Homework 7 - CS6210

Ally Warner

April 1, 2015

1 Grid

Write a program to generate a 'finite difference grid' for the problem. This program should accept a value of h as input. To generate a grid means to break up the solution domain into uniform subregions and specifying the node points. It is acceptable to require that 1/h be an integer in order to avoid problems that arise when the boundary is not on a grid line.

To create my 'finite difference grid', I first calculated the size of the grid based on the input of h, which is the distance between nodes on our grid. I went through and plotted nodes from the bottom to the top of the grid, going left to right in each row. I saved the x and y positions of the nodes to be able to convert the grid into a matrix problem. I plotted my grid at five different values of h shown in the following figures. The code to create the grid follows.

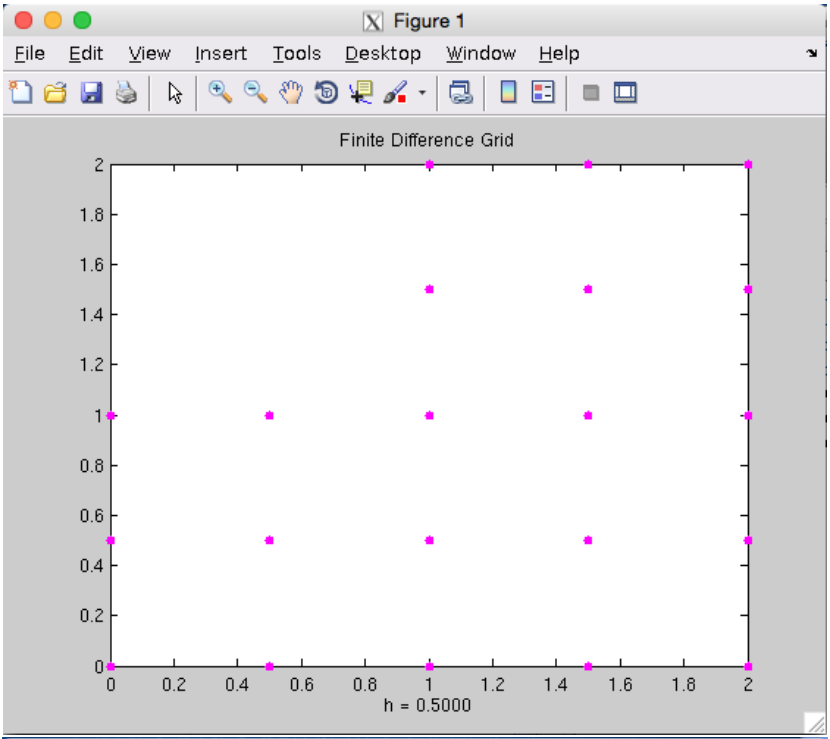


Figure 1
File Edit View Insert Tools Desktop Window Help
Finite Difference Grid
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
h = 0.5000

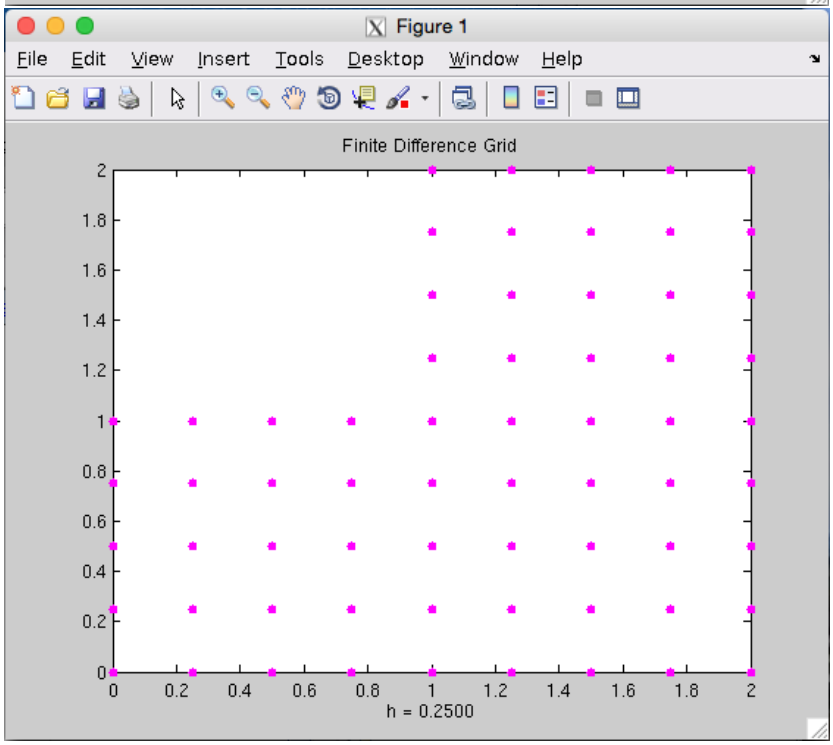


Figure 1
File Edit View Insert Tools Desktop Window Help
Finite Difference Grid
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
h = 0.2500

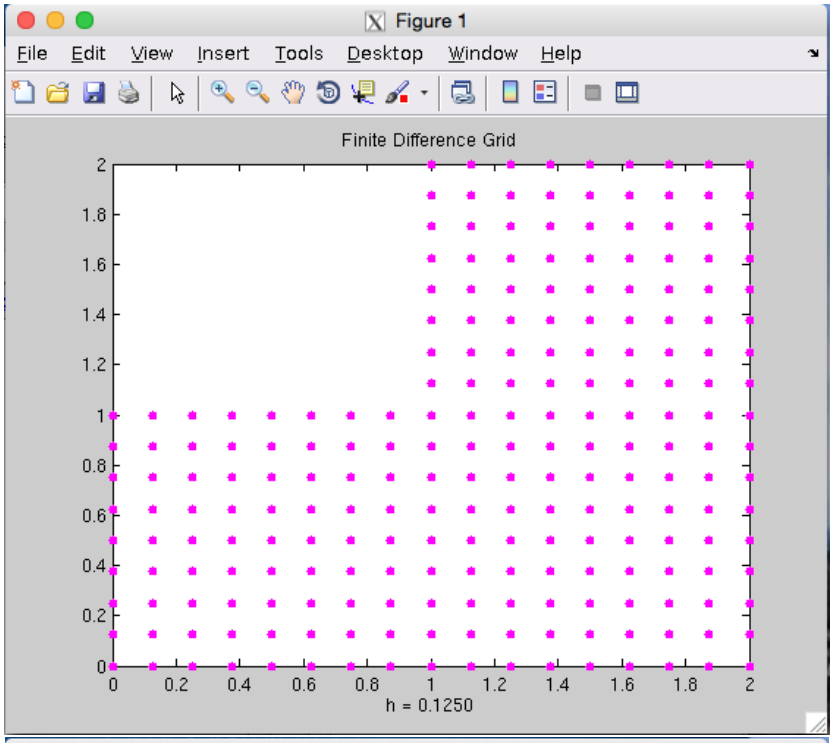


Figure 1
File Edit View Insert Tools Desktop Window Help
Finite Difference Grid
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
h = 0.1250

