**ECSE 543**

**Assignment 2**

**Numerical Methods in Electrical Engineering**

**AKEEL ALI: 260275389**

Table of Contents

## Question 1

### (a)

## Question 2

### (a)

The Python module listed in the Appendix was written to provide helper methods that build the structures for a mesh file. It creates two classes, Triangle and Node, and builds three structures with the method: a map of Nodes (nodeH), a list of triangles (triangleL), and a list of prescribed nodes (pList).

Extra care was taken to ensure that Python’s 0-based indexing was accounted for.

### (b)

was written that takes in the 3 data structures built in (a) and outputs the corresponding reduced matrix system (i.e., sparse matrix A and the vector b).

### (c)

is a program that takes in the reduced matrix system from (b) and solves the equation using Cholesky Decomposition.

### (d)

The program from (c) was adapted to find the potential at the point

Barycentric coordinates were computed for each triangle in a mesh and our point according to the following formulas [1]:

\lambda_1=\frac{(y_2-y_3)(x-x_3)+(x_3-x_2)(y-y_3)}{\det(T)}=\frac{(y_2-y_3)(x-x_3)+(x_3-x_2)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}\, ,

\lambda_2=\frac{(y_3-y_1)(x-x_3)+(x_1-x_3)(y-y_3)}{\det(T)}=\frac{(y_3-y_1)(x-x_3)+(x_1-x_3)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}\, ,

\lambda_3=1-\lambda_1-\lambda_2\, .

, where and , and the other coordinates are given by the nodes of the triangle in question.

The triangle giving us all three Barycentric coordinates obeying is the one containing our point whose potential would be given by:

, where are the three node voltages of the triangle.

The table and plot below show the relationship between the potential of the point and the number of unknowns or free nodes in the mesh.

|  |  |  |
| --- | --- | --- |
| Mesh | Free Nodes | Potential (V) |
| squareCoax3\_D | 44 | 4.095653785 |
| squareCoax4\_D | 84 | 4.029300427 |
| squareCoax5\_D | 144 | 4.049242342 |
| squareCoax6\_D | 220 | 4.048424185 |
| squareCoax7\_D | 312 | 3.99585225 |
| squareCoax8\_D | 406 | 3.989766296 |
| squareCoax9\_D | 544 | 3.963208292 |
| squareCoax10\_D | 666 | 3.969194918 |

## Question 3

### (a)

The conjugate gradient algorithm was implemented in found in the Appendix. The algorithm was taken straight from the notes, and the 2-norm residual tolerance threshold was set to .

The program was made to print out the infinity norm and 2-norm of the residual vector after each iteration for every mesh. These were calculated as follows:

The results for each mesh can be seen in the Appendix. The following plots show them in graphical form:

### (b)

The table below shows a comparison between the potential of point as calculated by both methods: Choleski decomposition and conjugate gradient.

|  |  |  |  |
| --- | --- | --- | --- |
| Mesh | Free Nodes | Choleski Potential (V) | CG Potential (V) |
| squareCoax3\_D | 44 | 4.095653785 | 4.095653785 |
| squareCoax4\_D | 84 | 4.029300427 | 4.029300427 |
| squareCoax5\_D | 144 | 4.049242342 | 4.049242342 |
| squareCoax6\_D | 220 | 4.048424185 | 4.048424185 |
| squareCoax7\_D | 312 | 3.99585225 | 3.99585225 |
| squareCoax8\_D | 406 | 3.989766296 | 3.989766296 |
| squareCoax9\_D | 544 | 3.963208292 | 3.963208292 |
| squareCoax10\_D | 666 | 3.969194918 | 3.969194918 |

It is clear from the table above that the values for the potential are exactly the same (up to 10 significant digits) for both methods.

**(c)**

## References

[1] Barycentric coordinate system, <http://en.wikipedia.org/wiki/Barycentric_coordinate_system_(mathematics)>

## Appendix

### Question 2 (a): MeshStructure.py

|  |
| --- |
| *#3 data structures for Problem 2 (a)*  nodeH = {}  triangleH = {}  triangleL = []  pList = []  **class** Node:  **def** \_\_init\_\_(self,number,x,y,voltage):  self.x = x  self.y = y  self.number = number  self.voltage = voltage    **def** \_\_repr\_\_(self):  **return** str(self.number)+**"("**+str(self.x)+**","**+str(self.y)+**")="**+str(self.voltage)    **class** Triangle:  **def** \_\_init\_\_(self,n1,n2,n3):  self.node1 = n1  self.node2 = n2  self.node3 = n3    **def** area(self):  **pass**    **def** \_\_repr\_\_(self):  **return** str(self.node1)+**"|"**+str(self.node2)+**"|"**+str(self.node3)    **def** buildStructs(filename):  import re    f = open(filename,**'r'**)    *# Create Node Hash#*  **global** nodeH  numNodes = int(f.readline())    **for** i in range(1,numNodes + 1):  line = f.readline()  m = re.match(**'(\d+.\d+) (\d+.\d+)'**,line)    nodeH[i] = Node(i,float(m.group(1)), float(m.group(2)), 0.0)      *# Create Triangle Hash#*  **global** triangleH  numT = int(f.readline())    **for** i in range(1, numT + 1):  line = f.readline()  *#TODO what is the last value?*  m = re.match(**'(\d+) (\d+) (\d+) (\d+.\d+)'**,line)  nodeNum1 = int(m.group(1))  nodeNum2 = int(m.group(2))  nodeNum3 = int(m.group(3))    triangleL.append(Triangle(nodeH[nodeNum1],nodeH[nodeNum2],nodeH[nodeNum3]))    *# Read in boundary conditions #*  *# Update Nodes + store prescribed in list #*  **global** pList  numBC = int(f.readline())    **for** i in range(1,numBC + 1):  line = f.readline()    m = re.match(**'(\d+) (\d+.\d+)'**,line)  nodeNum = int(m.group(1))  voltage = float(m.group(2))    nodeH[nodeNum].voltage = voltage;  pList.append(nodeNum)      f.close() |

