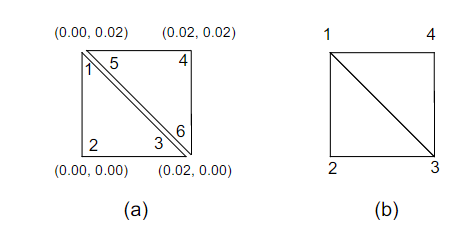
ECSE 543 Assignment 2

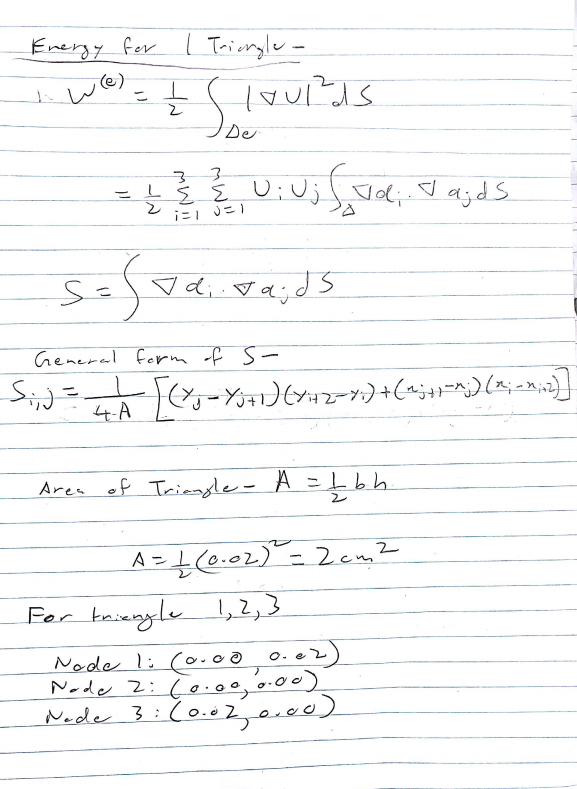
Razi Murshed

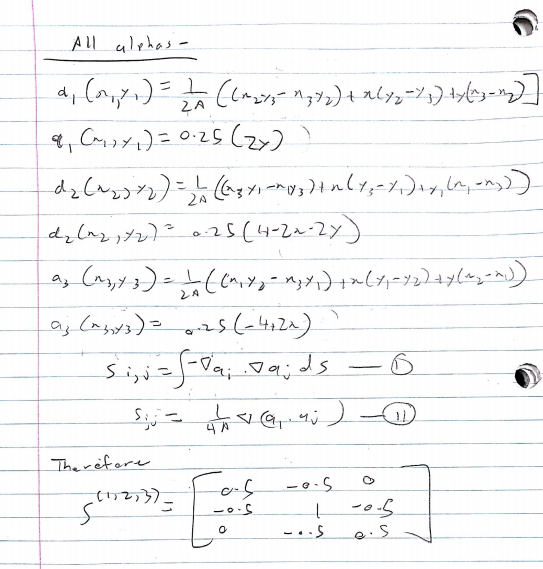
260516333

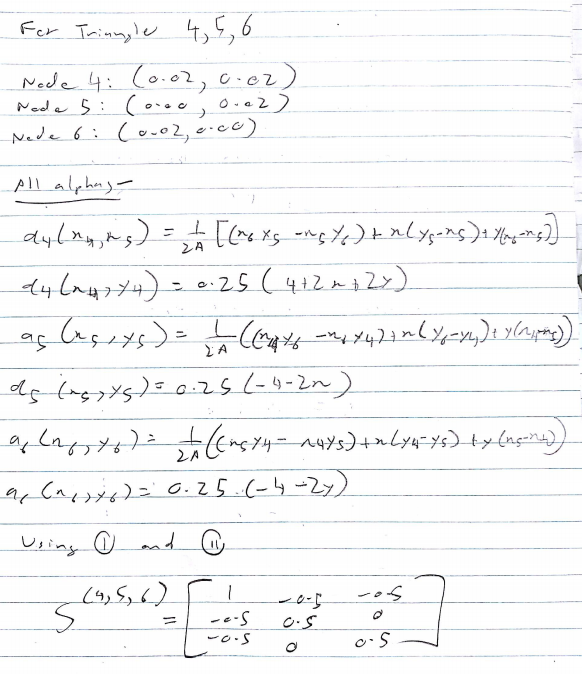
# Question 1

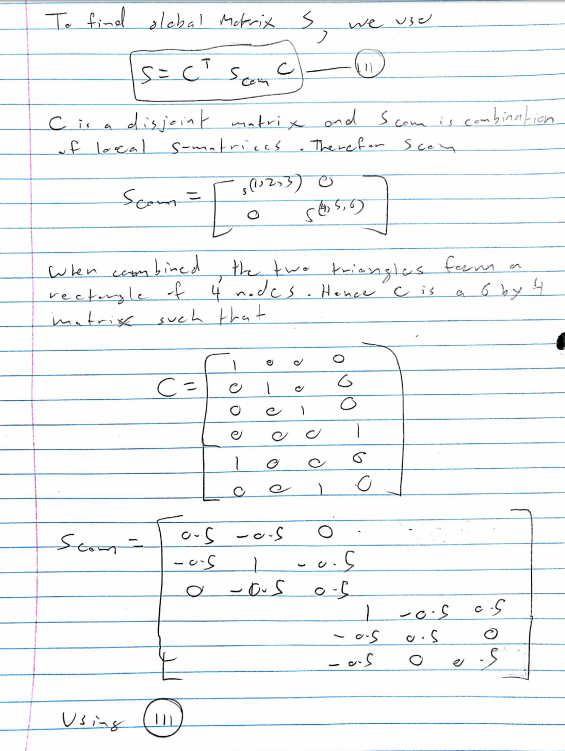


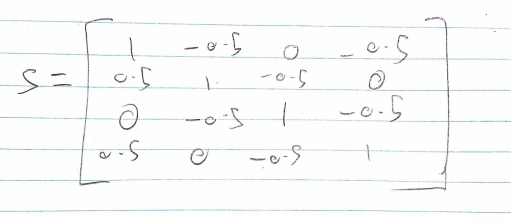
For each of the triangles shown above we can find the energy as shown below –











# Question 2

## Part A

The mesh was created according to the following diagram –

It was specified in an input file called “meshGen.txt”. This was created by hand and then converted to a .dat file for input to the SIMPLE2D program in order to give us the output file, “Simple2DOutput.txt”.