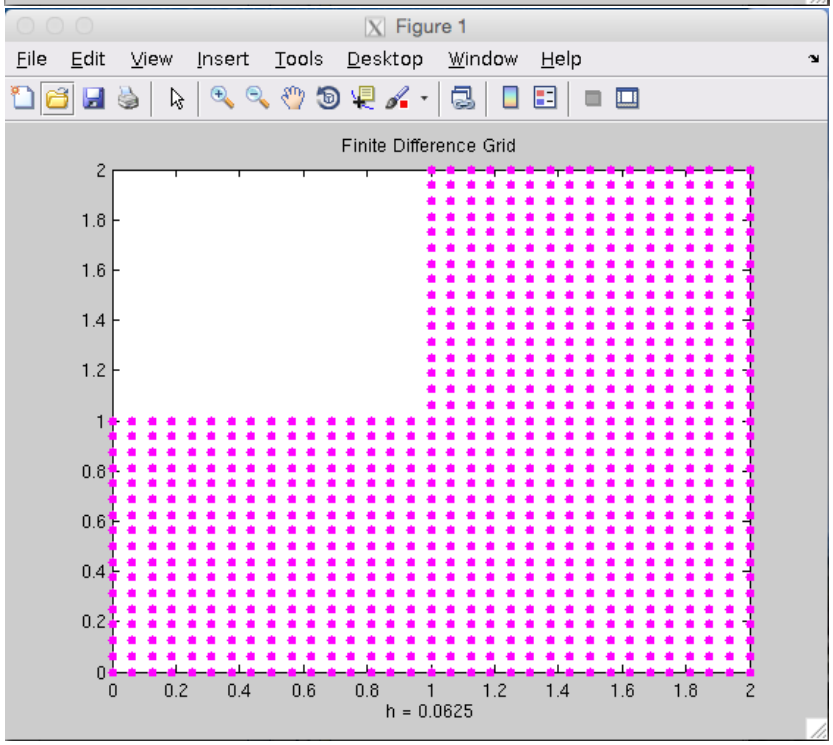


Figure 1
File Edit View Insert Tools Desktop Window Help
Finite Difference Grid
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
h = 0.0625