### Question 2 (b): MeshSolver.py

|  |
| --- |
| from Matrix import Matrix  import MeshStructure  import math  *#from Choleski import Choleski*  S\_g = 0  S\_ff = 0  S\_fp = 0 *#actually holds -1\* S\_fp*  U\_f = 0  U\_p = 0  U\_con\_order = []  **def** getArea(triangle):  *#TODO i'm getting different areas in one mesh?*  n1 = triangle.node1  n2 = triangle.node2  n3 = triangle.node3    **return** math.fabs( (n1.x \* (n2.y - n3.y) + n2.x \* (n3.y - n1.y) + n3.x \* (n1.y - n2.y) )/ 2 )  *# returns the inverse of a 3x3 matrix*  **def** inverse3x3(matrix):  a = matrix.get(1,1)  b = matrix.get(1,2)  c = matrix.get(1,3)  d = matrix.get(2,1)  e = matrix.get(2,2)  f = matrix.get(2,3)  g = matrix.get(3,1)  h = matrix.get(3,2)  k = matrix.get(3,3)    determinant = a\*(e\*k-f\*h)+b\*(f\*g-k\*d)+c\*(d\*h-e\*g)    matrix.set(1,1,e\*k-f\*h)  matrix.set(1,2,f\*g-d\*k)  matrix.set(1,3,d\*h-e\*g)  matrix.set(2,1,c\*h-b\*k)  matrix.set(2,2,a\*k-c\*g)  matrix.set(2,3,g\*b-a\*h)  matrix.set(3,1,b\*f-c\*e)  matrix.set(3,2,c\*d-a\*f)  matrix.set(3,3,a\*e-b\*d)    inverse = matrix.transpose()    **for** i in range(1,3+1):  **for** j in range(1,3+1):  inverse.set(i,j,inverse.get(i,j)\*(1.0/determinant))    **return** inverse    **def** buildLocal\_S(triangle):  area = getArea(triangle)    n1 = triangle.node1  n2 = triangle.node2  n3 = triangle.node3    *# build X matrix*  X = Matrix([ [1, n1.x, n1.y],  [1, n2.x, n2.y],  [1, n3.x, n3.y]  ])  *# compute inverse(X)*  X\_inverse = inverse3x3(X)    *# compute transpose(X\_inverse)*  X\_i\_t = X\_inverse.transpose()    *# get delta\_alpha (i and j components)*  alphaI = Matrix([ [X\_i\_t.get(1,2)],  [X\_i\_t.get(2,2)],  [X\_i\_t.get(3,2)]  ])  alphaJ = Matrix([ [X\_i\_t.get(1,3)],  [X\_i\_t.get(2,3)],  [X\_i\_t.get(3,3)]  ])    S = alphaI.multiply(alphaI.transpose()).add(alphaJ.multiply(alphaJ.transpose()))    **for** i in range(1,3+1):  **for** j in range(1,3+1):  S.set(i,j,S.get(i,j)\*(area))    **return** S  **def** findJointNodes(triangleL):  joint=[]    **for** k in range(0,len(triangleL)):  k\_nodes=[]  k\_nodes.append(triangleL[k].node1)  k\_nodes.append(triangleL[k].node2)  k\_nodes.append(triangleL[k].node3)  *#compare starting from k+1 as previous have already compared*  **for** j in range(k+1,len(triangleL)):  j\_nodes=[]  j\_nodes.append(triangleL[j].node1)  j\_nodes.append(triangleL[j].node2)  j\_nodes.append(triangleL[j].node3)    *#compare the two triangles*  **for** a in range(0,len(k\_nodes)):  **for** b in range(0,len(j\_nodes)):  **if** (k\_nodes[a].x==j\_nodes[b].x and k\_nodes[a].y==j\_nodes[b].y and k\_nodes[a]!=j\_nodes[b]):  joint.append([k\_nodes[a].number,j\_nodes[b].number])    **return** joint      **def** buildGlobal\_S(nodeH, triangleL, pList):  **global** S\_g  **global** U\_f  **global** U\_p  **global** U\_con\_order  *#populate global S with zeroes*  **print** **"#create global S with zeroes"**  S\_g = Matrix(i=(len(triangleL)\*3),j=(len(triangleL)\*3))    *#build local S for each triangle and place it in global S*  **print** **"#build local S for each triangle and place it in global S"**  i = 1 *#index where we'll be placing the local S into global S*  j = 1  **for** k in range(0,len(triangleL)):  S\_l = buildLocal\_S(triangleL[k])  *#add it to S\_g in appropriate place*  **for** a in range(1,S\_l.rows+1):  **for** b in range(1, S\_l.columns+1):  S\_g.set(i,j,S\_l.get(a,b))  j+=1  i+=1  j-=3  j+=3    *#No need for findJoint in the case of our meshes as nodes are already joined*  *#joint = findJointNodes(triangleL)*    *#build a hash from id(node) -> node*  nodeID = {}  **for** i in range(1,len(nodeH)+1):  nodeID[id(nodeH[i])] = nodeH[i]    *#build U disjoint with node ids (rather than voltages)*  **print** **"#build U disjoint with node ids (rather than voltages)"**  U\_dis = Matrix(i=(len(triangleL)\*3),j=1)  row = 1  **for** k in range(0,len(triangleL)):  U\_dis.set(row,1,id(triangleL[k].node1))  U\_dis.set(row+1,1,id(triangleL[k].node2))  U\_dis.set(row+2,1,id(triangleL[k].node3))  row+=3      *#build U conjoint with node ids (and build C along the way)*  **print** **"#build U conjoint with node ids (and build C along the way)"**  C = Matrix(i=U\_dis.rows,j=len(nodeH))  row\_C = 1  U\_con = Matrix(i=len(nodeH),j=1)  row\_U\_c = 1 *#row in conjoint where we place our next element*  **for** row\_d in range(1,U\_dis.rows+1):  node\_id = U\_dis.get(row\_d,1)  add = True  *#check if it's already in U\_con*  **for** j in range(row\_U\_c-1,0,-1):  **if** (U\_con.get(j,1) == node\_id):  add = False  **break**  **if** (add):  *#add it to U\_con since it hasn't been added yet*  U\_con.set(row\_U\_c,1,node\_id)  C.set(row\_C,row\_U\_c,1.0)  row\_U\_c+=1  row\_C+=1  **else**:  C.set(row\_C,j,1.0)  row\_C+=1    *#rearrange U\_con and C (to put free up, and predefined down)*  **print** **"#rearrange U\_con and C (to put free up, and predefined down)"**  *#method used is 2 pointers moving in opposite directions on U\_con*  *#the pointer moving up looks for a free node, and the pointer moving down looks for predef node*  *#swap both, stop process when cross*  down = 1 *#index moving down*  up = U\_con.rows *#index moving up*  **while** (down < up):  **while** (down < up and nodeID[U\_con.get(down,1)].number not in pList):  down+=1  **if** (nodeID[U\_con.get(down,1)].number not in pList):  **break** *#nothing more to switch around*    **while** (up > down and nodeID[U\_con.get(up,1)].number in pList): *#looking for free from bottom*  up-=1    **if** (nodeID[U\_con.get(up,1)].number in pList):  **break** *#nothing more to switch around*    *#do the swap for U\_con*  tmp = U\_con.get(up,1)  U\_con.set(up,1,U\_con.get(down,1))  U\_con.set(down,1,tmp)    *#do the swap for C (swap columns)*  **for** i in range(1,C.rows+1):  tmp = C.get(i,up)  C.set(i,up,C.get(i,down))  C.set(i,down,tmp)    *#convert U\_con to actual voltage values (but keep track of ordering)*  **for** i in range(1,U\_con.rows+1):  U\_con\_order.append(nodeID[U\_con.get(i,1)].number)  U\_con.set(i,1,nodeID[U\_con.get(i,1)].voltage)    *#build U\_p and U\_f*  **print** **"#build U\_p and U\_f"**  U\_f = Matrix(i=down-1,j=1)  U\_p = Matrix(i=U\_con.rows-U\_f.rows,j=1)  **for** i in range(1,U\_f.rows+1):  U\_f.set(i,1,U\_con.get(i,1))  **for** i in range(1,U\_p.rows+1):  U\_p.set(i,1,U\_con.get(i+down-1,1))    *#finally compute S\_g*  **print** **"#finally compute S\_g"**  **print** **" #transpose C"**  C\_T = C.transpose()  **print** **" #multiply C transpose by S\_g"**  S\_g = (C\_T.multiply(S\_g))  **print** **" #multiply result by C"**  S\_g = S\_g.multiply(C)  **def** reduced():  **global** S\_ff  **global** S\_fp    S\_ff = Matrix(i=(U\_f.rows),j=(U\_f.rows))  S\_fp = Matrix(i=(U\_f.rows),j=(U\_p.rows))  **for** i in range(1,S\_ff.rows+1):  **for** j in range(1,S\_g.columns+1):  **if** (j <= U\_f.rows):  S\_ff.set(i,j,S\_g.get(i,j))  **else**:  S\_fp.set(i,j-U\_f.rows,-1\*S\_g.get(i,j))  **def** printResults(nodeH,pList,filename):  f = open(filename,**"w"**)    *#put results in appropriate nodes in nodeH*  **for** i in range(1,U\_f.rows+1):  nodeH[U\_con\_order[i-1]].voltage = U\_f.get(i,1)    f.write(**"node,x,y,potential,predefined\n"**)  **for** i in range(1,len(nodeH)+1):  f.write(str(nodeH[i].number)+**","**+str(nodeH[i].x)+**","**+str(nodeH[i].y)+**","**+str(nodeH[i].voltage))  **if** ( nodeH[i].number in pList):  f.write(**",y\n"**)  **else**:  f.write(**"\n"**)    f.close()    import sys  import pickle    **if** (len(sys.argv) > 1):  filename = sys.argv[1]  **print** **"--------------------------------"**  **print** **"Creating A, b for File "**+filename+**".msh"**  **print** **"--------------------------------"**  MeshStructure.buildStructs(filename+**".msh"**)  buildGlobal\_S(MeshStructure.nodeH, MeshStructure.triangleL, MeshStructure.pList)  reduced()  A = S\_ff  b = S\_fp.multiply(U\_p)    *#dump data*  fd = open(filename+**".dump"**,**'w'**)  data = [A,b,MeshStructure.nodeH,MeshStructure.pList,U\_con\_order]  pickle.dump(data,fd)  fd.close()    *#printResults(MeshStructure.nodeH,MeshStructure.pList,filename+".csv")*  **else**:  **print** **"please specify a filename without extension"** |