## Part B

From the file “Simple2DOutput.txt” we can see that the potential at 0.06,0.04 is 40.5265V.

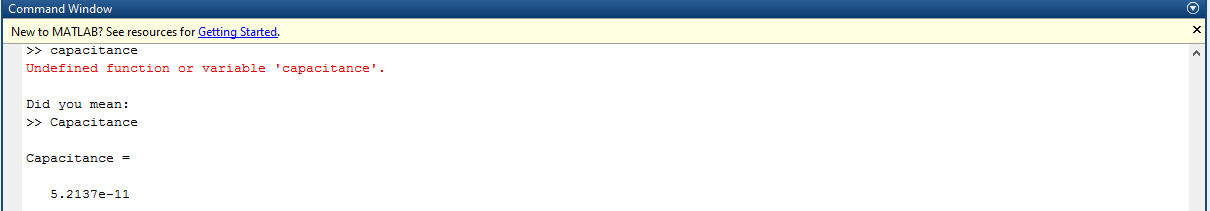
## Part C

We can find the capacitance by using the values found for energy in a capacitor and energy at a potential equation. After rearranging these equations, we come up with -

Then we find the energies of each element by using the equations as shown in question 1 and computing their sum. It is possible to find the energy of a square when given the potential at corners of the square. We find of the square to be –

