0], [26.0], [33.0], [40.0], [51.0], [70.0], [77.0], [92.0],
```

Figure 1: An Matrices and x vectors used to generate b vectors which were then used to compute xnhat

d) Two separate programs were made. The first had the purpose of taking the matrices Rk,Jk,Ek, and the reduced incidence matrix and produce a network text file. The second would read that network file(or any text file of the same organization) and produce the matrices Rk,Jk, EK and the reduced incidence matrix. The way the files are orinizes are as follows:

- First line contains the network name
- Fourth line contains the number of nodes
- The sixth line contains the number of branches
- Starting from line 7 each line represent a branch providing the branch's respective Jk,Ek,Rk values
- After the completion of the branch data the reduced incidence matrix follows which is then follow by the letter 'E' which is used to terminate the program.

An example of a network file can be shown bellow:

```
.Network Name: ckt1
 JK,RK,EK and reduced A Matrix Parameters
number of nodes:
2;
number of branches
2;
Jk Rk Ek
0.000000,0.100000,10.000000;
0.000000,0.100000,0.000000;
A
-1.000000,-1.000000;
E
```

Figure 3: Network File of Ckt1

```
In [63]: X1
Out[63]: [[5.0]]

In [64]: X2
Out[64]: [[50.00000000000001]]

In [65]: X3
Out[65]: [[55.00000000000001]]

In [66]: X4
Out[66]: [[19.999999999999993], [34.999999999999986]]

In [67]: X5
Out[67]: [[5.000187511719483], [3.7502343896493544], [3.7502343896493544]]
```

Figure 2: Example Circuit Solution Obtained by Cholesky Decomposition

Problem 2

a) A small file was used to generate the incidence matrix and then reduce it according the terminals that we desire to measure the resistance from. Accordingly, the line corresponding to the node (n*n-n) was removed. That same program was then used to produce reduced incidence matrices for N=2,3,4,….,15.

In order to measure the resistance between the nodes (n-1) and (n*n-n) a dummy current source of 1 A was used. A parallel resistance of 10kOhm was used which thus reduced the matrix Y into the scalar (1/10kOhm) multiplied by the identity matrix. The problem to be solved is thus:

$$\frac{1}{10{,}000} A\, A^T\, v = A\, J_k$$

Cholesky Decomposition was then used to solve for L, followed by y and then x for N=2,3,4,……,15. The results can be seen bellow.

b) $O(n^3) = O(N^6)$ is the theoretical time needed for cholskey decomposition (N unknowns).It should also be noted that solving for y and x takes a theoretical time $O(n^2)$ for each of them. However the larger N gets the more that Cholesky's theoretical time dominates the time needed for solving the system of equations Ax=b. There is a clear mismatch between the practical results and the theoretical time in some instances as can be seen in figure 4. That can be attributed to the different operating condition of the computer itself considering the time computed would differ every time the solver is started again. However, it does seem that for larger networks the solver follows the theoretical time more faithfully.

```
In [152]: time
Out[152]:
[0,
 0,
 datetime.timedelta(0, 0, 1820),
 datetime.timedelta(0, 0, 521),
 datetime.timedelta(0, 0, 2904),
 datetime.timedelta(0, 0, 10164),
 datetime.timedelta(0, 0, 24536),
 datetime.timedelta(0, 0, 53132),
 datetime.timedelta(0, 0, 110009),
 datetime.timedelta(0, 0, 383787),
 datetime.timedelta(0, 0, 452975),
 datetime.timedelta(0, 0, 744658),
 datetime.timedelta(0, 1, 307058),
 datetime.timedelta(0, 2, 244809),
 datetime.timedelta(0, 3, 333851),
 datetime.timedelta(0, 5, 149453)]
```

Figure 5:Simulation Time

```
In [153]: Rnet
Out[153]:
[0,
 0,
 10000.000000000004,
 15000.000000000018,
 18571.428571428638,
 21363.63636363649,
 23656.565656565774,
 25601.443464314583,
 27289.76763169849,
 28781.173737717007,
 30116.695648965637,
 31325.76980554853,
 32430.225844643734,
 33446.69725816847,
 34388.147716622225,
 35264.87565970423]
```

Figure 4: R(N)

c) For n=2 a half bandwidth of 2 can be observed. However, for n larger than 2 the maximum halfbandwidth is at (n+1).

The time for solving for each x and y individually is reduced from $O(n^2)$ to $O(b^{-2}n)$ with b^{-2} being the means square of the halfbandwidth values for a given n.

It can be clearly seen how by taking advantage of sparsity the time needed is much lower values of N is concerned. However, the higher N gets that advantage starts to disappear considering how the simulation time is dominated by the $O(n^3)$ of the Cholskey Decomposition.

```
In [150]: time
Out[150]:
[0,
 0,
 datetime.timedelta(0, 0, 193),
 datetime.timedelta(0, 0, 584),
 datetime.timedelta(0, 0, 4152),
 datetime.timedelta(0, 0, 9336),
 datetime.timedelta(0, 0, 25894),
 datetime.timedelta(0, 0, 54235),
 datetime.timedelta(0, 0, 115706),
 datetime.timedelta(0, 0, 230690),
 datetime.timedelta(0, 0, 438690),
 datetime.timedelta(0, 0, 785357),
 datetime.timedelta(0, 1, 312495),
 datetime.timedelta(0, 2, 148573),
 datetime.timedelta(0, 3, 384963),
 datetime.timedelta(0, 5, 275388)]
```

Figure 7:Simulation Time(Taking advantage of Sparsity)