### Question 2 (c & d): CholeskiMesh.py

|  |
| --- |
| from Choleski import Choleski  from Matrix import Matrix  import MeshStructure  import pickle  import sys  **def** printResults(nodeH,pList,filename):  f = open(filename,**"w"**)    f.write(**"node,x,y,potential,predefined\n"**)  **for** i in range(1,len(nodeH)+1):  f.write(str(nodeH[i].number)+**","**+str(nodeH[i].x)+**","**+str(nodeH[i].y)+**","**+str(nodeH[i].voltage))  **if** ( nodeH[i].number in pList):  f.write(**",y\n"**)  **else**:  f.write(**"\n"**)    f.close()  **def** getAlpha(x1,y1,x2,y2,x3,y3,x,y):  denominator = (y2-y3)\*(x1-x3) + (x3-x2)\*(y1-y3)  **if** (denominator == 0):  **return** 99999 *#for infinity*  **else**:  **return** ((y2-y3)\*(x-x3)+(x3-x2)\*(y-y3)) / (denominator\*1.0)    **def** getBeta(x1,y1,x2,y2,x3,y3,x,y):  denominator = (y2-y3)\*(x1-x3) + (x3-x2)\*(y1-y3)  **if** (denominator == 0):  **return** 99999 *#for infinity*  **else**:  **return** ((y3-y1)\*(x-x3)+(x1-x3)\*(y-y3)) / (denominator\*1.0)    filename = sys.argv[1]  **print** **"--------------------------------"**  **print** **"Creating results csv for File "**+filename+**".msh"**  **print** **"--------------------------------"**  *#load data*  **print** **"#load data"**  fd = open(filename+**".dump"**,**'r'**)  data = pickle.load(fd)  fd.close()  A = data[0]  b = data[1]  nodeH = data[2]  pList = data[3]  U\_con\_order = data[4]  *#Call Choleski*  **print** **"#Call Choleski"**  U\_f = Choleski(A,b)  *#put results in appropriate nodes in nodeH*  **for** i in range(1,U\_f.rows+1):  nodeH[U\_con\_order[i-1]].voltage = U\_f.get(i,1)  *#nodeH at this point has all the proper voltages*  *#for each triangle, see if the point we're after is inside*  MeshStructure.buildStructs(filename+**".msh"**) *#we need triangleL*  triangleL=MeshStructure.triangleL  **for** i in range(0,len(triangleL)):  n1 = triangleL[i].node1  n2 = triangleL[i].node2  n3 = triangleL[i].node3  alpha = getAlpha(n1.x,n1.y,n2.x,n2.y,n3.x,n3.y,0.05,0.05)  beta = getBeta(n1.x,n1.y,n2.x,n2.y,n3.x,n3.y,0.05,0.05)  gamma = 1 - alpha - beta    **if** ((alpha >= 0 and alpha <=1) and (beta >=0 and beta <=1) and (gamma >=0 and gamma <=1)):  potential = nodeH[n1.number].voltage\*alpha + nodeH[n2.number].voltage\*beta + nodeH[n3.number].voltage\*gamma  f = open(**'Chol\_results.csv'**,**'a'**)  f.write( str(U\_f.rows)+**","**+str(potential)+**"\n"** )  f.close() |