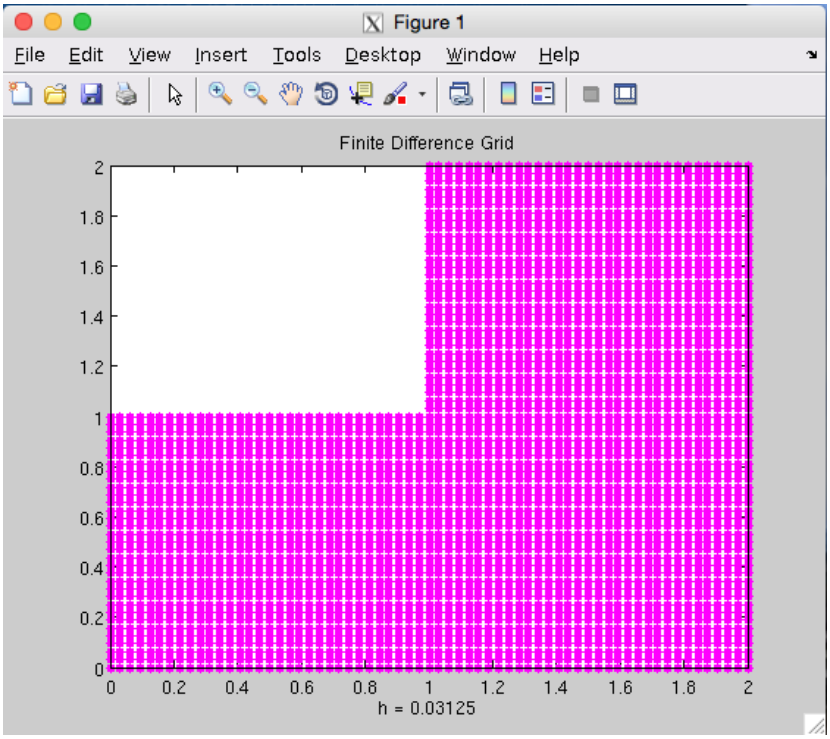


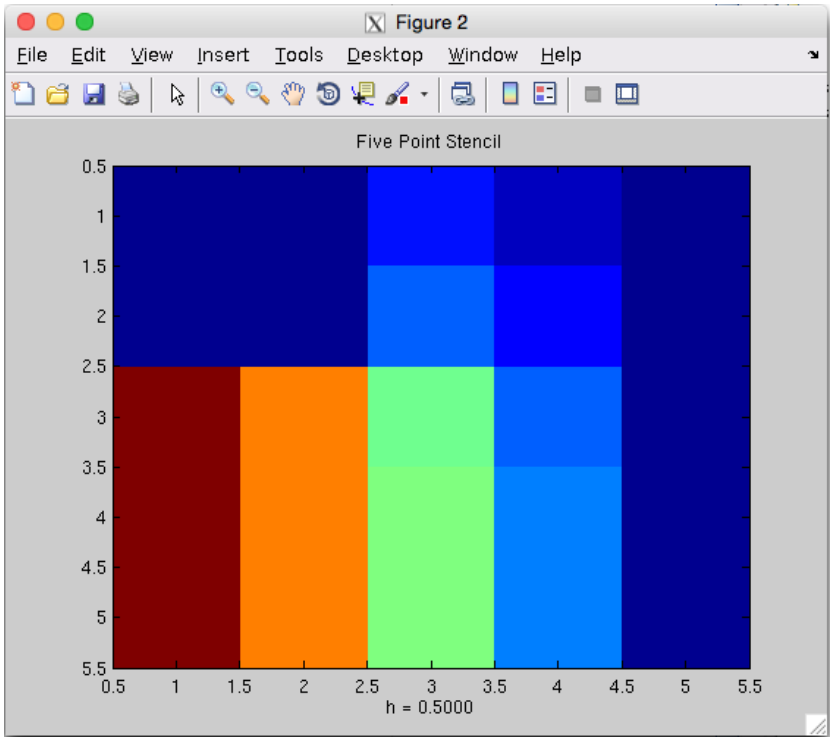
Figure 1
File Edit View Insert Tools Desktop Window Help
Finite Difference Grid
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2
h = 0.03125

```
function [xPosArr,gridSize] = makeGrid(h)
%Generates a finite difference grid in a L-Shaped region.

%Set at 0.5 for testing.
if nargin < 1
    h = 0.5;
end

%Checks to see if 1/h is an integer.
if floor(1/h) ~= 1/h
    error('1/h must be an integer, please input a different step size.');
end

%Sets the size of the grid based on h.
gridSize = (2/h) +1;

%Initialized the positions of the grid.
xPos = 0;
yPos = 0;
xPosArr = [];
yPosArr = [];

%Draws the bottom part of the grid and saves the positions.
for i = 1:ceil(gridSize/2)
    for j = 1:gridSize
        plot(xPos,yPos,'m.','MarkerSize',15);
        hold on
        xPosArr = [xPosArr;xPos];
        xPos = xPos + h;
    end
    yPosArr = [yPosArr;yPos];
    yPos = yPos + h;
    xPos = 0;
end

%Sets the width and height of the grid which is known to be 2 for this
%problem.
xlim([0 2])
ylim([0 2])

xPos = 1;

%Draws the top part of the grid and saves the positions.
for i = ceil(gridSize/2)+1:gridSize
    for j = ceil(gridSize/2):gridSize
        plot(xPos,yPos,'m.','MarkerSize',15);
        hold on
        xPosArr = [xPosArr;xPos];
        xPos = xPos + h;
    end
    yPosArr = [yPosArr;yPos];
    yPos = yPos + h;
    xPos = 1;
end

title('Finite Difference Grid')
xlabel(sprintf('h = %0.6f',h))

end
```

2 Solving a Linear System

Write a program to convert this grid into a matrix problem (a linear system Ax = b) using a 5 point centered difference finite difference approximation.

To convert my position information obtained from making the finite difference grid, I dealt with the bottom part of the region separately from the top part of the region to make the calculations easier to work with. Then for each case of node, weights were assigned to the current node and the four nodes surrounding. These weights were put into a matrix, A, where each row corresponds to a node. The solution matrix, b, was also created in this program based on the boundary conditions of our system. After A and b were constructed, I used the backslash operator in MATLAB to quickly solve this linear system and visualize my results. The results were visualized for five different values of h shown in the following figures. The code to create the stencil follows.



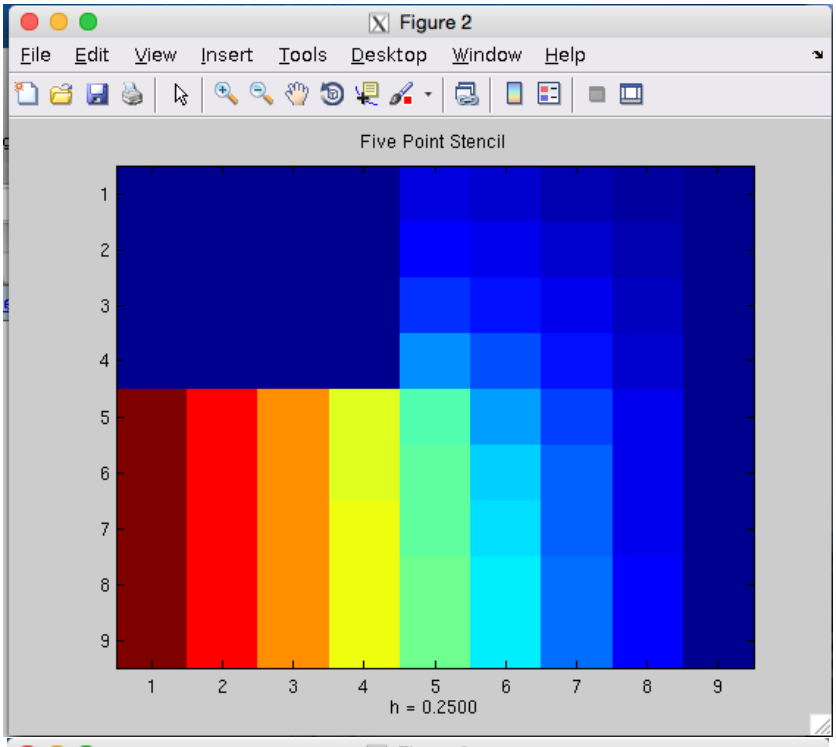


Figure 2
File Edit View Insert Tools Desktop Window Help
Five Point Stencil
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
h = 0.2500

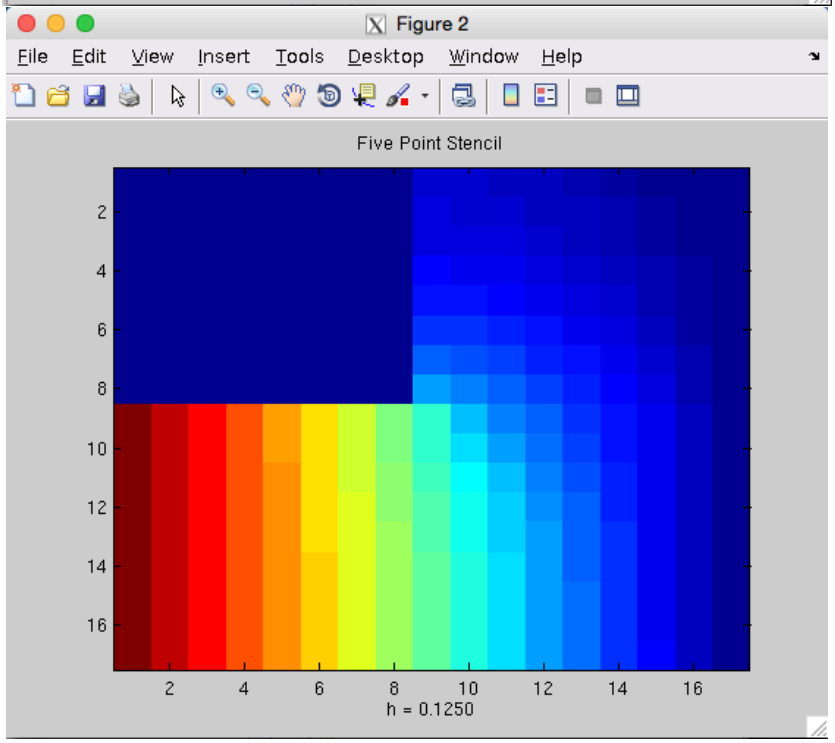


Figure 2
File Edit View Insert Tools Desktop Window Help
Five Point Stencil
2 4 6 8 10 12 14 16
2 4 6 8 10 12 14 16
h = 0.1250