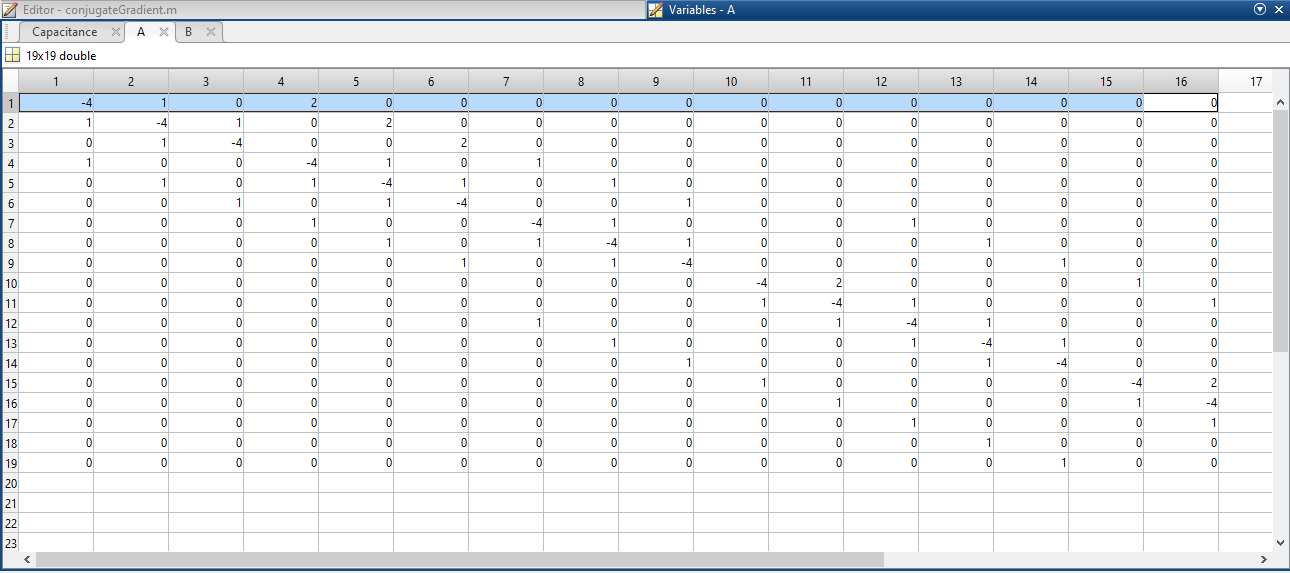
S is computed in question 1 and U is a vector of the 4 potentials at the corners of the square. If we assume these potentials at corner n to be Pn, the following equation can be derived –

Then we substitute in the capacitance equation as shown before and multiply the answer by 4 to find the potential of the Coaxial cable which comes out to be -

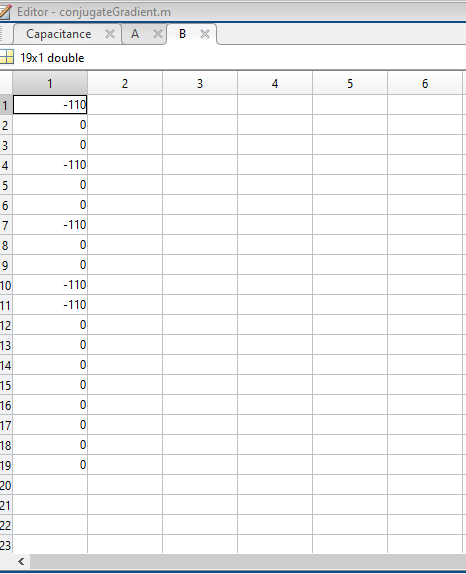


# Question 3

The code for this question can be found in the Appendix in the “conjugateGradient.m” file. The Cholesky Decomposition code can be found under “choleskyDecompose.m”. The matrix A was constructed as shown below –



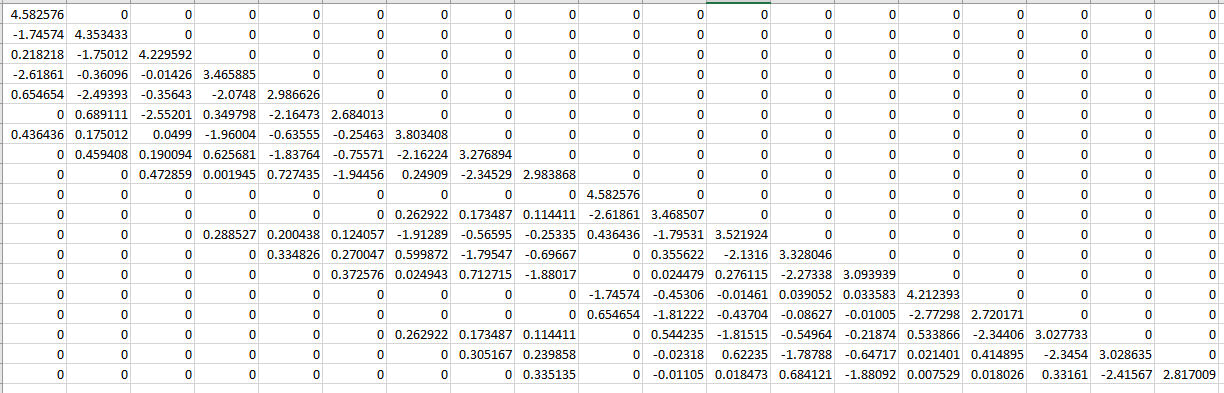
The vector B was constructed as shown below –