```
In [156]: RnetSparse
Out[156]:
[0,
 0,
 10000.000000000004,
 15000.000000000018,
 18571.428571428638,
 21363.63636363649,
 23656.565656565774,
 25601.443464314583,
 27289.76763169849,
 28781.173737717007,
 30116.695648965637,
 31325.76980554853,
 32430.225844643734,
 33446.69725816847,
 34388.147716622225,
 35264.87565970423]
```

Figure 6: R(N) taking advantage of Sparsity

d)

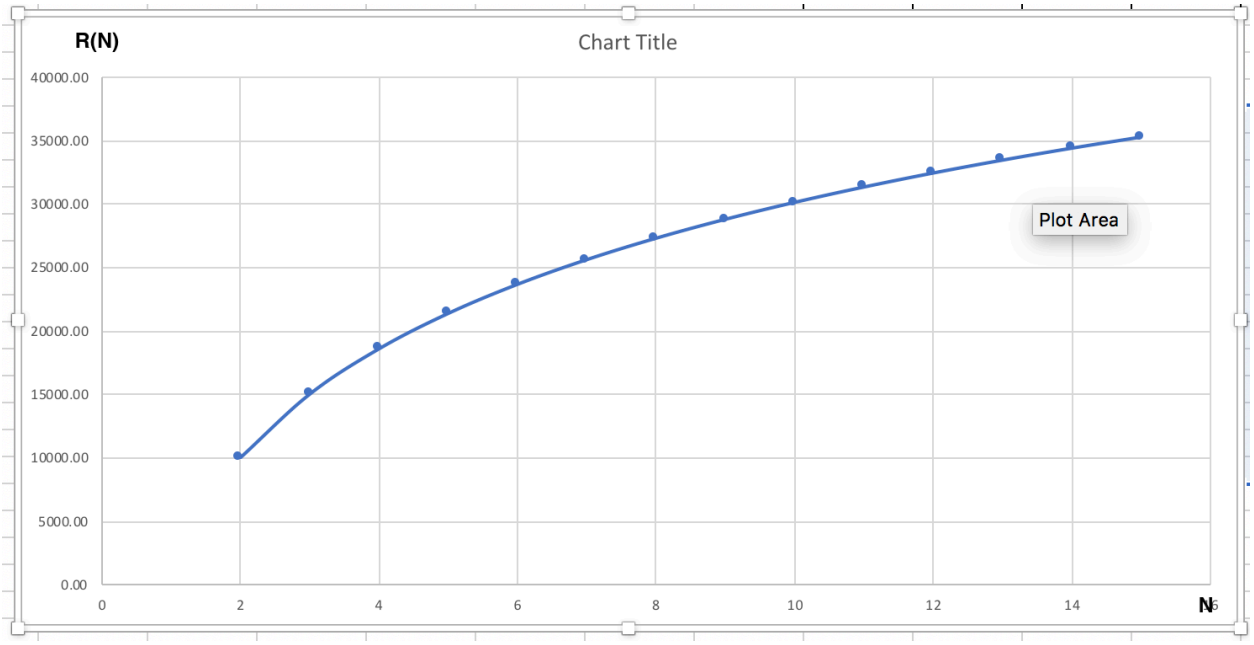


Figure 8: R(N) vs N

The function is: R(N) = C1 (1- exp(-C2*N^2))

Values for C1, C2 that would give a close curve are 35000 and 0.1 respectively

Problem 3

a) The program written to solve this problem is segmented into several parts. It is designed for the variable spacing problem, but it however can be used for both equal and variable spacing's. It checks for errors in the variable spacing chosen and the values for h, deltax and delta to ensure their suitability to the problem geometry and inclusion of the target output point. After having checked the inputs for any errors; first Gauss Seidel is computed which is then followed by the SOR relaxation. Following that the residuals are checked for the tolerance inputed at the beginning of the program. The moment all the residuals go bellow the preset tolerance the program terminates and returns the time, solution matrix and voltage at point (0.06,0.04).

b)

Figure 9: Iteration #, w and the Voltage at point(0.06,0.04)

w	Iterations	V(x,y)
1	45	40.5264911
1.1	40	40.5264907
1.2	36	40.5264911
1.3	33	40.5264935
1.4	30	40.5264937
1.5	37	40.5265026
1.6	51	40.5265026
1.7	77	40.5265026
1.8	127	40.5265026
1.9	289	40.5265026

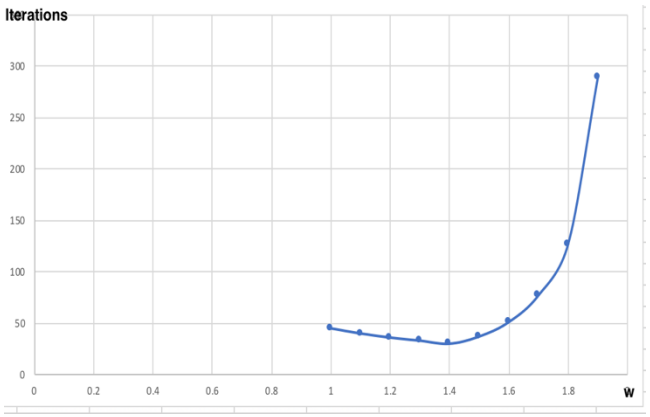


Figure 10: Iterations to Convergence vs w

Optimal w: 1.4

c)

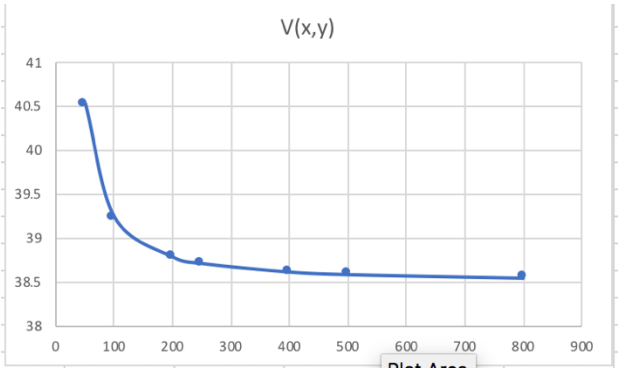


Figure 12: V at point (0.06,0.04) vs 1/h

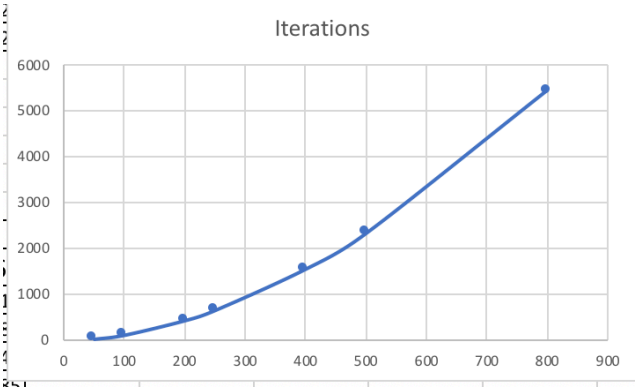


Figure 11: # of Iterations vs 1/h

It can be clearly seen in figure 12 how decreasing the spacing h yields a more accurate value for the voltage we are attempting to solve for. However, looking at Figure 11 we can see that the cost of decreasing the spacing h is excessive considering how the computational demand seems to increase in a level that greatly overweighs the improvement in voltage estimation provided by the decreased spacing h. Expected voltage is around 38.52.

Figure 13: w= 1.4; Change in Number of iterations and voltage measurments in response to decreasing h

h	1/h	Iterations	V(x,y)
0.02	50	30	40.5265
0.01	100	118	39.2382615
0.005	200	434	38.7881889
0.004	250	656	38.7150047
0.0025	400	1551	38.6172785
0.002	500	2327	38.5882754
0.00125	800	5431	38.5479213
0.001	1000	8094	38.5346504

d)

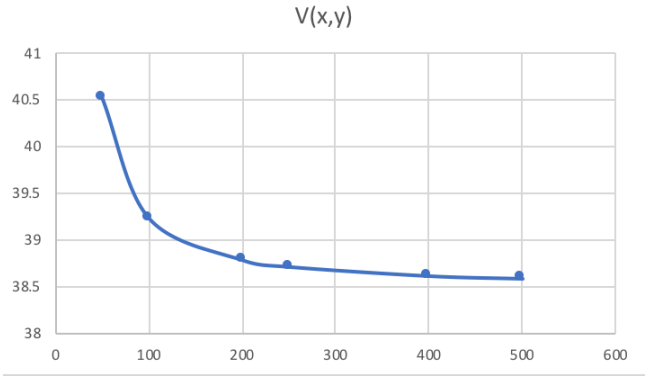


Figure 14:Jacobi Method- V(x,y) vs 1/h

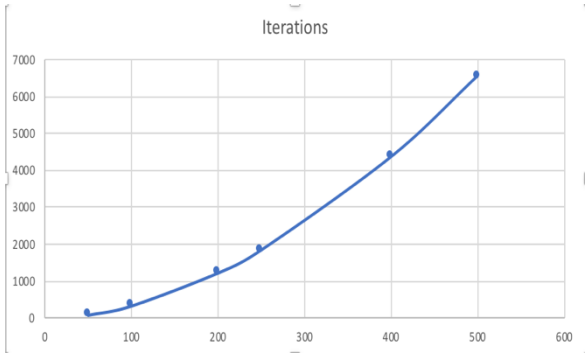


Figure 15:Jacobi Method- # of Iterations vs 1/h

e) h	1/h	Iterations	V(x,y)
0.02	50	88	40.5265
0.01	100	337	39.2382739
0.005	200	1226	38.7882251
0.004	250	1848	38.7150485
0.0025	400	4365	38.6173665
0.002	500	6545	38.5883926

Figure 16(Jacobi Method):; Change in Number of iterations and voltage measurments in response to decreasing h

It can be clearly seen from both the table and the figures how the Jacobi takes a lower number of iterations for the same spacing when compared to it's SOR counterpart. However, it can also be seen how the Jacobi method need a smaller spacing h for the result to converge as much as the SOR solution at a larger spacing. This agrees with the fundamental difference between the Jacobi and SOR methods, Considering how the SOR approach updates the weights gradually (Gauss-Seidel) and uses weights to influence its result.

.

e)

Appendix : Assignment 1 Code

```
##### BasicFunctions
from math import sqrt


def VerticalVector(HorizontalVector,dimension):
    n=dimension
    V=[0]*(n*n)
    for i in range (n*n):
        if ((i%n)==0): V[i]=HorizontalVector[i//n]

    VV=Matrix(V,n,1)

    return VV
#function to create a matrix
def Matrix(dataList,rows,columns ):
    Matrix = []
    counter=0
    for i in range(rows):
        rowList = []
        for j in range(columns):
            # you need to increment through dataList here, like this:
            rowList.append(dataList[counter])
            counter=counter+1
        Matrix.append(rowList)

    return Matrix
def SqMatrix(dataList,n):
    Matrix = []
    for i in range(n):
        rows = []
        for j in range(n):
            # you need to increment through dataList here, like this:
            rows.append(dataList[n * i + j])
        Matrix.append(rows)

    return Matrix
def rowdimension(s):
    return 1 + rowdimension(s[1:]) if s else 0
def columndimension(s):
    x=s[0]
    return  rowdimension(x)
#searches for largest value in square matrix of size n returns index of t
def TransposeSquare(rows,columns,M):
    MTinit=[0]*(n*n)
    MT=Matrix(MTinit,rows,columns)
    for i in range (rows):
        for j in range (columns):

                MT[i][j]=M[j][i]
    return MT
def Transpose(rows,columns,M):
```

```
    MTinit=[0]*(rows*columns)
    MT=Matrix(MTinit,columns,rows)
    for j in range (columns):
        for i in range (rows):

                MT[j][i]=M[i][j]
    return MT
def Multiplication (M1,M2):
    r1=rowdimension(M1)
    c1=columndimension(M1)

    r2=rowdimension(M2)
    c2=columndimension(M2)
    Productzeros=[0]*(r1*c2)
    Product=Matrix(Productzeros,r1,c2)
    if (c1!=r2):
        print("Matrix Multiplication is Invalid: Column count of first do
        return
    for i in range (r1):
        for j in range (c2):
            for k in range (c1):
                Product[i][j] = Product [i][j]+ M1[i][k]*M2[k][j]
    return Product
def MatrixVectorMultiplication(M1,V): #MV1=V2
    r1=rowdimension(M1)
    c1=columndimension(M1)
    r2=rowdimension(V)
    c2=1
    Productzeros=[0]*(c1*r2)
    Product=VerticalVector(Productzeros,r1*c2)
    if (c1!=r2):
        print("Matrix Multiplication is Invalid: Column count of first do
        return
    for i in range (r1):
            for k in range (c1):
                    Product[i][0] = Product [i][0]+ M1[i][k]*V[k]
    return Product
```

```
#CholskeyCode

def CholskeyL(A):

    #1st step: Check if A is square

    if(rowdimension(A)!=columndimension(A)):
        print ("matrix not square; Cholskey Decomp. Not Possible")
        return


    n=rowdimension(A)
    zeros=[0] * (n*n)
    L=Matrix(zeros,n,n)




    for j in range (n):
        for i in range (j,n) :
            if (i==j):

                #will have to add something here as j increases
                L[j][j]=sqrt(A[j][j])



            if(i!=j):
                L[i][j]=A[i][j]/L[j][j]
                for k in range (j+1,i+1):
                    A[i][k]=A[i][k]-L[i][j]*L[k][j]

            if(i==(j+1)):
                A[j][j]=A[j][j]-L[i][j]*L[i][j]




    return L
```

```
#Solver y from L and b > x from y and LT File:"SolverSeperate.py"

def Lyb(L,b):
    L=L
    b=b
    n=rowdimension(L)
    y=[0]*n
    yy=VerticalVector(y,n)


    for i in range (n):
        sum=0
        for j in range (i):
            sum=sum+L[i][j]*yy[j][0]


        yy[i][0]=(b[i][0]-sum)/L[i][i]



    return yy




def LTxY (L,yy):
    n=rowdimension(L)
    LT=Transpose(n,n,L)
    x=[0]*n
    xx=VerticalVector(x,n)

    for i in range (n):
        counter=n-i-1
        sum=0
        for k in range (i):
            sum=sum+LT[counter][counter+k+1]*xx[counter+k+1][0]

        xx[counter][0]=(yy[counter][0]-sum)/LT[counter][counter]


    return xx
```

```
#Q1 B
#Building Real, Symmetric, Posetive Definite  Matrices A2,A3,,,,A10
#File:"q1bmatrixbuild.py"

n=2
L2vector=[0]*(n*n)
L2=Matrix(L2vector,n,n)
for i in range (n) :
    for j in range (n):
        if i>=j:
            L2[i][j]=(i+1)+(j+1)
L2T=Transpose(n,n,L2)
A2=Multiplication(L2,L2T)