### Question 3: CG.py

|  |
| --- |
| from Matrix import Matrix  import sys  import pickle  import copy  import math  import MeshStructure  LIMITERROR = 10\*\*-10  **def** printResults(nodeH,pList,filename):  f = open(filename,**"w"**)    f.write(**"node,x,y,potential,predefined\n"**)  **for** i in range(1,len(nodeH)+1):  f.write(str(nodeH[i].number)+**","**+str(nodeH[i].x)+**","**+str(nodeH[i].y)+**","**+str(nodeH[i].voltage))  **if** ( nodeH[i].number in pList):  f.write(**",y\n"**)  **else**:  f.write(**"\n"**)    f.close()    **def** getAlpha(x1,y1,x2,y2,x3,y3,x,y):  denominator = (y2-y3)\*(x1-x3) + (x3-x2)\*(y1-y3)  **if** (denominator == 0):  **return** 99999 *#for infinity*  **else**:  **return** ((y2-y3)\*(x-x3)+(x3-x2)\*(y-y3)) / (denominator\*1.0)    **def** getBeta(x1,y1,x2,y2,x3,y3,x,y):  denominator = (y2-y3)\*(x1-x3) + (x3-x2)\*(y1-y3)  **if** (denominator == 0):  **return** 99999 *#for infinity*  **else**:  **return** ((y3-y1)\*(x-x3)+(x1-x3)\*(y-y3)) / (denominator\*1.0)  **def** cg(A,b):  *#guess x all 0s*  x = Matrix(i=A.columns,j=1)  *#set r and p*  r = b.subtract(A.multiply(x))  p = b.subtract(A.multiply(x))    r\_norm\_inf = 0  **for** i in range(1,r.rows+1):  v = r.get(i,1)  **if** (v > r\_norm\_inf):  r\_norm\_inf = v  r\_norm\_2 = 0  r\_norm\_2 = math.sqrt(r.transpose().multiply(r).get(1,1))    iteration = 1    **while**( r\_norm\_2 > LIMITERROR):  p\_t = p.transpose()  alpha = p\_t.multiply(r).get(1,1) / p\_t.multiply(A.multiply(p)).get(1,1)      *# a\_p for alpha\*p*  a\_p = copy.deepcopy(p)  **for** i in range(1,a\_p.rows+1):  **for** j in range(1,a\_p.columns+1):  a\_p.set(i,j,a\_p.get(i,j)\*alpha)    x = x.add(a\_p)    r = b.subtract(A.multiply(x))  beta = -1 \* (p\_t.multiply(A.multiply(r)).get(1,1) / p\_t.multiply(A.multiply(p)).get(1,1))    *# b\_p for beta\*p*  b\_p = p *#no need to copy (as we update cell by cell of p to b\_p and we don't need it later)*  **for** i in range(1,b\_p.rows+1):  **for** j in range(1,b\_p.columns+1):  b\_p.set(i,j,p.get(i,j)\*beta)  p = r.add(b\_p)    *# compute r norms and f.write*  r\_norm\_inf = 0  **for** i in range(1,r.rows+1):  v = r.get(i,1)  **if** (v > r\_norm\_inf):  r\_norm\_inf = v    r\_norm\_2 = math.sqrt(r.transpose().multiply(r).get(1,1))    f.write(str(iteration)+**","**+str(r\_norm\_inf)+**","**+str(r\_norm\_2)+**"\n"**)  iteration+=1    f.write(**",,,"**+str(iteration-1))  **return** x    f = open(**'resultsCG.csv'**,**'a'**)  filename = sys.argv[1]  **print** **"--------------------------------"**  **print** **"Creating results csv for File "**+filename+**".msh"**  **print** **"--------------------------------"**  *#load data*  **print** **"#load data"**  fd = open(filename+**".dump"**,**'r'**)  data = pickle.load(fd)  fd.close()  A = data[0]  b = data[1]  nodeH = data[2]  pList = data[3]  U\_con\_order = data[4]  f.write(**"iteration,r\_norm\_inf,r\_norm\_2,iterations,free\_nodes,Potential\n"**)  U\_f = cg(A,b)  *#put results in appropriate nodes in nodeH*  **for** i in range(1,U\_f.rows+1):  nodeH[U\_con\_order[i-1]].voltage = U\_f.get(i,1)  *#nodeH at this point has all the proper voltages*  *#for each triangle, see if the point we're after is inside*  MeshStructure.buildStructs(filename+**".msh"**) *#we need triangleL*  triangleL=MeshStructure.triangleL  **for** i in range(0,len(triangleL)):  n1 = triangleL[i].node1  n2 = triangleL[i].node2  n3 = triangleL[i].node3  alpha = getAlpha(n1.x,n1.y,n2.x,n2.y,n3.x,n3.y,0.05,0.05)  beta = getBeta(n1.x,n1.y,n2.x,n2.y,n3.x,n3.y,0.05,0.05)  gamma = 1 - alpha - beta    **if** ((alpha >= 0 and alpha <=1) and (beta >=0 and beta <=1) and (gamma >=0 and gamma <=1)):  potential = nodeH[n1.number].voltage\*alpha + nodeH[n2.number].voltage\*beta + nodeH[n3.number].voltage\*gamma    f.write(**","**+str(U\_f.rows)+**","**+str(potential)+**"\n\n"**)    **break**    f.close() |