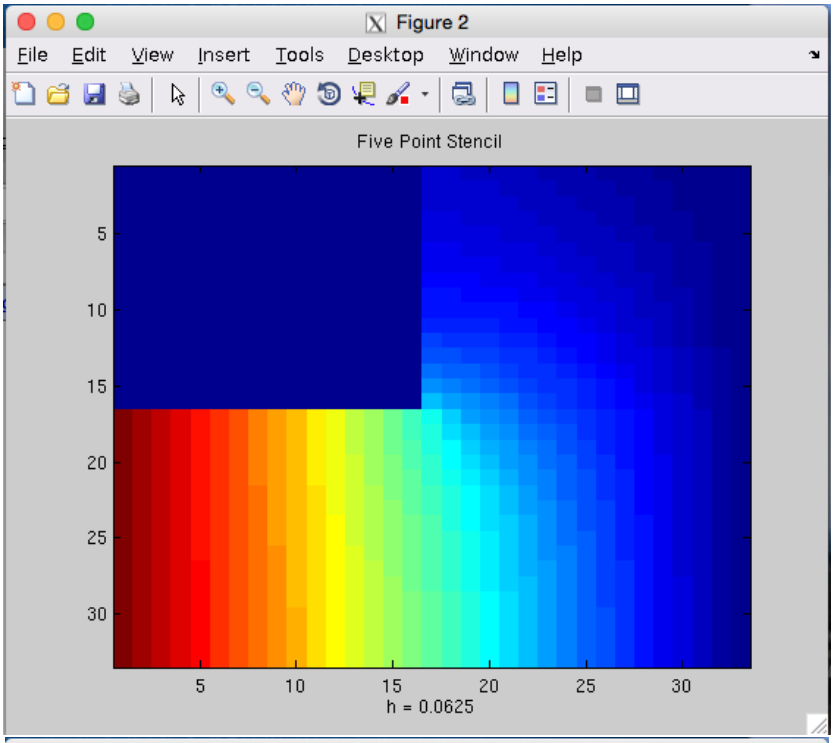


Figure 2
File Edit View Insert Tools Desktop Window Help
Five Point Stencil
5
10
15
20
25
30
5 10 15 20 25 30
h = 0.0625

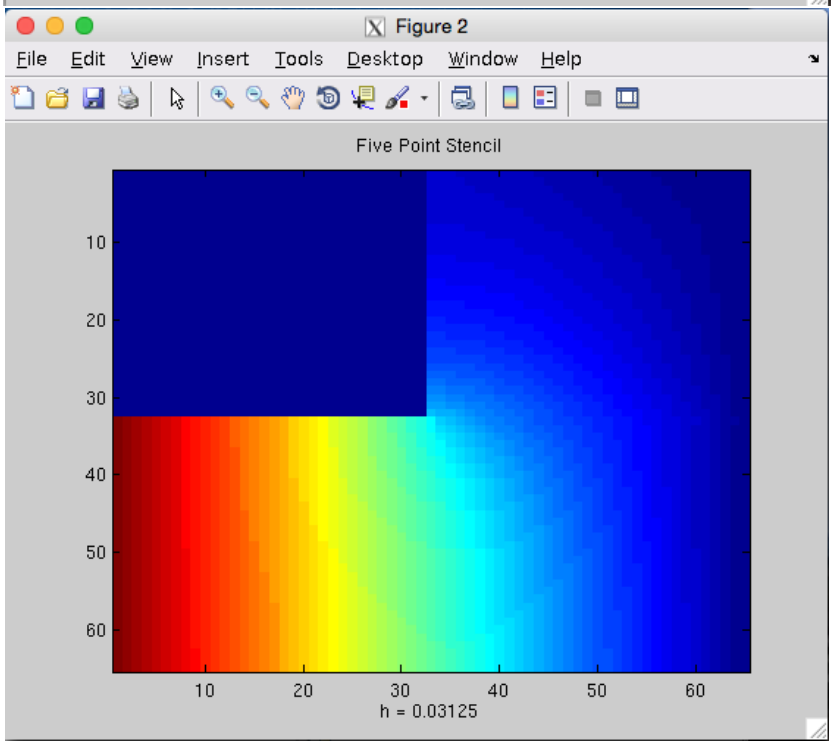


Figure 2
File Edit View Insert Tools Desktop Window Help
Five Point Stencil
10
20
30
40
50
60
10 20 30 40 50 60
h = 0.03125

```
function [A,b,e] = matrix5(hIn)
%Converts the grid information into a matrix A and b.
t = cputime;
%Gets the spatial info from the Grid function.
[xPosArr,gridSize] = makeGrid(hIn);
A = zeros(length(xPosArr));
b = zeros(length(xPosArr),1);
inx = 1;
%Makes the A and b arrays for the bottom half of the grid.
%Different conditions based on where you are in the space.
for i = 1:ceil(gridSize/2)
    for j = 1:gridSize
        if xPosArr(inx) == 0
            b(inx) = 1;
            A(inx,inx) = 1;
        elseif xPosArr(inx) == 2
            A(inx,inx) = 1;
        elseif i == 1
            A(inx,inx) = -4;
            A(inx,inx-1) = 1;
            A(inx,inx+1) = 1;
            A(inx,inx+gridSize) = 2;
        elseif xPosArr(inx) <= 1 && i == ceil(gridSize/2)
            A(inx,inx) = -4;
            A(inx,inx+1) = 1;
            A(inx,inx-1) = 1;
            A(inx,inx-gridSize) = 2;
        else
            A(inx,inx) = -4;
            A(inx,inx-1) = 1;
            A(inx,inx+1) = 1;
            A(inx,inx+gridSize) = 1;
            A(inx,inx-gridSize) = 1;
        end
        inx = inx + 1;
    end
end
```

```
%Makes the A and b arrays for the top half of the grid. Different
%conditions based on where you are in the space.
for i = ceil(gridSize/2)+1:gridSize
    for j = ceil(gridSize/2):gridSize
        if xPosArr(inx) == 2
            A(inx,inx) = 1;
        elseif xPosArr(inx) >= 1 && i == gridSize
            A(inx,inx) = -4;
            A(inx,inx+1) = 2;
            A(inx,inx-ceil(gridSize/2)) = 2;
        elseif xPosArr(inx) == 1
            A(inx,inx) = -4;
            A(inx,inx+1) = 2;
            A(inx,inx-ceil(gridSize/2)) = 1;
            A(inx,inx+ceil(gridSize/2)) = 1;
        elseif i == gridSize
            A(inx,inx) = -4;
            A(inx,inx+1) = 1;
            A(inx,inx-1) = 1;
            A(inx,inx-ceil(gridSize/2)) = 2;
        else
            A(inx,inx) = -4;
            A(inx,inx-1) = 1;
            A(inx,inx+1) = 1;
            A(inx,inx+ceil(gridSize/2)) = 1;
            A(inx,inx-ceil(gridSize/2)) = 1;
        end
        inx = inx + 1;
    end
end
e = cputime - t;
end
```