##########################################################################
##########################################################################
n=3
L3vector=[0]*(n*n)
L3=Matrix(L3vector,n,n)

for i in range (n) :
    for j in range (n):
        if i>=j:
            L3[i][j]=(i+1)+(j+1)
L3T=Transpose(n,n,L3)
A3=Multiplication(L3,L3T)
##########################################################################
##########################################################################
n=4
L4vector=[0]*(n*n)
L4=Matrix(L4vector,n,n)

for i in range (n) :
    for j in range (n):
        if i>=j:
            L4[i][j]=(i+1)+(j+1)
L4T=Transpose(n,n,L4)
A4=Multiplication(L4,L4T)
##########################################################################
##########################################################################
n=5
L5vector=[0]*(n*n)
L5=Matrix(L5vector,n,n)
for i in range (n) :
    for j in range (n):
        if i>=j:
            L5[i][j]=(i+1)+(j+1)
L5T=Transpose(n,n,L5)
A5=Multiplication(L5,L5T)
##########################################################################
##########################################################################
```

```
n=6
L6vector=[0]*(n*n)
L6=Matrix(L6vector,n,n)

for i in range (n) :
    for j in range (n):
        if i>=j:
            L6[i][j]=(i+1)+(j+1)
L6T=Transpose(n,n,L6)
A6=Multiplication(L6,L6T)
##########################################################################
##########################################################################
n=7
L7vector=[0]*(n*n)
L7=Matrix(L7vector,n,n)

for i in range (n) :
    for j in range (n):
        if i>=j:
            L7[i][j]=(i+1)+(j+1)
L7T=Transpose(n,n,L7)
A7=Multiplication(L7,L7T)
##########################################################################
##########################################################################
n=8
L8vector=[0]*(n*n)
L8=Matrix(L8vector,n,n)
for i in range (n) :
    for j in range (n):
        if i>=j:
            L8[i][j]=(i+1)+(j+1)
L8T=Transpose(n,n,L8)
A8=Multiplication(L8,L8T)
##########################################################################
##########################################################################
n=9
L9vector=[0]*(n*n)
L9=Matrix(L9vector,n,n)
for i in range (n) :
    for j in range (n):
        if i>=j:
            L9[i][j]=(i+1)+(j+1)
L9T=Transpose(n,n,L9)
A9=Multiplication(L9,L9T)
##########################################################################
##########################################################################
n=10
L10vector=[0]*(n*n)
L10=Matrix(L10vector,n,n)
for i in range (n) :
    for j in range (n):
```

```
        if i>=j:
            L10[i][j]=(i+1)+(j+1)
L10T=Transpose(n,n,L10)
A10=Multiplication(L10,L10T)
##########################################################################
##########################################################################
```

```
#Test 1 Cholskey decompostion: Getting L from A
#Filename:"q1btestmatricesCholskeytest.py"
from math import sqrt
##########################################################################
##########################################################################
n=2
L2error=0
L2hat=CholskeyL(A2)

for i in range (n) :
    for j in range (n):
        L2error=L2error+(L2hat[i][j]-L2[i][j])*(L2hat[i][j]-L2[i][j])
##########################################################################
##########################################################################
n=3
L3error=0
L3hat=CholskeyL(A3)

for i in range (n) :
    for j in range (n):
        L3error=L3error+(L3hat[i][j]-L3[i][j])*(L3hat[i][j]-L3[i][j])
##########################################################################
##########################################################################
n=4
L4error=0
L4hat=CholskeyL(A4)

for i in range (n) :
    for j in range (n):
        L4error=L4error+(L4hat[i][j]-L4[i][j])*(L4hat[i][j]-L4[i][j])
##########################################################################
##########################################################################
n=5
L5error=0
L5hat=CholskeyL(A5)

for i in range (n) :
    for j in range (n):
        L5error=L5error+(L5hat[i][j]-L5[i][j])*(L5hat[i][j]-L5[i][j])
##########################################################################
##########################################################################
n=6
L6error=0
L6hat=CholskeyL(A6)

for i in range (n) :
    for j in range (n):
        L6error=L6error+(L6hat[i][j]-L6[i][j])*(L6hat[i][j]-L6[i][j])
##########################################################################
##########################################################################
n=7
```

```
L7error=0
L7hat=CholskeyL(A7)


for i in range (n) :
    for j in range (n):
        L7error=L7error+(L7hat[i][j]-L7[i][j])*(L7hat[i][j]-L7[i][j])
##########################################################################
##########################################################################
n=8
L8error=0
L8hat=CholskeyL(A8)

for i in range (n) :
    for j in range (n):
        L8error=L8error+(L8hat[i][j]-L8[i][j])*(L8hat[i][j]-L8[i][j])
##########################################################################
##########################################################################
n=9
L9error=0
L9hat=CholskeyL(A9)

for i in range (n) :
    for j in range (n):
        L9error=L9error+(L9hat[i][j]-L9[i][j])*(L9hat[i][j]-L9[i][j])
##########################################################################
##########################################################################
n=10
L10error=0
L10hat=CholskeyL(A10)

for i in range (n) :
    for j in range (n):
        L10error=L10error+(L10hat[i][j]-L10[i][j])*(L10hat[i][j]-L10[i][j
```

```
#q2A solution
#computing the transpose of the reduced incidence Matrix
#File:"q2abResistance.py"
##########################################################################
##########################################################################

vterminal1= [0]*16
vterminal2= [0]*16#solution vector for different size

for i in range (2,16):

    n=i
    k=2*(n*n-n)+1
    A=Incidence(n)
    AT=Transpose(n*n-1,k,A)
    Jk=[0]*k
    Jk[k-1]=1 #dummy current source
    Jk=Matrix(Jk,k,1)

    lefthandside=Multiplication(A,AT)
    for z in range (n*n-1):
        for j in range (n*n-1):
            lefthandside[z][j]=lefthandside[z][j] / 10000

    righthandside=Multiplication(A,Jk)

    L=CholskeyL(lefthandside)
    y=Lyb(L,righthandside)
    x=LTxY (L,y)
    firstterminal=n-1
    secondterminal=n*n-n
    vterminal1[i]=x[firstterminal][0]




Sourcevoltagedifference= [0]*16
Rnet=[0] *16
for i in range (2,16):
    Sourcevoltagedifference[i]=vterminal1[i]
    Rnet[i]=10000*Sourcevoltagedifference[i]/(10000-Sourcevoltagedifferen
```

```
#File:"Q2A-Incidence.py"

n=4

def Incidence(N):

    n=N
    columns= (2*(n*n-n))
    columns=columns+1 #column added for dummy source current
    rows= n*n
    numberofelements=rows*columns
    Vector = [0] *( numberofelements )


    A=Matrix(Vector, rows, columns)

    i=0
    x=0
    currentcounter=0
    counteri=0
    for i in range (0,n*(n-1)):

        if ((i)%(n-1)==0 and i !=0 ):
            x=x+1



        count=0
        alpha=i+x
        beta=i+x+2
        for j in range (alpha,beta):
            count=count+1
            if (count%2==0):
                A[j][i]=-1
            else: A[j][i]=+1


        counteri=counteri+1



    for i in range (0,n*(n-1)):

        j1= i
        j2= i+n

        A[j1][counteri]=+1
        A[j2][counteri]=-1
        counteri=counteri+1
```

```
    import numpy as np


    #Source Current Values

    A[n-1][columns-1]=+1
    A[n*n-n][columns-1]=-1
    A0=np.array(A)



#remove the last line to obtain the reduced incidence matrix / the remove
#is that of one of the terminals of which we want measure the resistance
    Ar=[0]* ((rows-1)*columns)
    Ar=Matrix(Ar,rows-1,columns)
    for i in range (n*n-n):
        for j in range (columns):
            Ar[i][j]=A[i][j]

    for i in range (n*n-n+1,rows):
        for j in range (columns):
            Ar[i-1][j]=A[i][j]



    return Ar
```

```
#halfbandwidth computation for each matrix
#File:"q2chalfbandwidth"
n=4
def halfbandwidth(L):
    L=L

    n=rowdimension(L)
    y=[0]*n
    yy=VerticalVector(y,n)
    hbandwidth=[0]*n
    b=[0]*n

    for j in range (n):

        for i in range (n):
            if (L[n-i-1][j]==0):
                hbandwidth[j]=hbandwidth[j]+1
            else:
                break




    for i in range (n):
        b[i]=n-i-hbandwidth[i]


    return  b



k=2*(n*n-n)+1
A=Incidence(n)
AT=Transpose(n*n-1,k,A)
Jk=[0]*k
Jk[k-1]=1 #dummy current source
Jk=Matrix(Jk,k,1)

lefthandside=Multiplication(A,AT)
for z in range (n*n-1):
    for j in range (n*n-1):
        lefthandside[z][j]=lefthandside[z][j] / 10000

righthandside=Multiplication(A,Jk)

L=CholskeyL(lefthandside)

halfbandwidth=halfbandwidth(L)
halfbandwidth
```

```
SummaryHalfband=[0]*16

SummaryHalfband[2]=2
SummaryHalfband[3]=4
SummaryHalfband[4]=5
SummaryHalfband[5]=6
SummaryHalfband[6]=7
SummaryHalfband[7]=8
SummaryHalfband[8]=9
SummaryHalfband[9]=10
SummaryHalfband[10]=11
SummaryHalfband[11]=12
SummaryHalfband[12]=13
SummaryHalfband[13]=14
SummaryHalfband[14]=15
SummaryHalfband[15]=16
```