## Part a

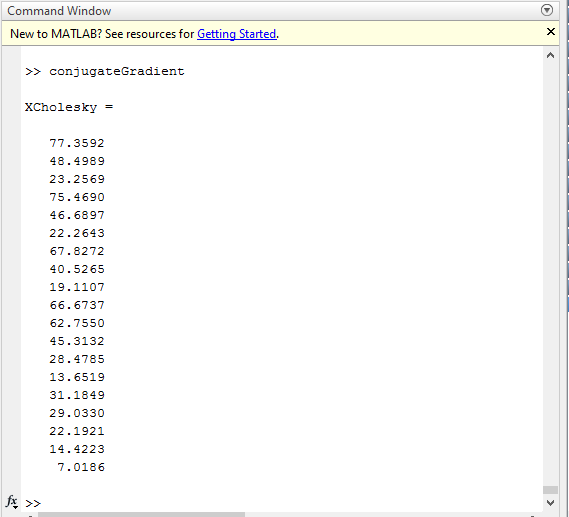
The matrix A was tested using the “choleskyDecompose.m” function. It failed seeing as A is not positive definite as the values it gave for the decomposed matrix were complex.

To solve this issue we can multiply A by AT in order to get an SPD matrix. To preserve the integrity of the equation we have to multiply both sides by AT to get . This fix works as we can see the decomposed matrix A below –

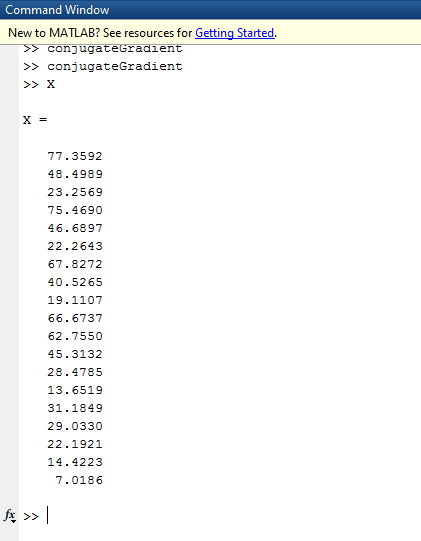


## Part B

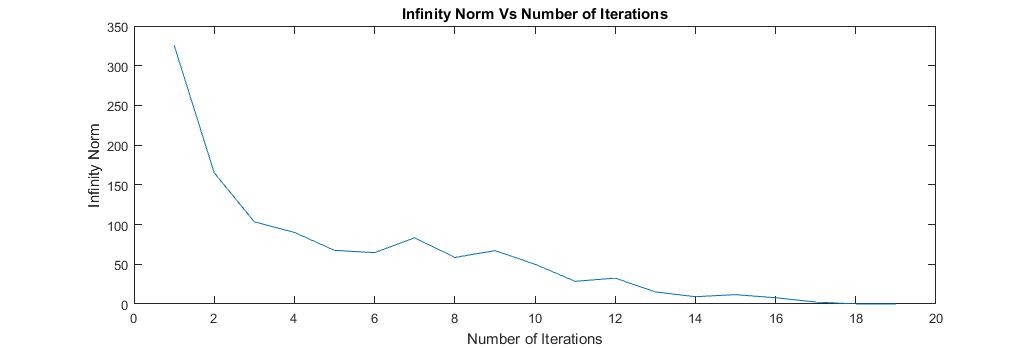
We proceed to solve the matrix equation by first using the cholesky decomposition method as was done in the previous assignment. Since the assignment was coded in MATLAB instead of Java this time, we rewrote the forward and back substitution parts as helper methods.