### Question 3 (a): Residual Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 8.183611 | 25.06276 |  |  |  |
| 2 | 5.136505 | 14.51312 |  |  |  |
| 3 | 2.575464 | 9.759611 |  |  |  |
| 4 | 1.38318 | 4.706173 |  |  |  |
| 5 | 0.438205 | 1.625396 |  |  |  |
| 6 | 0.169926 | 0.502354 |  |  |  |
| 7 | 0.116094 | 0.266468 |  |  |  |
| 8 | 0.017629 | 0.045056 |  |  |  |
| 9 | 0.001327 | 0.006205 |  |  |  |
| 10 | 0.000569 | 0.001752 |  |  |  |
| 11 | 2.82E-05 | 0.000141 |  |  |  |
| 12 | 7.11E-15 | 2.09E-14 |  |  |  |
|  |  |  | 12 | 44 | 4.095654 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 10.32568 | 30.02089 |  |  |  |
| 2 | 6.47568 | 20.65741 |  |  |  |
| 3 | 3.896625 | 13.23631 |  |  |  |
| 4 | 2.797801 | 9.636142 |  |  |  |
| 5 | 1.458879 | 6.712215 |  |  |  |
| 6 | 0.89874 | 3.184684 |  |  |  |
| 7 | 0.327874 | 1.18857 |  |  |  |
| 8 | 0.121198 | 0.429886 |  |  |  |
| 9 | 0.043923 | 0.217788 |  |  |  |
| 10 | 0.046523 | 0.119472 |  |  |  |
| 11 | 0.00847 | 0.041301 |  |  |  |
| 12 | 0.002794 | 0.013916 |  |  |  |
| 13 | 0.001102 | 0.005926 |  |  |  |
| 14 | 0.000199 | 0.000992 |  |  |  |
| 15 | 5.96E-05 | 0.000256 |  |  |  |
| 16 | 1.38E-05 | 7.30E-05 |  |  |  |
| 17 | 3.61E-06 | 1.89E-05 |  |  |  |
| 18 | 2.09E-07 | 1.04E-06 |  |  |  |
| 19 | 2.18E-08 | 1.28E-07 |  |  |  |
| 20 | 6.92E-09 | 2.62E-08 |  |  |  |
| 21 | 8.74E-13 | 2.31E-12 |  |  |  |
|  |  |  | 21 | 84 | 4.0293 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 9.362134 | 29.8383 |  |  |  |
| 2 | 5.505963 | 20.51544 |  |  |  |
| 3 | 3.925647 | 16.3883 |  |  |  |
| 4 | 2.72148 | 12.72624 |  |  |  |
| 5 | 1.915874 | 9.546177 |  |  |  |
| 6 | 1.291494 | 7.089931 |  |  |  |
| 7 | 0.705021 | 4.449247 |  |  |  |
| 8 | 0.451772 | 2.432864 |  |  |  |
| 9 | 0.219669 | 1.379232 |  |  |  |
| 10 | 0.123834 | 0.780344 |  |  |  |
| 11 | 0.052163 | 0.381432 |  |  |  |
| 12 | 0.038457 | 0.214572 |  |  |  |
| 13 | 0.027563 | 0.128428 |  |  |  |
| 14 | 0.029743 | 0.089772 |  |  |  |
| 15 | 0.012414 | 0.055393 |  |  |  |
| 16 | 0.006076 | 0.029976 |  |  |  |
| 17 | 0.005155 | 0.018856 |  |  |  |
| 18 | 0.00139 | 0.004888 |  |  |  |
| 19 | 0.000368 | 0.001793 |  |  |  |
| 20 | 8.29E-05 | 0.000381 |  |  |  |
| 21 | 2.15E-05 | 0.00017 |  |  |  |
| 22 | 1.64E-05 | 7.66E-05 |  |  |  |
| 23 | 5.14E-06 | 3.55E-05 |  |  |  |
| 24 | 2.48E-06 | 9.71E-06 |  |  |  |
| 25 | 1.05E-06 | 3.11E-06 |  |  |  |
| 26 | 2.41E-07 | 9.19E-07 |  |  |  |
| 27 | 4.95E-08 | 2.41E-07 |  |  |  |
| 28 | 1.49E-08 | 9.01E-08 |  |  |  |
| 29 | 2.70E-09 | 1.36E-08 |  |  |  |
| 30 | 7.93E-10 | 3.18E-09 |  |  |  |
| 31 | 7.93E-11 | 4.80E-10 |  |  |  |
| 32 | 2.31E-11 | 1.33E-10 |  |  |  |
| 33 | 6.32E-12 | 3.76E-11 |  |  |  |
|  |  |  | 33 | 144 | 4.049242 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 12.20984 | 34.26322 |  |  |  |
| 2 | 8.309486 | 23.3001 |  |  |  |
| 3 | 5.990655 | 18.20413 |  |  |  |
| 4 | 4.627383 | 15.38618 |  |  |  |
| 5 | 1.921657 | 12.11159 |  |  |  |
| 6 | 1.923738 | 10.48947 |  |  |  |
| 7 | 1.038399 | 8.122866 |  |  |  |
| 8 | 0.911753 | 6.712188 |  |  |  |
| 9 | 0.563234 | 4.672418 |  |  |  |
| 10 | 0.40032 | 2.750082 |  |  |  |
| 11 | 0.247922 | 1.546967 |  |  |  |
| 12 | 0.123197 | 0.930317 |  |  |  |
| 13 | 0.082799 | 0.610509 |  |  |  |
| 14 | 0.047767 | 0.353682 |  |  |  |
| 15 | 0.032485 | 0.228502 |  |  |  |
| 16 | 0.025405 | 0.157244 |  |  |  |
| 17 | 0.023788 | 0.109384 |  |  |  |
| 18 | 0.021427 | 0.076406 |  |  |  |