```
#lookahead ra8am 3
#File:"q2cSparsitysolversep.py"
from math import sqrt
def Lybsparse(L,b):
    L=L
    b=b
    n=rowdimension(L)
    y=[0]*n
    yy=VerticalVector(y,n)

    halfbandwidth=int(sqrt(n+1))+1
    z=0
    for i in range (n):
        sum=0
        if (i==halfbandwidth):
            z=z+1

        for j in range (z,i):
            sum=sum+L[i][j]*yy[j][0]


        yy[i][0]=(b[i][0]-sum)/L[i][i]




    return yy


Lybsparse(L,righthandside)
Lyb(L,righthandside)

def LTxYSparse (L,yy):
    n=rowdimension(L)
    LT=Transpose(n,n,L)
    x=[0]*n
    xx=VerticalVector(x,n)
    halfbandwidth=int(sqrt(n+1))+1
    z=0
    for i in range (n):
        counter=n-i-1
        sum=0
        if (i==halfbandwidth):
            z=z+1
        for k in range (i-z):
            sum=sum+LT[counter][counter+k+1]*xx[counter+k+1][0]

        xx[counter][0]=(yy[counter][0]-sum)/LT[counter][counter]
```

```
    return xx
```

```
#File:"q2cSparsityExcution
vterminal1= [0]*16
vterminal2= [0]*16#solution vector for different size

for i in range (2,16):

    n=i
    k=2*(n*n-n)+1
    A=Incidence(n)
    AT=Transpose(n*n-1,k,A)
    Jk=[0]*k
    Jk[k-1]=1 #dummy current source
    Jk=Matrix(Jk,k,1)

    lefthandside=Multiplication(A,AT)
    for z in range (n*n-1):
        for j in range (n*n-1):
            lefthandside[z][j]=lefthandside[z][j] / 10000

    righthandside=Multiplication(A,Jk)

    L=CholskeyL(lefthandside)
    y=Lybsparse(L,righthandside)
    x=LTxYSparse (L,y)
    firstterminal=n-1
    secondterminal=n*n-n
    vterminal1[i]=x[firstterminal][0]




Sourcevoltagedifference= [0]*16
RnetSparse=[0] *16
for i in range (2,16):
    Sourcevoltagedifference[i]=vterminal1[i]
    RnetSparse[i]=10000*Sourcevoltagedifference[i]/(10000-Sourcevoltagedi
```

```
#Q1C part1 invent an x , produce a b
#each X is a vector of dimension nx1
#File:"q1cinventxproduceb.py"

#########################################################################
#########################################################################
n=2
z=[0]*n
x2=Matrix(z,n,1)

for i in range (n):
    x2[i]=i^2+10*i+2
b2=MatrixVectorMultiplication(A2,x2)

#########################################################################
#########################################################################
n=3
z=[0]*n
x3=Matrix(z,n,1)

for i in range (n):
    x3[i]=i^2+10*i+2
b3=MatrixVectorMultiplication(A3,x3)
#########################################################################
#########################################################################
n=4
z=[0]*n
x4=Matrix(z,n,1)
for i in range (n):
    x4[i]=i^2+10*i+2
b4=MatrixVectorMultiplication(A4,x4)

#########################################################################
#########################################################################
n=5
z=[0]*n
x5=Matrix(z,n,1)
for i in range (n):
    x5[i]=i^2+10*i+2
b5=MatrixVectorMultiplication(A5,x5)

#########################################################################
#########################################################################
n=6
z=[0]*n
x6=Matrix(z,n,1)
for i in range (n):
    x6[i]=i^2+10*i+2
b6=MatrixVectorMultiplication(A6,x6)

#########################################################################
```

```
#########################################################################
n=7
z=[0]*n
x7=Matrix(z,n,1)
for i in range (n):
    x7[i]=i^2+10*i+2
b7=MatrixVectorMultiplication(A7,x7)

#########################################################################
#########################################################################
n=8
z=[0]*n
x8=Matrix(z,n,1)
for i in range (n):
    x8[i]=i^2+10*i+2
b8=MatrixVectorMultiplication(A8,x8)

#########################################################################
#########################################################################
n=9
z=[0]*n
x9=Matrix(z,n,1)
for i in range (n):
    x9[i]=i^2+10*i+2
b9=MatrixVectorMultiplication(A9,x9)


#########################################################################
#########################################################################
n=10
z=[0]*n
x10=Matrix(z,n,1)
for i in range (n):
    x10[i]=i^2+10*i+2
b10=MatrixVectorMultiplication(A10,x10)
```

```
#Test 2 Getting x from A b (output is Xnhat: Input An and bn matrices)
#File:"q1cproducexhat.py"

#Lhat was obtained in an earlier segment of the code and so is being used
##########################################################################
##########################################################################
n=2
L=L2hat
b=b2
y=Lyb(L,b)
X2hat=LTxY (L,y)
X2error=0
for i in range (n):
    X2error=X2error+ (x2[i]-X2hat[i][0])*(x2[i]-X2hat[i][0])
##########################################################################
##########################################################################
n=3
L=L3hat
b=b3
y=Lyb(L,b)
X3hat=LTxY (L,y)
X3error=0
for i in range (n):
    X3error=X3error+ (x3[i]-X3hat[i][0])*(x3[i]-X3hat[i][0])
##########################################################################
##########################################################################
n=4
L=L4hat
b=b4
y=Lyb(L,b)
X4hat=LTxY (L,y)
X4error=0
for i in range (n):
    X4error=X4error+ (x4[i]-X4hat[i][0])*(x4[i]-X4hat[i][0])
##########################################################################
##########################################################################
n=5
L=L5hat
b=b5
y=Lyb(L,b)
X5hat=LTxY (L,y)
X5error=0
for i in range (n):
    X5error=X5error+ (x5[i]-X5hat[i][0])*(x5[i]-X5hat[i][0])
##########################################################################
##########################################################################
n=6
L=L6hat
b=b6
y=Lyb(L,b)
X6hat=LTxY (L,y)
```

```
X6error=0
for i in range (n):
    X6error=X6error+ (x6[i]-X6hat[i][0])*(x6[i]-X6hat[i][0])
##########################################################################
##########################################################################
n=7
L=L7hat
b=b7
y=Lyb(L,b)
X7hat=LTxY (L,y)
X7error=0
for i in range (n):
    X7error=X7error+ (x7[i]-X7hat[i][0])*(x7[i]-X7hat[i][0])
##########################################################################
##########################################################################
n=8
L=L8hat
b=b8
y=Lyb(L,b)
X8hat=LTxY (L,y)
X8error=0
for i in range (n):
    X8error=X8error+ (x8[i]-X8hat[i][0])*(x8[i]-X8hat[i][0])
##########################################################################
##########################################################################
n=9
L=L9hat
b=b9
y=Lyb(L,b)
X9hat=LTxY (L,y)
X9error=0
for i in range (n):
    X9error=X9error+ (x9[i]-X9hat[i][0])*(x9[i]-X9hat[i][0])
##########################################################################
##########################################################################
n=10
L=L10hat
b=b10
y=Lyb(L,b)
X10hat=LTxY (L,y)
X10error=0
for i in range (n):
    X10error=X10error+ (x10[i]-X10hat[i][0])*(x10[i]-X10hat[i][0])
##########################################################################
##########################################################################
```

```
#q1d create input file / read input file
#File:"q1dcreateinputfilereadinputfile.py"

def SaveNetworkMatricesAsText(networkname,JK,RK,EK,A,n,k):

    name= networkname
    f= open(name,"w+") #creates a network file

    f.write("Network Name: %s \r\n JK,RK,EK and reduced A Matrix Paramete
    f.write("number of nodes:\r\n")
    f.write("%d;\r\n" %(n) )
    f.write("number of branches\r\n")
    f.write("%d;\r\n" %(k) )#write parameters of the 4 matrices to be use

    #first matrix saved is A; comma seperated; 1 line

    #2nd matrix JK
    f.write("Jk Rk Ek\r\n" )
    for i in range(k):
        a=JK[i][0]
        b=RK[i][0]
        c=EK[i][0]
        f.write("%f,%f,%f;\r\n" % (a,b,c))



    f.write("A\r\n" )
    for i in range(n-1):
        for j in range (k):
            if j!=(k-1):
                f.write("%f," % A[i][j])
            else:
                f.write("%f" % A[i][j])


        f.write(";\r\n" )



    f.write("E" )

def ReadMatrices(string):

    f= open(string) #creates a network file

    lines=f.readlines()
```

```
i=0
while lines[3][i]!=';':
    i=i+1

n=int(lines[3][0:i])

i=0
while lines[5][i] !=';':
    i=i+1

k=int(lines[5][0:i])


Jk=[0]*k
Ek=[0]*k
Rk=[0]*k
A=[0]*(k*(n-1))




for i in range (k):
    j=0
    #writes down Jk elements
    counter1=0
    while lines[7+i][j]!=',':
        counter1=counter1+1
        j=j+1
    sumcounter=counter1
    Jk[i]=float(lines[7+i][sumcounter-counter1+1*0:sumcounter])
    j=j+1



    #writes down Rk elements
    counter2=0
    while lines[7+i][j]!=',':
        counter2=counter2+1
        j=j+1


    sumcounter=sumcounter+counter2
    Rk[i]=float(lines[7+i][sumcounter-counter2+1*1:sumcounter])
    j=j+1


    ##writes down Ek elements
    counter3=0
```

```
    while lines[7+i][j]!=';' :
        counter3=counter3+1
        j=j+1
    sumcounter=sumcounter+counter3
    Ek[i]=float(lines[7+i][sumcounter-counter3+1*2:sumcounter]   )



#A will start from 7+k+1
#have to account for negative elements in A
element=0
for i in range (k+8,n+k+8):

    if (lines[i][0]=='E'):
        break

    sumcounter=0
    counter=[0]*k
    j=0
    for z in range (k):

        while (lines[i][j]!=',' ):
            counter[z]=counter[z]+1
            j=j+1
            if (lines[i][j]==';'):
                break

        j=j+1
        sumcounter=sumcounter+counter[z]