3 Solving using Iterative Solvers

Solve the resulting linear system using Jacobi, Gauss-Seidel, and Conjugate Gradient iterative solvers, and determine which solver worked best for this system. Solve the problem (from end- to-end) using several different values of h. Graph these results. Discuss the relationship between the number of elements and the time to solve the linear system.

Using prewritten Jacobi, Gauss-Seidel and Conjugate Gradient iterative solvers, I solved the linear system using the A and b matrices from the five-point stencil format for five different values of h. With the way I constructed my A matrix, the conjugate gradient solver wouldn't converge so I chose to use the biconjugate gradient solver instead. The results of these iterative solvers are shown in the figures below.

Jacobi:

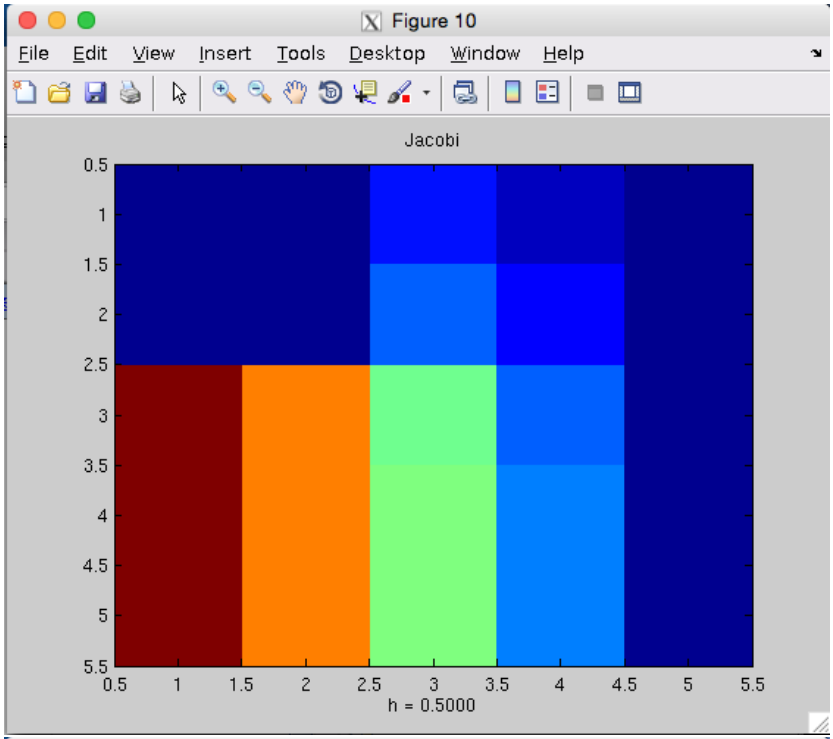


Figure 10
File Edit View Insert Tools Desktop Window Help
Jacobi
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5
h = 0.5000

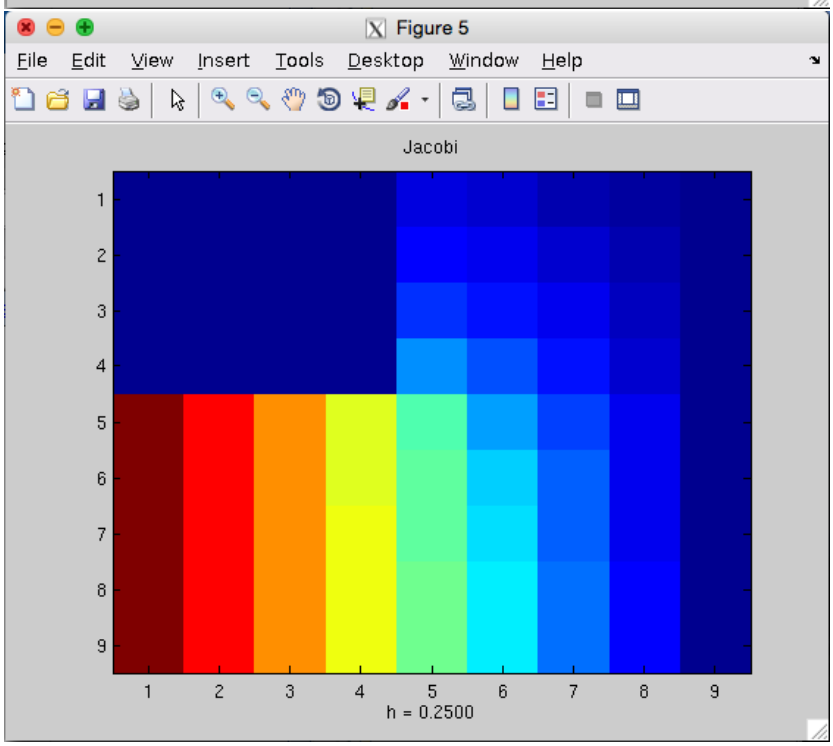


Figure 5
File Edit View Insert Tools Desktop Window Help
Jacobi
1
2
3
4
5
6
7
8
9
1 2 3 4 5 6 7 8 9
h = 0.2500