| 19 | 0.011135 | 0.050343 |  |  |  |
| 20 | 0.006506 | 0.02822 |  |  |  |
| 21 | 0.003491 | 0.019726 |  |  |  |
| 22 | 0.001858 | 0.00734 |  |  |  |
| 23 | 0.000718 | 0.003311 |  |  |  |
| 24 | 0.000171 | 0.001183 |  |  |  |
| 25 | 7.90E-05 | 0.000574 |  |  |  |
| 26 | 4.18E-05 | 0.000268 |  |  |  |
| 27 | 2.69E-05 | 0.00013 |  |  |  |
| 28 | 1.47E-05 | 7.25E-05 |  |  |  |
| 29 | 7.72E-06 | 3.83E-05 |  |  |  |
| 30 | 3.28E-06 | 1.72E-05 |  |  |  |
| 31 | 1.48E-06 | 6.15E-06 |  |  |  |
| 32 | 7.90E-07 | 3.25E-06 |  |  |  |
| 33 | 2.53E-07 | 1.22E-06 |  |  |  |
| 34 | 8.13E-08 | 3.44E-07 |  |  |  |
| 35 | 1.62E-08 | 9.60E-08 |  |  |  |
| 36 | 3.02E-09 | 2.08E-08 |  |  |  |
| 37 | 1.48E-09 | 9.96E-09 |  |  |  |
| 38 | 4.18E-10 | 3.34E-09 |  |  |  |
| 39 | 1.98E-10 | 1.20E-09 |  |  |  |
| 40 | 7.37E-11 | 4.45E-10 |  |  |  |
| 41 | 2.67E-11 | 1.28E-10 |  |  |  |
| 42 | 9.63E-12 | 3.63E-11 |  |  |  |
|  |  |  | 42 | 220 | 4.048424 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 8.836965 | 34.04321 |  |  |  |
| 2 | 5.815818 | 23.83505 |  |  |  |
| 3 | 4.003426 | 19.26285 |  |  |  |
| 4 | 2.806075 | 15.47209 |  |  |  |
| 5 | 2.35865 | 13.07946 |  |  |  |
| 6 | 1.938015 | 11.3715 |  |  |  |
| 7 | 1.444521 | 8.708836 |  |  |  |
| 8 | 1.229253 | 7.527394 |  |  |  |
| 9 | 0.983339 | 6.161051 |  |  |  |
| 10 | 0.761275 | 4.657478 |  |  |  |
| 11 | 0.430878 | 2.908916 |  |  |  |
| 12 | 0.291299 | 1.8538 |  |  |  |
| 13 | 0.205733 | 1.36421 |  |  |  |
| 14 | 0.14201 | 0.933509 |  |  |  |
| 15 | 0.072286 | 0.572446 |  |  |  |
| 16 | 0.049345 | 0.362502 |  |  |  |
| 17 | 0.031022 | 0.259818 |  |  |  |
| 18 | 0.02492 | 0.168389 |  |  |  |
| 19 | 0.019413 | 0.120778 |  |  |  |
| 20 | 0.016292 | 0.093505 |  |  |  |
| 21 | 0.012145 | 0.069447 |  |  |  |
| 22 | 0.006364 | 0.049417 |  |  |  |
| 23 | 0.003595 | 0.031158 |  |  |  |
| 24 | 0.003945 | 0.021831 |  |  |  |
| 25 | 0.001683 | 0.012265 |  |  |  |
| 26 | 0.000785 | 0.005591 |  |  |  |
| 27 | 0.000475 | 0.00234 |  |  |  |
| 28 | 0.000206 | 0.001133 |  |  |  |
| 29 | 9.42E-05 | 0.000592 |  |  |  |
| 30 | 3.65E-05 | 0.000319 |  |  |  |
| 31 | 1.30E-05 | 0.000133 |  |  |  |
| 32 | 9.41E-06 | 7.10E-05 |  |  |  |
| 33 | 5.34E-06 | 4.84E-05 |  |  |  |
| 34 | 4.75E-06 | 3.07E-05 |  |  |  |
| 35 | 1.65E-06 | 1.21E-05 |  |  |  |
| 36 | 5.84E-07 | 4.58E-06 |  |  |  |
| 37 | 3.43E-07 | 1.91E-06 |  |  |  |
| 38 | 1.05E-07 | 8.25E-07 |  |  |  |
| 39 | 3.65E-08 | 3.26E-07 |  |  |  |
| 40 | 1.73E-08 | 1.51E-07 |  |  |  |
| 41 | 8.50E-09 | 6.56E-08 |  |  |  |
| 42 | 3.05E-09 | 2.23E-08 |  |  |  |
| 43 | 8.97E-10 | 6.97E-09 |  |  |  |
| 44 | 4.96E-10 | 3.24E-09 |  |  |  |
| 45 | 2.43E-10 | 1.60E-09 |  |  |  |
| 46 | 7.48E-11 | 5.77E-10 |  |  |  |
| 47 | 2.63E-11 | 2.20E-10 |  |  |  |
| 48 | 9.39E-12 | 7.26E-11 |  |  |  |
|  |  |  | 48 | 312 | 3.995852 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 10.62671 | 37.55257 |  |  |  |
| 2 | 6.692795 | 25.29832 |  |  |  |
| 3 | 4.642575 | 20.18821 |  |  |  |
| 4 | 3.279914 | 16.50945 |  |  |  |
| 5 | 2.456898 | 14.01958 |  |  |  |
| 6 | 2.01835 | 12.22628 |  |  |  |
| 7 | 1.58172 | 10.39 |  |  |  |
| 8 | 1.237908 | 8.591781 |  |  |  |
| 9 | 1.107358 | 7.473543 |  |  |  |
| 10 | 0.878101 | 6.392326 |  |  |  |
| 11 | 0.660017 | 5.142608 |  |  |  |
| 12 | 0.458086 | 3.591412 |  |  |  |
| 13 | 0.345053 | 2.359248 |  |  |  |
| 14 | 0.23595 | 1.697189 |  |  |  |
| 15 | 0.140852 | 1.291011 |  |  |  |
| 16 | 0.076912 | 0.840964 |  |  |  |