        A[element]=float(lines[i][sumcounter-counter[z]+1*z:sumcounte

        element=element+1


        if (element==k):
            break



A=Matrix(A,n-1,k)
JK=Matrix(Jk,k,1)
EK=Matrix(Ek,k,1)
RK=Matrix(Rk,k,1)

return {'A':A, 'JK':JK ,'RK':RK,'EK':EK, 'n':n, 'k':k }
```

```
#creating the input files for the circuits
#File:"q1dcreateinputfilesfortestcircuits"
##########################################################################
##########################################################################
#circuit 1
n=2
k=2
A=[-1, -1]
Rk=[1/10, 1/10]
Ek=[10,0]
Jk=[0,0]
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
A=Matrix(A,n-1,k)
for i in range (k):
        Y[i][i]=Rk[i]
Ek=Matrix(Ek,k,1)
Jk=Matrix(Jk,k,1)
Rk=Matrix(Rk,k,1)
Networkname='ckt1'
SaveNetworkMatricesAsText(Networkname,Jk,Rk,Ek,A,n,k)
##########################################################################
##########################################################################
#circuit 2
n=2
k=2
A=[1, 1]
Rk=[0.1, 0.1]
Ek=[0,0]
Jk=[10,0]
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
A=Matrix(A,n-1,k)
for i in range (k):
        Y[i][i]=Rk[i]
Ek=Matrix(Ek,k,1)
Jk=Matrix(Jk,k,1)
Rk=Matrix(Rk,k,1)
Networkname='ckt2'
SaveNetworkMatricesAsText(Networkname,Jk,Rk,Ek,A,n,k)
##########################################################################
##########################################################################
#circuit 3
n=2
k=2
A=[-1, 1]
Rk=[1/10, 1/10]
Ek=[10,0]
Jk=[0,10]
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
```

```
A=Matrix(A,n-1,k)
for i in range (k):
        Y[i][i]=Rk[i]
Ek=Matrix(Ek,k,1)
Jk=Matrix(Jk,k,1)
Rk=Matrix(Rk,k,1)
Networkname='ckt3'
SaveNetworkMatricesAsText(Networkname,Jk,Rk,Ek,A,n,k)
##########################################################################
##########################################################################
#circuit 4
n=3
k=4
A=[-1, 1,1,0,0,0,-1,1]
Rk=[1/10, 1/10,1/5,1/5]
Ek=[10,0,0,0]
Jk=[0,0,0,10]
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
A=Matrix(A,n-1,k)
for i in range (k):
        Y[i][i]=Rk[i]
Ek=Matrix(Ek,k,1)
Jk=Matrix(Jk,k,1)
Rk=Matrix(Rk,k,1)
Networkname='ckt4'
SaveNetworkMatricesAsText(Networkname,Jk,Rk,Ek,A,n,k)
##########################################################################
##########################################################################
#circuit 5
n=4
k=6
A=[-1, 1,1,0,0,0,0,-1,0,1,1,0,0,0,-1,-1,0,1]
Rk=[1/20, 1/10,1/10,1/30,1/30,1/30]
Ek=[10,0,0,0,0,0]
Jk=[0,0,0,0,0,0]
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
A=Matrix(A,n-1,k)
for i in range (k):
        Y[i][i]=Rk[i]
Ek=Matrix(Ek,k,1)
Jk=Matrix(Jk,k,1)
Rk=Matrix(Rk,k,1)
Networkname='ckt5'
SaveNetworkMatricesAsText(Networkname,Jk,Rk,Ek,A,n,k)
##########################################################################
##########################################################################
```

```
#extract matrices from input files ckt1 to ckt5 / Solution
#File:"q1dExtractMatricesSolutions.py"
##########################################################################
##########################################################################

z=ReadMatrices('ckt1')

A=z['A'] #A[0][j]
Ek=z['EK']
Rk=z['RK']
Jk=z['JK']
n=z['n']
k=z['k']
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
for i in range (k):
        Y[i][i]=Rk[i][0]
AT=Transpose(n-1,k,A)
lefthandside=Multiplication(A,Y)
lefthandside=Multiplication(lefthandside,AT)
righthandside= Multiplication(Y,Ek)
for i in range (k):
    righthandside[i][0]=Jk[i][0]-righthandside[i][0]
b=Multiplication(A,righthandside)
righthandside=b




L=CholskeyL(lefthandside)
y=Lyb(L,righthandside)
X=LTxY (L,y)

##########################################################################
##########################################################################
z=ReadMatrices('ckt2')

A=z['A'] #A[0][j]
Ek=z['EK']
Rk=z['RK']
Jk=z['JK']
n=z['n']
k=z['k']
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
for i in range (k):
        Y[i][i]=Rk[i][0]
AT=Transpose(n-1,k,A)
lefthandside=Multiplication(A,Y)
lefthandside=Multiplication(lefthandside,AT)
```

```
righthandside= Multiplication(Y,Ek)

for i in range (k):
    righthandside[i][0]=Jk[i][0]-righthandside[i][0]
b=Multiplication(A,righthandside)
righthandside=b




L=CholskeyL(lefthandside)
y=Lyb(L,righthandside)
X=LTxY (L,y)


##########################################################################
##########################################################################
z=ReadMatrices('ckt3')

A=z['A'] #A[0][j]
Ek=z['EK']
Rk=z['RK']
Jk=z['JK']
n=z['n']
k=z['k']
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
for i in range (k):
        Y[i][i]=Rk[i][0]
AT=Transpose(n-1,k,A)
lefthandside=Multiplication(A,Y)
lefthandside=Multiplication(lefthandside,AT)
righthandside= Multiplication(Y,Ek)
for i in range (k):
    righthandside[i][0]=Jk[i][0]-righthandside[i][0]
b=Multiplication(A,righthandside)
righthandside=b




L=CholskeyL(lefthandside)
y=Lyb(L,righthandside)
X=LTxY (L,y)

##########################################################################
##########################################################################
z=ReadMatrices('ckt4')
```

```
A=z['A'] #A[0][j]
Ek=z['EK']
Rk=z['RK']
Jk=z['JK']
n=z['n']
k=z['k']
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
for i in range (k):
        Y[i][i]=Rk[i][0]
AT=Transpose(n-1,k,A)
lefthandside=Multiplication(A,Y)
lefthandside=Multiplication(lefthandside,AT)
righthandside= Multiplication(Y,Ek)
for i in range (k):
    righthandside[i][0]=Jk[i][0]-righthandside[i][0]
b=Multiplication(A,righthandside)
righthandside=b




L=CholskeyL(lefthandside)
y=Lyb(L,righthandside)
X=LTxY (L,y)
################################################################################
################################################################################
z=ReadMatrices('ckt5')

A=z['A'] #A[0][j]
Ek=z['EK']
Rk=z['RK']
Jk=z['JK']
n=z['n']
k=z['k']
Y=[0]*(k*k)
Y=Matrix(Y,k,k)
for i in range (k):
        Y[i][i]=Rk[i][0]
AT=Transpose(n-1,k,A)
lefthandside=Multiplication(A,Y)
lefthandside=Multiplication(lefthandside,AT)
righthandside= Multiplication(Y,Ek)
for i in range (k):
    righthandside[i][0]=Jk[i][0]-righthandside[i][0]
b=Multiplication(A,righthandside)
righthandside=b
```

```
L=CholskeyL(lefthandside)
y=Lyb(L,righthandside)
X=LTxY (L,y)
```

```
#new gauss

#q3 test



def Gauss(Vk1,deltax,deltay,alpha,beta, bottomleftedgeXnode,bottomleftedg
    for i in range (1,bottomleftedgeXnode):
        for j in range (1,bottomleftedgeYnode):
            alpha1=alpha[i-1]
            alpha2=alpha[i]
            beta1=beta[j-1]
            beta2=beta[j]


        #first term
            term1=deltax*deltax
            term1=term1*deltay*deltay
            term1=term1*alpha1*alpha2*beta1*beta2
            term1=term1/2
            term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


        #second term
            term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
            term2=2/term2

        #third term
            term3=deltay*deltay*beta1*beta2*(beta1+beta2)
            term3=2/term3




        #change2 to 1 (mn el jehatein entabih)
            Vk1[i][j]= (term2 * alpha2 * Vk1[i-1][j])
            Vk1[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i
            Vk1[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
            Vk1[i][j]= (term3*beta1 * Vk1[i][j+1]) +              Vk1[i
            Vk1[i][j]= term1 *             Vk1[i][j]

        #vk1[i][j-1] vk1[i-1][j] will have been computed for k+1 to be us

        for j in range (bottomleftedgeYnode,nodesxdirection-1):

            alpha1=alpha[i-1]
            alpha2=alpha[i]
            beta1=beta[j-1]
```

```
    beta2=beta[j]


#first term
    term1=deltax*deltax
    term1=term1*deltay*deltay
    term1=term1*alpha1*alpha2*beta1*beta2
    term1=term1/2
    term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


#second term
    term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
    term2=2/term2

#third term
    term3=deltay*deltay*beta1*beta2*(beta1+beta2)
    term3=2/term3




#change2 to 1 (mn el jehatein entabih)
    Vk1[i][j]= (term2 * alpha2 * Vk1[i-1][j])
    Vk1[i][j]=  (term2 * alpha1 * Vk1[i+1][j] )          + Vk1[i]
    Vk1[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
    Vk1[i][j]= (term3*beta1 * Vk1[i][j+1]) +                Vk1[i
    Vk1[i][j]= term1 *             Vk1[i][j]



#boundary/beta1 term is removed
j=nodesydirection-1
alpha1=alpha[i-1]
alpha2=alpha[i]

beta1=beta[j-1]
beta2=beta1



#first term
term1=deltax*deltax
term1=term1*deltay*deltay
term1=term1*alpha1*alpha2*beta1*beta2
term1=term1/2
term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*delta
```

```
        #second term
        term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
        term2=2/term2

        #third term
        term3=deltay*deltay*beta1*beta2*(beta1+beta2)
        term3=2/term3