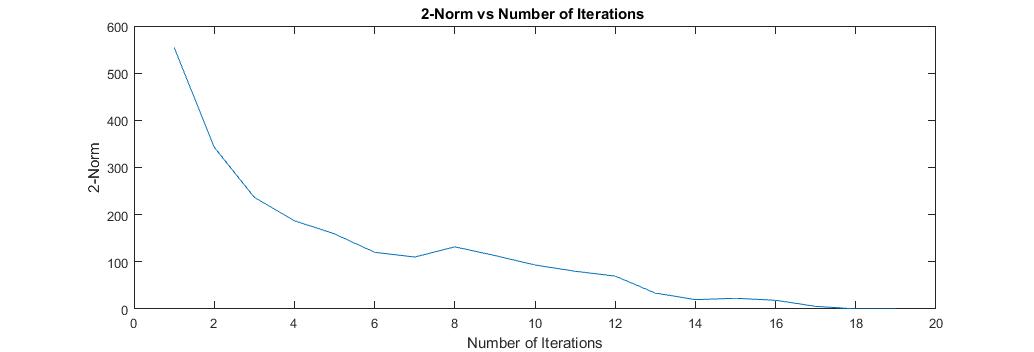


Then we move on to solve again but this time with the conjugate gradient method. This yields the following results –



PART C  
The following graphs display inf-norm and two norm for the conjugate program –





## part D

The results are summarized in the following table –

|  |  |
| --- | --- |
| Solving Method | Result(V) |
| Cholesky Decomposition | 40.5265 |
| Conjugate Gradient | 40.5265 |
| Simple 2D program | 40.5265 |
| IterateSOR | 40.5265 |

As we can see the values are all equal given that every method approximates a solution to Ax=b very closely. The true solution remains same regardless of the method used given an identical mesh.

## part E

We must find the potential at every node and then we can use the Simple 2D program and in conjunction with the methods used in question 2 we can find the capacitance per unit length.