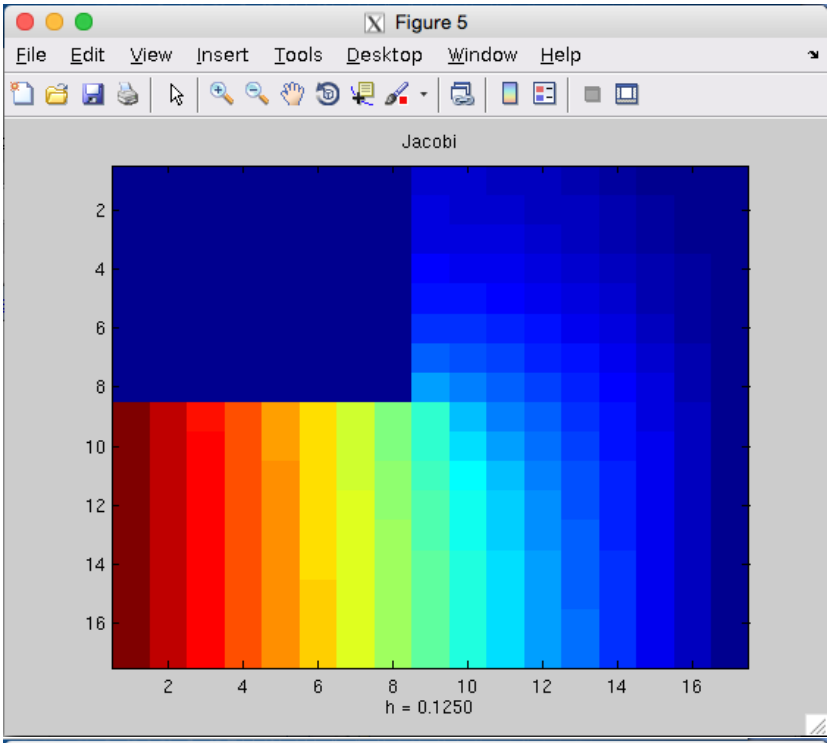


Figure 5
File Edit View Insert Tools Desktop Window Help
Jacobi
2
4
6
8
10
12
14
16
2 4 6 8 10 12 14 16
h = 0.1250

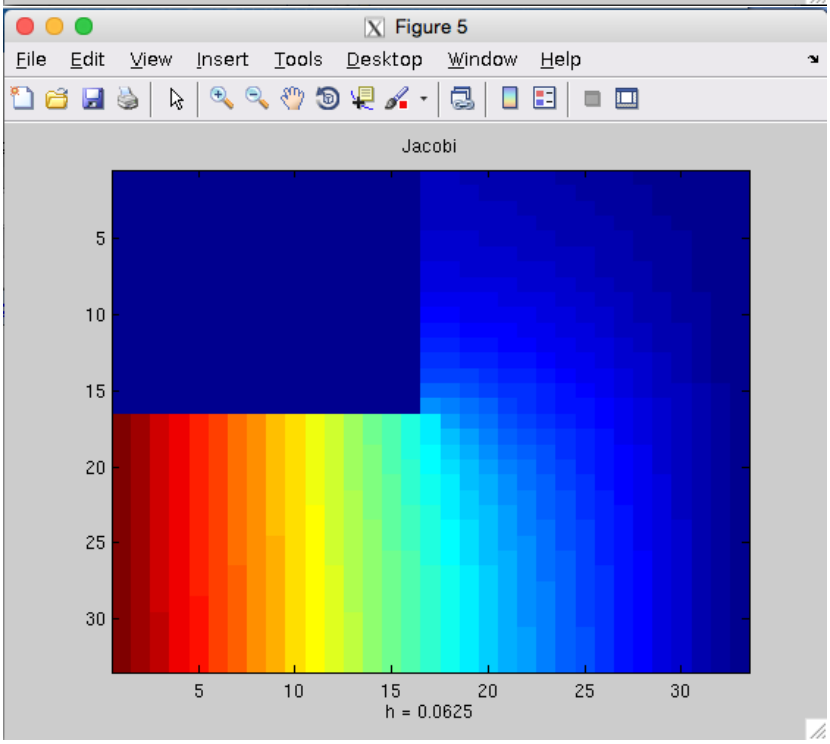
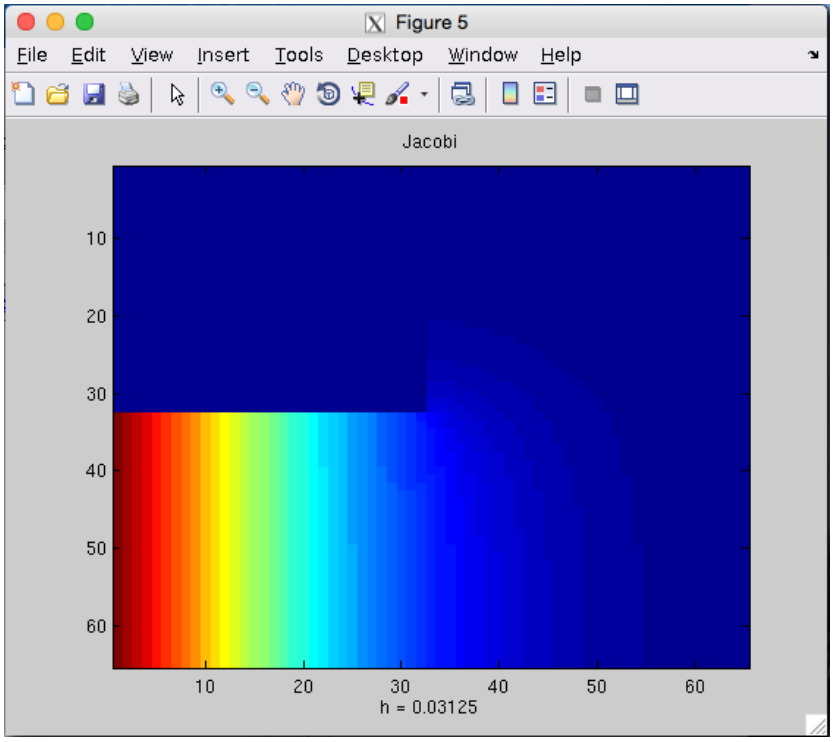


Figure 5
File Edit View Insert Tools Desktop Window Help
Jacobi
5
10
15
20
25
30
5 10 15 20 25 30
h = 0.0625



Biconjugate Gradient:

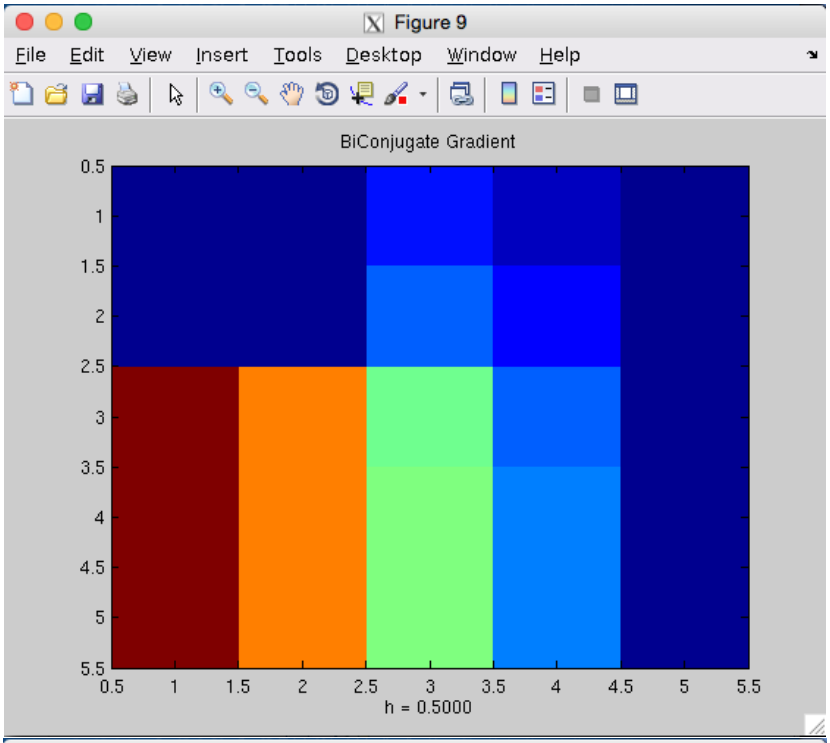


Figure 9
File Edit View Insert Tools Desktop Window Help
BiConjugate Gradient
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5
h = 0.5000

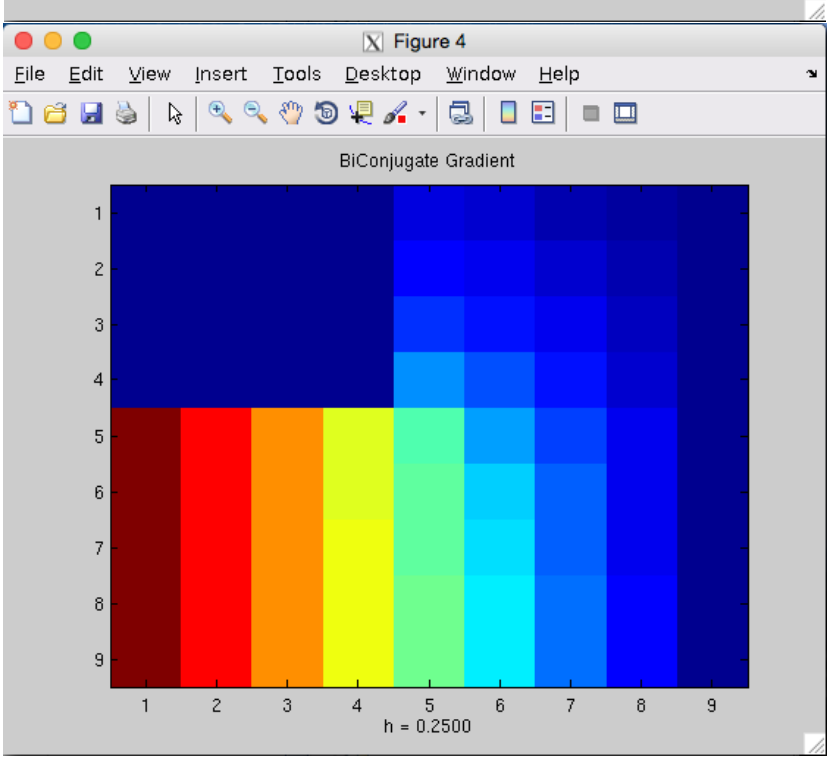


Figure 4
File Edit View Insert Tools Desktop Window Help
BiConjugate Gradient
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
h = 0.2500

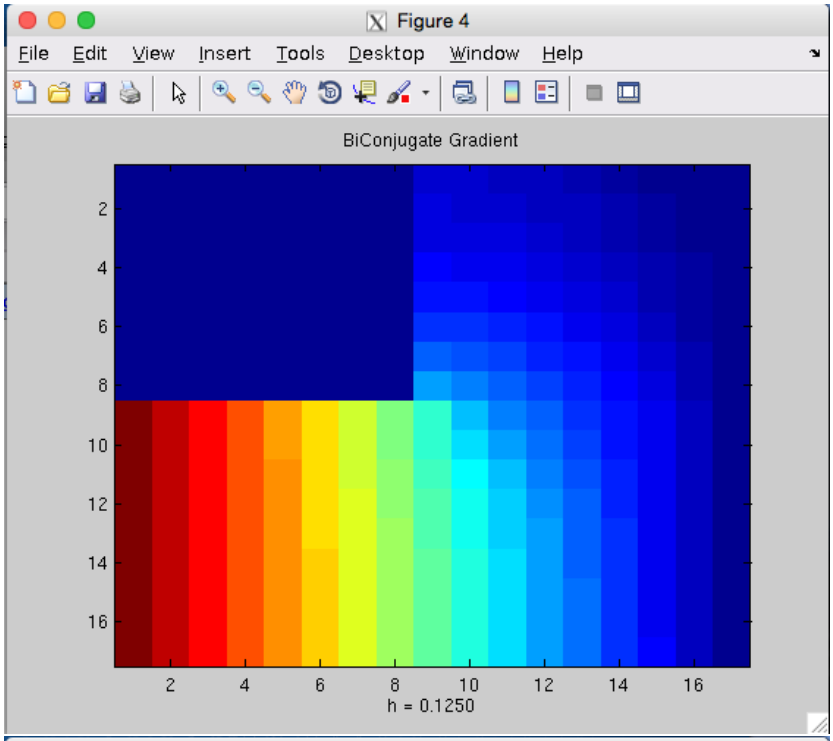


Figure 4
File Edit View Insert Tools Desktop Window Help
BiConjugate Gradient
2
4
6
8
10
12
14
16
2 4 6 8 10 12 14 16
h = 0.1250