        #vk[i][j+1] replaced with vk[i][j-1] and beta2 is used instead of
        #these parameters are equal due to mirror symmetry
        #change2 to 1 (mn el jehatein entabih)
        Vk1[i][j]= (term2 * alpha2 * Vk1[i-1][j])
        Vk1[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i][j]
        Vk1[i][j]= (term3 * beta1 * Vk1[i][j-1]) +              Vk1[i][j
        Vk1[i][j]= (term3*beta1 * Vk1[i][j-1]) +               Vk1[i][j]
        Vk1[i][j]= term1 *             Vk1[i][j]




for i in range (bottomleftedgeXnode,nodesxdirection-1):
    for j in range (1,bottomleftedgeYnode):
        alpha1=alpha[i-1]
        alpha2=alpha[i]
        beta1=beta[j-1]
        beta2=beta[j]


        #first term
        term1=deltax*deltax
        term1=term1*deltay*deltay
        term1=term1*alpha1*alpha2*beta1*beta2
        term1=term1/2
        term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


        #second term
        term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
        term2=2/term2

        #third term
        term3=deltay*deltay*beta1*beta2*(beta1+beta2)
        term3=2/term3
```

```
        #change2 to 1 (mn el jehatein entabih)
            Vk1[i][j]= (term2 * alpha2 * Vk1[i-1][j])
            Vk1[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i
            Vk1[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
            Vk1[i][j]= (term3*beta1 * Vk1[i][j+1]) +               Vk1[i
            Vk1[i][j]= term1 *             Vk1[i][j]



    for j in range (1,bottomleftedgeYnode):
        i=nodesxdirection-1
        #alpha1 removed
        alpha1=alpha[i-1]
        alpha2=alpha1
        beta1=beta[j-1]
        beta2=beta[j]
        #first term
        term1=deltax*deltax
        term1=term1*deltay*deltay
        term1=term1*alpha1*alpha2*beta1*beta2
        term1=term1/2
        term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*delta
        #second term
        term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
        term2=2/term2
        #third term
        term3=deltay*deltay*beta1*beta2*(beta1+beta2)
        term3=2/term3



        #vk[i+1][j] and alpha2 replaced with vk[i-1][j] and alpha1 since
        #change2 to 1 mn el jehatein entabih
        Vk1[i][j]= (term2 * alpha1 * Vk1[i-1][j])
        Vk1[i][j]=  (term2 * alpha1 * Vk1[i-1][j] )          + Vk1[i][j]
        Vk1[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[i][j
        Vk1[i][j]= (term3*beta1 * Vk1[i][j+1]) +               Vk1[i][j]
        Vk1[i][j]= term1 *             Vk1[i][j]


    return Vk1

def Residual(Vk2,bottomleftedgeXnode,bottomleftedgeYnode,nodesxdirection,

    max=0

    for i in range (1,nodesxdirection-1):
        for j in range (1,nodesydirection-1):
            z= Vk2[i+1][j]
            z= Vk2[i-1][j] + z
```

```
        z= Vk2[i][j+1]+ z
        z= Vk2[i][j-1]+ z
        z= z - 4 * Vk2[i][j]
        R[i][j]=z
        # to be modified
# I get the effect of the internal conductor then I zero it's terms i


        #boundary nodes /making use of symmetry


j=nodesydirection-1
for i in range (1,bottomleftedgeXnode):
    z=Vk2[i-1][j]
    z=z+Vk2[i+1][j]
    z=z+Vk2[i][j-1]
    z=z+Vk2[i][j-1]
    z=z-4 * Vk2[i][j]

    R[i][j]=z
i=nodesxdirection-1
for j in range (1,bottomleftedgeYnode):
    z=Vk2[i-1][j]
    z= Vk2[i-1][j] + z
    z=Vk2[i][j-1]+ z
    z=Vk2[i][j+1]+ z
    z=z-4 * Vk2[i][j]
    R[i][j]=z



for i in range (bottomleftedgeXnode,nodesxdirection):
    for j in range (bottomleftedgeYnode,nodesydirection):
        R[i][j]=0

i=0
for j in range (nodesydirection):
    R[i][j]=0

j=0
for i in range (nodesxdirection):
    R[i][j]=0



for i in range (nodesxdirection):
    for j in range (nodesydirection):
        if R[i][j]>max:
            max=R[i][j]
```

```
    return max

def SORaddition(Vk0,Vk1,w,nodesxdirection,nodesydirection):
    for i in range (nodesxdirection):
        for j in range (nodesydirection):
            Vk1[i][j]=(1-w)*Vk0[i][j]+(w)*Vk1[i][j]

    return Vk1


def Residualnew(Vk2,alpha,beta,deltax,deltay, bottomleftedgeXnode,bottoml

    max=0

    #designed s.t. deltax=delta y for consistency of results

    for i in range (1,nodesxdirection-1):
        for j in range (1,nodesydirection-1):
            alpha1=alpha[i-1]
            alpha2=alpha[i]
            beta1=beta[j-1]
            beta2=beta[j]

            x=Vk2[i][j]

            z1= (Vk2[i+1][j]-x)/(alpha2*(alpha1+alpha2))
            z1= (Vk2[i-1][j]-x)/(alpha1*(alpha1+alpha2)) + z1
            z1=z1*2
            z2= (Vk2[i][j+1]-x)/(beta2*(beta1+beta2))
            z2= (Vk2[i][j-1]-x)/(beta1*(beta1+beta2))+ z2
            z2=  z2*2
            R[i][j]=z1+z2

    # I get the effect of the internal conductor then I zero it's terms i


            #boundary nodes /making use of symmetry


    j=nodesydirection-1
    for i in range (1,bottomleftedgeXnode):

        alpha1=alpha[i-1]
        alpha2=alpha[i]
        beta1=beta[j-1]
        beta2=beta1


        x=Vk2[i][j]
```

```
    z1=(Vk2[i-1][j]-x)/(alpha1*(alpha1+alpha2))
    z1=(Vk2[i+1][j]-x)/(alpha2*(alpha1+alpha2)) +z1

    z1=z1*2

    z2=(Vk2[i][j-1]-x)/(beta1*(beta1+beta2))
    z2=(Vk2[i][j-1]-x)/(beta1*(beta1+beta2))+z2

    z2=  z2*2


    z=z1+z2

    R[i][j]=z




i=nodesxdirection-1
for j in range (1,bottomleftedgeYnode):


    alpha1=alpha[i-1]
    alpha2=alpha1
    beta1=beta[j-1]
    beta2=beta[j]

    x=Vk2[i][j]

    z1=(Vk2[i-1][j]-x)/(alpha1*(alpha1+alpha2))
    z1= (Vk2[i-1][j]-x)/(alpha1*(alpha1+alpha2)) + z1


    z1=z1*2

    z2=(Vk2[i][j-1]-x) /(beta1*(beta1+beta2))
    z2=(Vk2[i][j+1]-x)/(beta2*(beta1+beta2)) + z2

    z2=  z2*2

    z=z1+z2


    R[i][j]=z



    #la hooooooon
```

```
for i in range (bottomleftedgeXnode,nodesxdirection):
    for j in range (bottomleftedgeYnode,nodesydirection):
        R[i][j]=0

i=0
for j in range (nodesydirection):
    R[i][j]=0

j=0
for i in range (nodesxdirection):
    R[i][j]=0



for i in range (nodesxdirection):
    for j in range (nodesydirection):
        if R[i][j]>max:
            max=R[i][j]




return max
```

```
#SOR code



def SOR(point,T,w,deltax,deltay, equalspacing, xydimension,Vinnerouter,ce


    #inner conudctor location
    center=[xydimension[0]/2,xydimension[1]/2]
    #deltaxanddeltay
    if equalspacing==1:
        deltax=h
        deltay=h


    edgex=(xydimension[0]/2)-centercond[1]/2



    edgey=(xydimension[1]/2-centercond[0]/2)



    point[0]


    point[1]


    pointcoordinateXcheck=0
    pointcoordinateYcheck=0



    #check validity of alpha/beta values

    if equalspacing!=1:
        checkx=0
        checky=0
        innerconductoredgextest=0 #inclusion of inner conductor edge x
        innerconductoredgeytest=0 #inclusion of inner conductor edge y
        nodesxdirection = rowdimension(alpha) +1
        nodesydirection = rowdimension(beta) +1
```

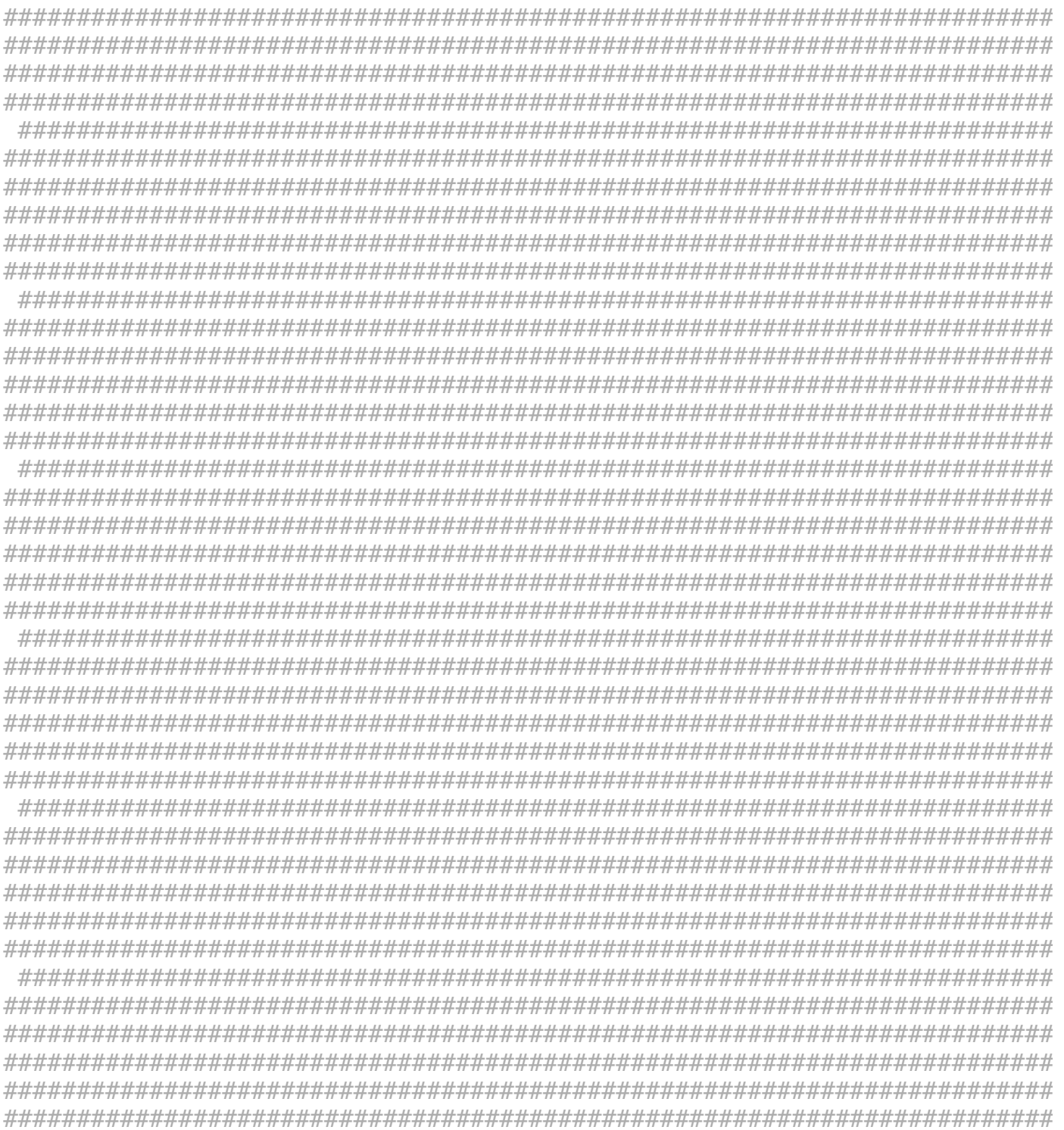
```
n=nodesxdirection* nodesydirection #total number of nodes
n=int(n)