| 17 | 0.054138 | 0.550409 |  |  |  |
| 18 | 0.03664 | 0.38302 |  |  |  |
| 19 | 0.026236 | 0.249338 |  |  |  |
| 20 | 0.019377 | 0.175528 |  |  |  |
| 21 | 0.01419 | 0.123047 |  |  |  |
| 22 | 0.010198 | 0.088989 |  |  |  |
| 23 | 0.009078 | 0.069867 |  |  |  |
| 24 | 0.008124 | 0.057414 |  |  |  |
| 25 | 0.006978 | 0.045099 |  |  |  |
| 26 | 0.004509 | 0.032204 |  |  |  |
| 27 | 0.003012 | 0.021197 |  |  |  |
| 28 | 0.002874 | 0.017478 |  |  |  |
| 29 | 0.001114 | 0.007776 |  |  |  |
| 30 | 0.000769 | 0.004589 |  |  |  |
| 31 | 0.000336 | 0.00202 |  |  |  |
| 32 | 0.000162 | 0.001188 |  |  |  |
| 33 | 8.20E-05 | 0.000666 |  |  |  |
| 34 | 4.96E-05 | 0.000369 |  |  |  |
| 35 | 2.50E-05 | 0.000153 |  |  |  |
| 36 | 9.93E-06 | 7.92E-05 |  |  |  |
| 37 | 3.84E-06 | 3.97E-05 |  |  |  |
| 38 | 2.71E-06 | 2.66E-05 |  |  |  |
| 39 | 2.41E-06 | 1.92E-05 |  |  |  |
| 40 | 1.35E-06 | 1.25E-05 |  |  |  |
| 41 | 7.50E-07 | 6.46E-06 |  |  |  |
| 42 | 3.84E-07 | 3.42E-06 |  |  |  |
| 43 | 2.27E-07 | 1.66E-06 |  |  |  |
| 44 | 1.05E-07 | 7.66E-07 |  |  |  |
| 45 | 3.98E-08 | 3.57E-07 |  |  |  |
| 46 | 2.22E-08 | 1.89E-07 |  |  |  |
| 47 | 1.11E-08 | 8.69E-08 |  |  |  |
| 48 | 8.44E-09 | 4.07E-08 |  |  |  |
| 49 | 6.52E-09 | 2.33E-08 |  |  |  |
| 50 | 1.25E-09 | 9.37E-09 |  |  |  |
| 51 | 7.11E-10 | 5.49E-09 |  |  |  |
| 52 | 4.12E-10 | 2.78E-09 |  |  |  |
| 53 | 1.40E-10 | 1.23E-09 |  |  |  |
| 54 | 5.29E-11 | 5.18E-10 |  |  |  |
| 55 | 2.65E-11 | 2.38E-10 |  |  |  |
| 56 | 1.06E-11 | 1.03E-10 |  |  |  |
| 57 | 7.60E-12 | 3.95E-11 |  |  |  |
|  |  |  | 57 | 406 | 3.989766 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 8.63357 | 37.67702 |  |  |  |
| 2 | 6.094363 | 26.71345 |  |  |  |
| 3 | 4.108534 | 21.71723 |  |  |  |
| 4 | 2.909657 | 17.35383 |  |  |  |
| 5 | 2.436617 | 14.83945 |  |  |  |
| 6 | 2.373506 | 13.12629 |  |  |  |
| 7 | 2.212448 | 11.39915 |  |  |  |
| 8 | 1.932398 | 10.35828 |  |  |  |
| 9 | 1.203462 | 8.848729 |  |  |  |
| 10 | 0.886595 | 7.085783 |  |  |  |
| 11 | 0.74505 | 6.680384 |  |  |  |
| 12 | 0.615378 | 5.561667 |  |  |  |
| 13 | 0.624318 | 4.841768 |  |  |  |
| 14 | 0.433617 | 3.340754 |  |  |  |
| 15 | 0.239324 | 2.281421 |  |  |  |
| 16 | 0.174902 | 1.680806 |  |  |  |
| 17 | 0.101031 | 1.414213 |  |  |  |
| 18 | 0.083721 | 1.101116 |  |  |  |
| 19 | 0.065702 | 0.716089 |  |  |  |
| 20 | 0.084526 | 0.531905 |  |  |  |
| 21 | 0.047329 | 0.415431 |  |  |  |
| 22 | 0.032551 | 0.290757 |  |  |  |
| 23 | 0.022371 | 0.212153 |  |  |  |
| 24 | 0.01571 | 0.159059 |  |  |  |
| 25 | 0.017233 | 0.120854 |  |  |  |
| 26 | 0.014328 | 0.103046 |  |  |  |
| 27 | 0.012412 | 0.089518 |  |  |  |
| 28 | 0.008572 | 0.065564 |  |  |  |
| 29 | 0.007021 | 0.051159 |  |  |  |
| 30 | 0.006309 | 0.04 |  |  |  |
| 31 | 0.003571 | 0.025079 |  |  |  |
| 32 | 0.002319 | 0.018488 |  |  |  |
| 33 | 0.002502 | 0.013752 |  |  |  |
| 34 | 0.000852 | 0.007263 |  |  |  |
| 35 | 0.000815 | 0.004476 |  |  |  |
| 36 | 0.000279 | 0.002404 |  |  |  |
| 37 | 0.000212 | 0.001578 |  |  |  |
| 38 | 0.000124 | 0.000878 |  |  |  |
| 39 | 8.96E-05 | 0.000528 |  |  |  |
| 40 | 3.83E-05 | 0.000276 |  |  |  |
| 41 | 2.90E-05 | 0.00018 |  |  |  |
| 42 | 1.55E-05 | 0.000103 |  |  |  |
| 43 | 9.67E-06 | 6.41E-05 |  |  |  |
| 44 | 5.79E-06 | 5.04E-05 |  |  |  |
| 45 | 3.67E-06 | 3.72E-05 |  |  |  |
| 46 | 3.12E-06 | 2.38E-05 |  |  |  |
| 47 | 1.09E-06 | 1.10E-05 |  |  |  |
| 48 | 7.57E-07 | 5.80E-06 |  |  |  |
| 49 | 4.02E-07 | 2.74E-06 |  |  |  |