# Appendix

## meshgen.txt

1 0.04 0.0

2 0.06 0.0

3 0.08 0.0

4 0.1 0.0

5 0.0 0.02

6 0.02 0.02

7 0.04 0.02

8 0.06 0.02

9 0.08 0.02

10 0.1 0.02

11 0.0 0.04

12 0.02 0.04

13 0.04 0.04

14 0.06 0.04

15 0.08 0.04

16 0.1 0.04

17 0.0 0.06

18 0.02 0.06

19 0.04 0.06

20 0.06 0.06

21 0.08 0.06

22 0.1 0.06

23 0.0 0.08

24 0.02 0.08

25 0.04 0.08

26 0.06 0.08

27 0.08 0.08

28 0.1 0.08

29 0.0 0.1

30 0.02 0.1

31 0.04 0.1

32 0.06 0.1

33 0.08 0.1

34 0.1 0.1

1 2 7 0.000

2 8 7 0.000

2 3 8 0.000

3 9 8 0.000

3 4 9 0.000

4 10 9 0.000

5 6 11 0.000

6 12 11 0.000

6 7 12 0.000

7 13 12 0.000

7 8 13 0.000

8 14 13 0.000

8 9 14 0.000

9 15 14 0.000

9 10 15 0.000

10 16 15 0.000

11 12 17 0.000

12 18 17 0.000

12 13 18 0.000

13 19 18 0.000

13 14 19 0.000

14 20 19 0.000

14 15 20 0.000

15 21 20 0.000

15 16 21 0.000

16 22 21 0.000

17 18 23 0.000

18 24 23 0.000

18 19 24 0.000

19 25 24 0.000

19 20 25 0.000

20 26 25 0.000

20 21 26 0.000

21 27 26 0.000

21 22 27 0.000

22 28 27 0.000

23 24 29 0.000

24 30 29 0.000

24 25 30 0.000

25 31 30 0.000

25 26 31 0.000

26 32 31 0.000

26 27 32 0.000

27 33 32 0.000

27 28 33 0.000

28 34 33 0.000

1 110.0

5 110.0

6 110.0

7 110.0

29 0.000

30 0.000

31 0.000

32 0.000

33 0.000

34 0.000

10 0.000

16 0.000

22 0.000

28 0.000

4 0.000

# Simple2DOutput.txt

1.0000 0.0400 0 110.0000

2.0000 0.0600 0 66.6737

3.0000 0.0800 0 31.1849

4.0000 0.1000 0 0

5.0000 0 0.0200 110.0000

6.0000 0.0200 0.0200 110.0000

7.0000 0.0400 0.0200 110.0000

8.0000 0.0600 0.0200 62.7550

9.0000 0.0800 0.0200 29.0330

10.0000 0.1000 0.0200 0

11.0000 0 0.0400 77.3592

12.0000 0.0200 0.0400 75.4690

13.0000 0.0400 0.0400 67.8272

14.0000 0.0600 0.0400 45.3132

15.0000 0.0800 0.0400 22.1921

16.0000 0.1000 0.0400 0

17.0000 0 0.0600 48.4989

18.0000 0.0200 0.0600 46.6897

19.0000 0.0400 0.0600 40.5265

20.0000 0.0600 0.0600 28.4785

21.0000 0.0800 0.0600 14.4223

22.0000 0.1000 0.0600 0

23.0000 0 0.0800 23.2569

24.0000 0.0200 0.0800 22.2643

25.0000 0.0400 0.0800 19.1107

26.0000 0.0600 0.0800 13.6519

27.0000 0.0800 0.0800 7.0186

28.0000 0.1000 0.0800 0

29.0000 0 0.1000 0

30.0000 0.0200 0.1000 0

31.0000 0.0400 0.1000 0

32.0000 0.0600 0.1000 0

33.0000 0.0800 0.1000 0

34.0000 0.1000 0.1000 0

## choleskyDecompose.m

function L = choleskyDecompose(M);

% Cholesky Decompose a Symmetric Positive Definite m by m matrix

n = length( M );

L = zeros( n, n );

for i=1:n

L(i, i) = sqrt(M(i, i) - L(i, :)\*L(i, :)');

for j=(i + 1):n

L(j, i) = (M(j, i) - L(i,:)\*L(j ,:)')/L(i, i);

end

end

end

## forwardSubstitution.m

function x=forwardSubstitution(L,b,n)

x=zeros(n,1);

for j=1:n

if (L(j,j)==0) error('Matrix is singular!'); end;

x(j)=b(j)/L(j,j);

b(j+1:n)=b(j+1:n)-L(j+1:n,j)\*x(j);

end

## BackSubstitution.m

function x=backSubstitution(U,b,n)

x=zeros(n,1);

for j=n:-1:1

if (U(j,j)==0) error('Matrix is singular!'); end;

x(j)=b(j)/U(j,j);

b(1:j-1)=b(1:j-1)-U(1:j-1,j)\*x(j);

end

## Q2Capacitance.m

nodesWide = 6;

nodesHigh = 6;

mesh = zeros(nodesWide, nodesHigh);

input = zeros(4, 34);

% read mesh values

file = fopen('Simple2DOutput.txt', 'r');

input = fscanf(file, '%f', size(input));

input = input';

fclose(file);

for x = 1: size(input, 1)

xNode = cast((input(x,2) / 0.02), 'int8');

yNode = cast((input(x,3) / 0.02), 'int8');

mesh(yNode + 1,xNode + 1) = input(x,4);

end

% % voltage is 10 volts on inner conductor

mesh(1,1) = 110.0;

mesh(1,2) = 110.0;

totalEnergy = 0.0;

for y = 1 : (nodesHigh - 1);

for x = 1: (nodesWide - 1);

u1 = mesh(y+1,x);

u2 = mesh(y,x);

u3 = mesh(y,x+1);

u4 = mesh(y+1,x+1);

totalEnergy = totalEnergy + (u1\*u1 - u1\*u2);

totalEnergy = totalEnergy + (-u1\*u4 + u2\*u2);

totalEnergy = totalEnergy + (-u2\*u3 + u3\*u3);

totalEnergy = totalEnergy + (-u3\*u4 + u4\*u4);

end

end

%

epsilon = 8.854187817620e-12;

voltageSquared = 12100;

Capacitance = totalEnergy\*(epsilon \* 4 / voltageSquared);

%

## conjugategradient.m

CABLE\_HEIGHT = 0.1;

CABLE\_WIDTH = 0.1;

CORE\_HEIGHT = 0.02;

CORE\_WIDTH = 0.04;

CORE\_POTENTIAL = 110;