for i in range (nodesxdirection-1):


    checkx=checkx+alpha[i]*deltax

    #checks if edges of inner conductor are accounted for by alph
    if (round(checkx, 10)==round(edgex, 10)):
        innerconductoredgextest=int(1)
        bottomleftedgeXnode=i+1


     #checks if edges of desired point x coordinate is located at
    if (round(checkx, 10)==round(point[0], 10)):
        pointcoordinateXcheck=int(1)
        outputnodex=i+1







    #to check whether alpha,beta,deltax,deltay cover internal con
    #which are essential for a correct solution for the problem

for i in range (nodesydirection-1):



    checky=checky+beta[i]*deltay

    #checks if edges of inner conductor are accounted for by alph


    if (round(checky, 10)==round(edgey, 10)):
        innerconductoredgeytest=int(1)
        bottomleftedgeYnode=i+1


    #checks if edges of desired point y coordinate is located at
    if (round(checky, 10)==round(point[1], 10)):
        pointcoordinateYcheck=int(1)
        outputnodey=j+1



#to check compatibility of alpha, beta, deltax deltay with proble
```

```
    if (round(checkx,10)!= (round(xydimension[0]/2,10))):
        print("alpha parameters and deltax invalid for problem geomet
        return
    if (round(checky,10)!= round((xydimension [1]/2),10)):
        print("beta parameters and deltay invalid for problem geometr
        return

    if (innerconductoredgextest==0):
        print("alpha parameters and deltax invalid for inclusion of i
        return

    if (innerconductoredgeytest==0):
        print("beta parameters and deltay invalid for inclusion of in
        return



#numberofnodes/constant(for both equal and unequal)because the assign
if equalspacing==1:
    nodesxdirection = int((xydimension[0]/(2*deltax) + 1)) #perline
    nodesydirection = int(xydimension[1]/(2*deltay)+1) #perline
    n=nodesxdirection* nodesydirection #total number of nodes
    n=int(n)
    numberofalphas=nodesxdirection-1
    numberofbetas=nodesydirection-1
    alpha=[1]*(numberofalphas)
    beta=[1]*(numberofbetas)
    bottomleftedgeXnode=edgex/deltax
    bottomleftedgeYnode=edgey/deltay
    outpointx=int(point[0]/h)
    outpointy=int(point[1]/h)

    if int(point[0]/h)!=outpointx:
        print ("h not appropriate for including output point x coordi
        return

    if int(point[1]/h)!=outpointy:
        print ("h not appropriate for including output point y coordi
        return


    if int(bottomleftedgeXnode)!=bottomleftedgeXnode:
        print ("h not appropriate for assigning nodes to edges. choos
        return
```

```
    bottomleftedgeXnode=int(bottomleftedgeXnode)


    if int(bottomleftedgeYnode)!=bottomleftedgeYnode:
        print ("h not appropriate for assigning nodes to edges. choos
        return

    if h>xydimension[0] or h>xydimension[1]:
        print ("h not appropriate for problem geometry")
        return

    bottomleftedgeYnode=int(bottomleftedgeYnode)




#define nodes


voltagevector=[0]*(n)   #symmetry exploited here
Voltage0=Matrix(voltagevector,nodesxdirection,nodesydirection)      #
Voltage1=Matrix(voltagevector,nodesxdirection,nodesydirection)
Voltage2=Matrix(voltagevector,nodesxdirection,nodesydirection)
R=Matrix(voltagevector,nodesxdirection,nodesydirection)

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


    #lhon

#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#flag for inappropriate deltax/deltay/h choice : problem geometry spe




#########################################################################
#########################################################################
```

```
#########################################################################
#########################################################################
#########################################################################
#########################################################################

#Setting initial voltage values / problem geometry specific

for i in range (nodesxdirection):
    Voltage0 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage0 [i][0]= Vinnerouter[1]   #left 0v conductor / symmetry e
    Voltage1 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage1 [i][0]= Vinnerouter[1]
    Voltage2 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage2 [i][0]= Vinnerouter[1]



#center conductor voltages
for i in range (bottomleftedgeXnode,nodesxdirection):
    for j in range (bottomleftedgeYnode,nodesydirection):
        Voltage0 [i][j]=Vinnerouter[0]
        Voltage1 [i][j]=Vinnerouter[0]
        Voltage2 [i][j]=Vinnerouter[0]


#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#numerical solution

#########################################################################
#step1 make a guess
#let the guess be zeros which is already there


#for statement begins here
#########################################################################
#compute gaus seidel method values at iteration k+1
#make a copy of voltage vector
Vk0=Voltage0 #k
Vk1=Voltage1 # after gauss
Vk2=Voltage2 #k+1 after SOR
#equationterms:
#terms matrices in terms of alpha and beta

outputpoint=[outpointx,outpointy]

#7ot equation gauss hon
```

```
    true=1
    i=0
    #newgaussfile
    counter=0
    while (true):

        for i in range (nodesxdirection):
            for j in range (nodesydirection):
                Vk0[i][j]=Vk2[i][j]

        #Vk0 is the old value
        #Vk1 is the value after gauss
        #Vk2 is the value after the Relaxation

        #the relaxation takes the old(VK0) and the Gauss output(Vk1) as i



        Vk1=Gauss(Vk2,deltax,deltay,alpha,beta, bottomleftedgeXnode,botto
        Vk2=SORaddition(Vk0,Vk1,w,nodesxdirection,nodesydirection)

        i=i+1
        counter=counter+1
        maxres=Residualnew(Vk2,alpha,beta,deltax,deltay, bottomleftedgeXn

        if ( maxres<=Tolerance):

            x=Vk2[outputpoint[0]][outputpoint[1]]
            #x=counter
            #x=Vk2



            return  {'x':x, 'Vk2':Vk2 ,'counter':counter }

##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
##########################################################################
 ##########################################################################
##########################################################################
```



```
def Jacobisolve(Vk1,Vk2,deltax,deltay,alpha,beta, bottomleftedgeXnode,bot



    for i in range (1,bottomleftedgeXnode):
        for j in range (1,bottomleftedgeYnode):
            alpha1=alpha[i-1]
            alpha2=alpha[i]
            beta1=beta[j-1]
            beta2=beta[j]


        #first term
            term1=deltax*deltax
            term1=term1*deltay*deltay
            term1=term1*alpha1*alpha2*beta1*beta2
            term1=term1/2
            term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


        #second term
            term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
            term2=2/term2

        #third term
            term3=deltay*deltay*beta1*beta2*(beta1+beta2)
            term3=2/term3




        #change2 to 1 (mn el jehatein entabih)
            Vk2[i][j]= (term2 * alpha2 * Vk1[i-1][j])
            Vk2[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i
            Vk2[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
            Vk2[i][j]= (term3*beta1 * Vk1[i][j+1]) +              Vk1[i
            Vk2[i][j]= term1 *             Vk1[i][j]

        #vk1[i][j-1] vk1[i-1][j] will have been computed for k+1 to be us

        for j in range (bottomleftedgeYnode,nodesxdirection-1):

            alpha1=alpha[i-1]
            alpha2=alpha[i]
            beta1=beta[j-1]
            beta2=beta[j]
```

```
#first term
    term1=deltax*deltax
    term1=term1*deltay*deltay
    term1=term1*alpha1*alpha2*beta1*beta2
    term1=term1/2
    term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


#second term
    term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
    term2=2/term2

#third term
    term3=deltay*deltay*beta1*beta2*(beta1+beta2)
    term3=2/term3




#change2 to 1 (mn el jehatein entabih)
    Vk2[i][j]= (term2 * alpha2 * Vk1[i-1][j])
    Vk2[i][j]=  (term2 * alpha1 * Vk1[i+1][j] )          + Vk1[i]
    Vk2[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
    Vk2[i][j]= (term3*beta1 * Vk1[i][j+1]) +               Vk1[i
    Vk2[i][j]= term1 *             Vk1[i][j]



#boundary/beta1 term is removed
j=nodesydirection-1
alpha1=alpha[i-1]
alpha2=alpha[i]

beta1=beta[j-1]
beta2=beta1



#first term
term1=deltax*deltax
term1=term1*deltay*deltay
term1=term1*alpha1*alpha2*beta1*beta2
term1=term1/2
term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*delta


#second term
```

```
    term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
    term2=2/term2