| 50 | 1.60E-07 | 1.41E-06 |  |  |  |
| 51 | 6.97E-08 | 6.99E-07 |  |  |  |
| 52 | 5.24E-08 | 4.37E-07 |  |  |  |
| 53 | 3.16E-08 | 2.58E-07 |  |  |  |
| 54 | 1.73E-08 | 1.48E-07 |  |  |  |
| 55 | 1.28E-08 | 6.64E-08 |  |  |  |
| 56 | 3.29E-09 | 2.62E-08 |  |  |  |
| 57 | 1.22E-09 | 1.32E-08 |  |  |  |
| 58 | 7.95E-10 | 5.58E-09 |  |  |  |
| 59 | 3.29E-10 | 2.95E-09 |  |  |  |
| 60 | 1.91E-10 | 1.49E-09 |  |  |  |
| 61 | 9.96E-11 | 7.41E-10 |  |  |  |
| 62 | 4.56E-11 | 3.00E-10 |  |  |  |
| 63 | 1.51E-11 | 1.12E-10 |  |  |  |
| 64 | 6.32E-12 | 5.30E-11 |  |  |  |
|  |  |  | 64 | 544 | 3.963208 |
|  |  |  |  |  |  |
| Iteration | Infinity Norm | 2-Norm | iterations | free\_nodes | Potential |
| 1 | 9.94488 | 40.75443 |  |  |  |
| 2 | 6.082834 | 27.79755 |  |  |  |
| 3 | 4.385009 | 22.63236 |  |  |  |
| 4 | 3.000046 | 18.44592 |  |  |  |
| 5 | 2.458795 | 15.46647 |  |  |  |
| 6 | 2.230829 | 13.64231 |  |  |  |
| 7 | 1.968443 | 12.11255 |  |  |  |
| 8 | 2.271648 | 11.01782 |  |  |  |
| 9 | 2.135661 | 9.924905 |  |  |  |
| 10 | 1.610266 | 8.388033 |  |  |  |
| 11 | 0.982836 | 7.399087 |  |  |  |
| 12 | 0.63977 | 6.662406 |  |  |  |
| 13 | 0.933172 | 5.854334 |  |  |  |
| 14 | 0.659825 | 5.170104 |  |  |  |
| 15 | 0.48247 | 4.172977 |  |  |  |
| 16 | 0.37677 | 2.965372 |  |  |  |
| 17 | 0.261705 | 2.036194 |  |  |  |
| 18 | 0.159119 | 1.632395 |  |  |  |
| 19 | 0.167634 | 1.401331 |  |  |  |
| 20 | 0.12041 | 1.011553 |  |  |  |
| 21 | 0.076455 | 0.680783 |  |  |  |
| 22 | 0.070463 | 0.540386 |  |  |  |
| 23 | 0.053464 | 0.425983 |  |  |  |
| 24 | 0.040453 | 0.290633 |  |  |  |
| 25 | 0.022774 | 0.219971 |  |  |  |
| 26 | 0.015889 | 0.171075 |  |  |  |
| 27 | 0.010168 | 0.12197 |  |  |  |
| 28 | 0.010438 | 0.103091 |  |  |  |
| 29 | 0.011288 | 0.087026 |  |  |  |
| 30 | 0.008381 | 0.066806 |  |  |  |
| 31 | 0.00561 | 0.057009 |  |  |  |
| 32 | 0.006587 | 0.048838 |  |  |  |
| 33 | 0.003776 | 0.036933 |  |  |  |
| 34 | 0.002828 | 0.025939 |  |  |  |
| 35 | 0.003214 | 0.019414 |  |  |  |
| 36 | 0.002772 | 0.01686 |  |  |  |
| 37 | 0.001161 | 0.009628 |  |  |  |
| 38 | 0.000576 | 0.005649 |  |  |  |
| 39 | 0.000505 | 0.003294 |  |  |  |
| 40 | 0.000303 | 0.002152 |  |  |  |
| 41 | 0.000211 | 0.001384 |  |  |  |
| 42 | 0.00011 | 0.000876 |  |  |  |
| 43 | 6.04E-05 | 0.000455 |  |  |  |
| 44 | 3.22E-05 | 0.000297 |  |  |  |
| 45 | 2.00E-05 | 0.000176 |  |  |  |
| 46 | 1.41E-05 | 9.82E-05 |  |  |  |
| 47 | 4.91E-06 | 5.84E-05 |  |  |  |
| 48 | 3.26E-06 | 4.19E-05 |  |  |  |
| 49 | 2.59E-06 | 3.16E-05 |  |  |  |
| 50 | 2.53E-06 | 2.55E-05 |  |  |  |
| 51 | 1.49E-06 | 1.67E-05 |  |  |  |
| 52 | 1.13E-06 | 1.03E-05 |  |  |  |
| 53 | 5.16E-07 | 5.27E-06 |  |  |  |
| 54 | 3.25E-07 | 3.06E-06 |  |  |  |
| 55 | 1.24E-07 | 1.54E-06 |  |  |  |
| 56 | 8.19E-08 | 8.32E-07 |  |  |  |
| 57 | 5.29E-08 | 5.23E-07 |  |  |  |
| 58 | 4.53E-08 | 3.25E-07 |  |  |  |
| 59 | 2.57E-08 | 1.85E-07 |  |  |  |
| 60 | 1.44E-08 | 1.06E-07 |  |  |  |
| 61 | 6.38E-09 | 5.59E-08 |  |  |  |
| 62 | 3.97E-09 | 2.80E-08 |  |  |  |
| 63 | 1.71E-09 | 1.55E-08 |  |  |  |
| 64 | 1.46E-09 | 8.41E-09 |  |  |  |
| 65 | 1.30E-09 | 5.49E-09 |  |  |  |
| 66 | 6.23E-10 | 2.94E-09 |  |  |  |
| 67 | 2.58E-10 | 1.52E-09 |  |  |  |
| 68 | 1.58E-10 | 7.67E-10 |  |  |  |
| 69 | 7.44E-11 | 5.04E-10 |  |  |  |
| 70 | 4.13E-11 | 3.01E-10 |  |  |  |
| 71 | 2.67E-11 | 1.78E-10 |  |  |  |
| 72 | 7.81E-12 | 8.14E-11 |  |  |  |
|  |  |  | 72 | 666 | 3.969195 |