MIN\_RESIDUAL = 0.0001;

height = 0.02;

nodesWide = int8(CABLE\_WIDTH/height) + 1;

nodesHigh = int8(CABLE\_HEIGHT/height) + 1;

nodeNumber = 19;

mesh = zeros(nodesWide,nodesHigh);

[rows, columns] = size(mesh);

for i = 1:rows

for j = 1:columns

if(( j <= int16(CORE\_WIDTH/height)+1)&&(i <= int16(CORE\_HEIGHT/height)+1))

mesh(i,j) = CORE\_POTENTIAL;

else

mesh(i,j) = 0;

end

end

end

%Constructing A and B matrices

B = zeros([nodeNumber 1]);

B(1) = -110;

B(4) = -110;

B(7) = -110;

B(10) = -110;

B(11) = -110;

A = zeros(nodeNumber) - [[4 -1 0 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[- 1 4 -1 0 -2 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

[0 -1 4 0 0 -2 0 0 0 0 0 0 0 0 0 0 0 0 0]

[- 1 0 0 4 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0 0]

[0 -1 0 -1 4 -1 0 -1 0 0 0 0 0 0 0 0 0 0 0]

[0 0 -1 0 -1 4 0 0 -1 0 0 0 0 0 0 0 0 0 0]

[0 0 0 -1 0 0 4 -1 0 0 0 -1 0 0 0 0 0 0 0]

[0 0 0 0 -1 0 -1 4 -1 0 0 0 -1 0 0 0 0 0 0]

[0 0 0 0 0 -1 0 -1 4 0 0 0 0 -1 0 0 0 0 0]

[0 0 0 0 0 0 0 0 0 4 -2 0 0 0 -1 0 0 0 0]

[0 0 0 0 0 0 0 0 0 -1 4 -1 0 0 0 -1 0 0 0]

[0 0 0 0 0 0 -1 0 0 0 -1 4 -1 0 0 0 -1 0 0]

[0 0 0 0 0 0 0 -1 0 0 0 -1 4 -1 0 0 0 -1 0]

[0 0 0 0 0 0 0 0 -1 0 0 0 -1 4 0 0 0 0 -1]

[0 0 0 0 0 0 0 0 0 -1 0 0 0 0 4 -2 0 0 0]

[0 0 0 0 0 0 0 0 0 0 -1 0 0 0 -1 4 -1 0 0]

[0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 -1 4 -1 0]

[0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 -1 4 -1]

[0 0 0 0 0 0 0 0 0 0 0 0 0 -1 0 0 0 -1 4]

];

%Cannot do this since it gives compleX values

%decomposedA = choleskYDecompose(A\*A');

%Solving using Cholesky

%Solution is to multiplY bY transpose

newA = A' \* A;

U = choleskyDecompose(newA);

B = A'\*B;

Ut = U';

%Solve the equation

Y = zeros([size(B) 1]);

X = zeros([size(B) 1]);

% Acuiring D vector

n = length(U);

% Now use a vector y to solve 'Ly=b'

Y=forwardSubstitution(U,B,n);

X=backSubstitution(Ut,Y, n);

XCholesky = X;

%Solving using conjugate gradient

X = zeros([nodeNumber 1]);

R = B-(newA\*X);

P = R;

infNormVec = [];

twoNormVec = [];

iterations = [];

results = zeros(2, nodeNumber);

for i = 1:nodeNumber

temp = (P'\*R);

temp1 = (P'\*newA\*P);

alpha = temp/temp1;

X = X + alpha\*P;

R = B - newA\*X;

temp = (P'\*newA\*R);

beta = ((-1)\*temp)/(temp1);

P = R + beta\*P;

results(1,1) = X(1);

results(2,1)= R(1);

for y = 1 : nodeNumber - 1

results(1,y+1) = X(y+1);

results(2,y+1)= R(y+1);

end

infNorm = 0;

twoNorm = 0;

for y = 1 : nodeNumber

val = abs(R(y,1));

if val > infNorm

infNorm = val;

end

twoNorm = twoNorm + (R(y,1))^2;

end

twoNorm = sqrt(twoNorm);

infNormVec = [infNormVec, infNorm];

twoNormVec = [twoNormVec, twoNorm];

iterations = [iterations,i];

end

figure

plot(iterations, infNormVec);

figure

plot(iterations, twoNormVec);