    #third term
    term3=deltay*deltay*beta1*beta2*(beta1+beta2)
    term3=2/term3



    #vk[i][j+1] replaced with vk[i][j-1] and beta2 is used instead of
    #these parameters are equal due to mirror symmetry
    #change2 to 1 (mn el jehatein entabih)
    Vk2[i][j]= (term2 * alpha2 * Vk1[i-1][j])
    Vk2[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i][j]
    Vk2[i][j]= (term3 * beta1 * Vk1[i][j-1]) +              Vk1[i][j
    Vk2[i][j]= (term3*beta1 * Vk1[i][j-1]) +              Vk1[i][j]
    Vk2[i][j]= term1 *             Vk1[i][j]



for i in range (bottomleftedgeXnode,nodesxdirection-1):
    for j in range (1,bottomleftedgeYnode):
        alpha1=alpha[i-1]
        alpha2=alpha[i]
        beta1=beta[j-1]
        beta2=beta[j]


    #first term
        term1=deltax*deltax
        term1=term1*deltay*deltay
        term1=term1*alpha1*alpha2*beta1*beta2
        term1=term1/2
        term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*d


    #second term
        term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
        term2=2/term2

    #third term
        term3=deltay*deltay*beta1*beta2*(beta1+beta2)
        term3=2/term3
```

```
        #change2 to 1 (mn el jehatein entabih)
            Vk2[i][j]= (term2 * alpha2 * Vk1[i-1][j])
            Vk2[i][j]=  (term2 *  alpha1 * Vk1[i+1][j] )          + Vk1[i
            Vk2[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[
            Vk2[i][j]= (term3*beta1 * Vk1[i][j+1]) +               Vk1[i
            Vk2[i][j]= term1 *             Vk1[i][j]



    for j in range (1,bottomleftedgeYnode):
        i=nodesxdirection-1
        #alpha1 removed
        alpha1=alpha[i-1]
        alpha2=alpha1
        beta1=beta[j-1]
        beta2=beta[j]
        #first term
        term1=deltax*deltax
        term1=term1*deltay*deltay
        term1=term1*alpha1*alpha2*beta1*beta2
        term1=term1/2
        term1=term1/(beta1*beta2*deltay*deltay+alpha1*alpha2*deltax*delta
        #second term
        term2=deltax*deltax*alpha1*alpha2*(alpha1+alpha2)
        term2=2/term2
        #third term
        term3=deltay*deltay*beta1*beta2*(beta1+beta2)
        term3=2/term3



        #vk[i+1][j] and alpha2 replaced with vk[i-1][j] and alpha1 since
        #change2 to 1 mn el jehatein entabih
        Vk2[i][j]= (term2 * alpha1 * Vk1[i-1][j])
        Vk2[i][j]=  (term2 * alpha1 * Vk1[i-1][j] )          + Vk1[i][j]
        Vk2[i][j]= (term3 * beta2 * Vk1[i][j-1]) +              Vk1[i][j
        Vk2[i][j]= (term3*beta1 * Vk1[i][j+1]) +               Vk1[i][j]
        Vk2[i][j]= term1 *             Vk1[i][j]


    return Vk2




def jacobisolution(point,T,w,deltax,deltay, equalspacing, xydimension,Vin
```

```
#inner conudctor location
center=[xydimension[0]/2,xydimension[1]/2]
#deltaxanddeltay
if equalspacing==1:
    deltax=h
    deltay=h


edgex=(xydimension[0]/2)-centercond[1]/2



edgey=(xydimension[1]/2-centercond[0]/2)



point[0]


point[1]


pointcoordinateXcheck=0
pointcoordinateYcheck=0


equalspacing=1

#check validity of alpha/beta values



#numberofnodes/constant(for both equal and unequal)because the assign
if equalspacing==1:
    nodesxdirection = int((xydimension[0]/(2*deltax) + 1)) #perline
    nodesydirection = int(xydimension[1]/(2*deltay)+1) #perline
    n=nodesxdirection* nodesydirection #total number of nodes
    n=int(n)
    numberofalphas=nodesxdirection-1
    numberofbetas=nodesydirection-1
    alpha=[1]*(numberofalphas)
    beta=[1]*(numberofbetas)
    bottomleftedgeXnode=edgex/deltax
    bottomleftedgeYnode=edgey/deltay
    outpointx=int(point[0]/h)
    outpointy=int(point[1]/h)
```

```
    if int(point[0]/h)!=outpointx:
        print ("h not appropriate for including output point x coordi
        return

    if int(point[1]/h)!=outpointy:
        print ("h not appropriate for including output point y coordi
        return


    if int(bottomleftedgeXnode)!=bottomleftedgeXnode:
        print ("h not appropriate for assigning nodes to edges. choos
        return

    bottomleftedgeXnode=int(bottomleftedgeXnode)


    if int(bottomleftedgeYnode)!=bottomleftedgeYnode:
        print ("h not appropriate for assigning nodes to edges. choos
        return

    if h>xydimension[0] or h>xydimension[1]:
        print ("h not appropriate for problem geometry")
        return

    bottomleftedgeYnode=int(bottomleftedgeYnode)




#define nodes


voltagevector=[0]*(n)   #symmetry exploited here
Voltage0=Matrix(voltagevector,nodesxdirection,nodesydirection)      #
Voltage1=Matrix(voltagevector,nodesxdirection,nodesydirection)
Voltage2=Matrix(voltagevector,nodesxdirection,nodesydirection)
R=Matrix(voltagevector,nodesxdirection,nodesydirection)

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


    #lhon

#########################################################################
#########################################################################
#########################################################################
#########################################################################
```

```
#########################################################################
#########################################################################
#flag for inappropriate deltax/deltay/h choice : problem geometry spe




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

#Setting initial voltage values / problem geometry specific

for i in range (nodesxdirection):
    Voltage0 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage0 [i][0]= Vinnerouter[1]   #left 0v conductor / symmetry e
    Voltage1 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage1 [i][0]= Vinnerouter[1]
    Voltage2 [0][i]= Vinnerouter[1]   #bottom 0v conductor /symmetry
    Voltage2 [i][0]= Vinnerouter[1]



#center conductor voltages
for i in range (bottomleftedgeXnode,nodesxdirection):
    for j in range (bottomleftedgeYnode,nodesydirection):
        Voltage0 [i][j]=Vinnerouter[0]
        Voltage1 [i][j]=Vinnerouter[0]
        Voltage2 [i][j]=Vinnerouter[0]


#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#########################################################################
#numerical solution

#########################################################################
#step1 make a guess
#let the guess be zeros which is already there
```

```
#for statement begins here
#####################################################################
#compute gaus seidel method values at iteration k+1
#make a copy of voltage vector
Vk0=Voltage0 #k
Vk1=Voltage1 # after gauss
Vk2=Voltage2 #k+1 after SOR
#equationterms:
#terms matrices in terms of alpha and beta

outputpoint=[outpointx,outpointy]

#7ot equation gauss hon

true=1
i=0
#newgaussfile
counter=0
while (true):

    for i in range (nodesxdirection):
        for j in range (nodesydirection):
            Vk0[i][j]=Vk2[i][j]

    #Vk0 is the old value
    #Vk1 is the value after gauss
    #Vk2 is the value after the Relaxation

    #the relaxation takes the old(VK0) and the Gauss output(Vk1) as i



    Vk2=Jacobisolve(Vk1,Vk2,deltax,deltay,alpha,beta, bottomleftedgeX


    i=i+1
    counter=counter+1
    maxres=Residualnew(Vk2,alpha,beta,deltax,deltay, bottomleftedgeXn

    if ( maxres<=Tolerance):

        x=Vk2[outputpoint[0]][outputpoint[1]]
        #x=counter
        #x=Vk2
        return  {'x':x, 'Vk2':Vk2 ,'counter':counter }
    else:
        for i in range(i):
            for j in range(j):
                Vk1[i][j]=Vk2[i][j]
```

```
#q3dmain


#main unequal



Tolerance = 1/100000
h=0.01
deltax=0.01
deltay=0.01
w=1.5



#dimensions of the outer conductor
xdimension=0.2
ydimension= 0.2
#conductor voltages
Vinner=110
Vouter=0
#inner conductor height and width
height=0.04
width=0.08
#equalspacing
equalspacing=0


pointx=0.06
pointy=0.04

point=[pointx,pointy]



alpha=[0.1,0.4,1.6,1.8,1.2,            0.9   ,                0.9 ,
beta=[0.1,1.1,2,1.4,1.4,1.2,0.8 ,                  0.6 ,0.7,0.7]

sum=0
for i in range(10):
    sum=beta[i]+sum

deltaxy=[deltax,deltay]
xydimension=[xdimension,ydimension]
Vinnerouter=[Vinner,Vouter]
centercond=[height,width]
```

```
Vk2 = Jacobisolve (Vk1,Vk2,deltax,deltay,alpha,beta, bottomleftedgeXnode,

for i in range (nodesxdirection):
        for j in range (nodesydirection):
            Vk1[i][j]=Vk2[i][j]



Solution=SOR(point,Tolerance,w,deltax,deltay, equalspacing, xydimension,V
Solution
```

```
#Q3E

Tolerance = 1/100000
h=0.02
deltax=0.01
deltay=0.01
w=1.4



#dimensions of the outer conductor
xdimension=0.2
ydimension= 0.2
#conductor voltages
Vinner=110
Vouter=0
#inner conductor height and width
height=0.04
width=0.08
#equalspacing
equalspacing=0


pointx=0.06
pointy=0.04

point=[pointx,pointy]



alpha=[1,1,1,1,1,            1.8   ,                0.2 ,        0.2 ,
beta=[1.8,1,1,1,1,            0.2   ,                 0.2,        1 ,

sum=0
for i in range(10):
    sum=beta[i]+sum

deltaxy=[deltax,deltay]
xydimension=[xdimension,ydimension]
Vinnerouter=[Vinner,Vouter]
centercond=[height,width]




Solution=SOR(point,Tolerance,w,deltax,deltay, equalspacing, xydimension,V
Solution
```