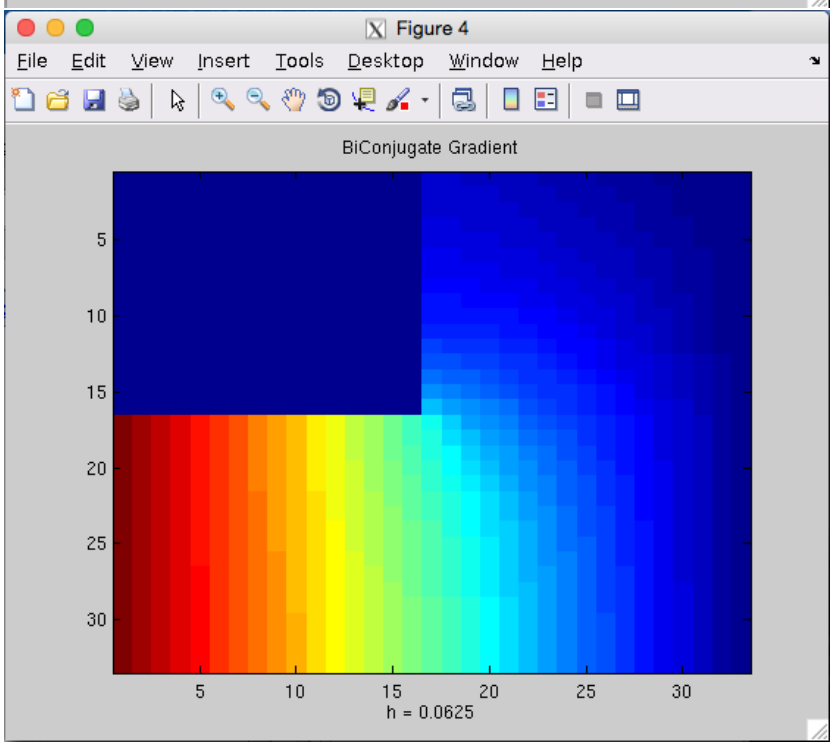
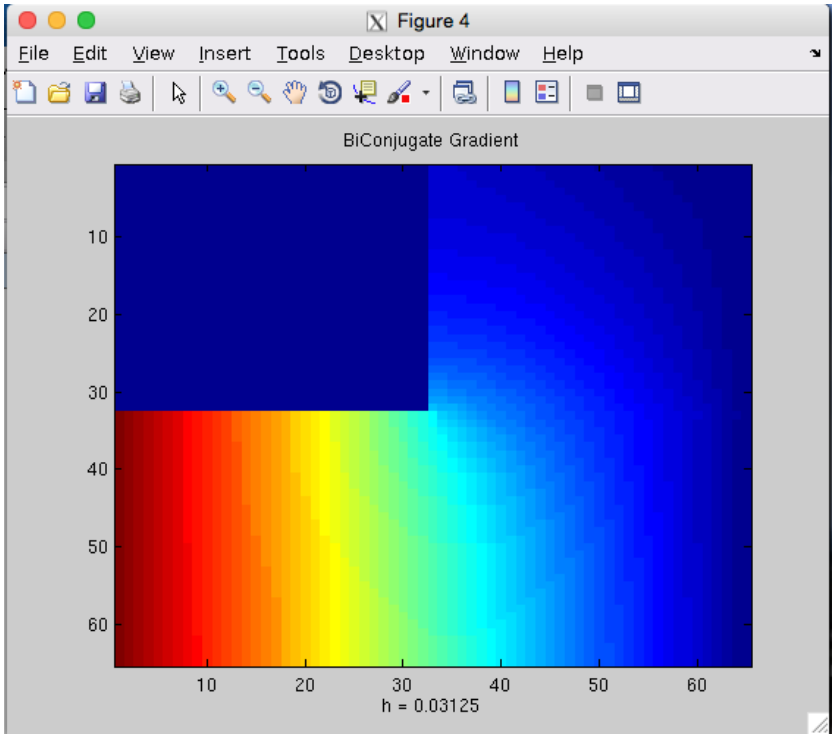


Figure 4
File Edit View Insert Tools Desktop Window Help
BiConjugate Gradient
5
10
15
20
25
30
5 10 15 20 25 30
h = 0.0625



Gauss-Seidel:

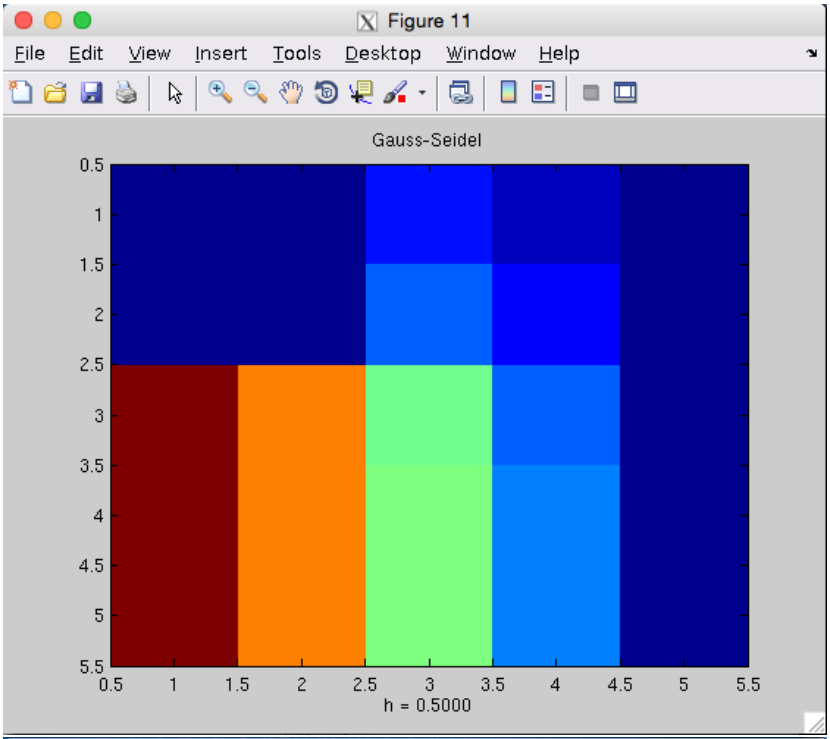


Figure 11
File Edit View Insert Tools Desktop Window Help
Gauss-Seidel
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5
0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5
h = 0.5000

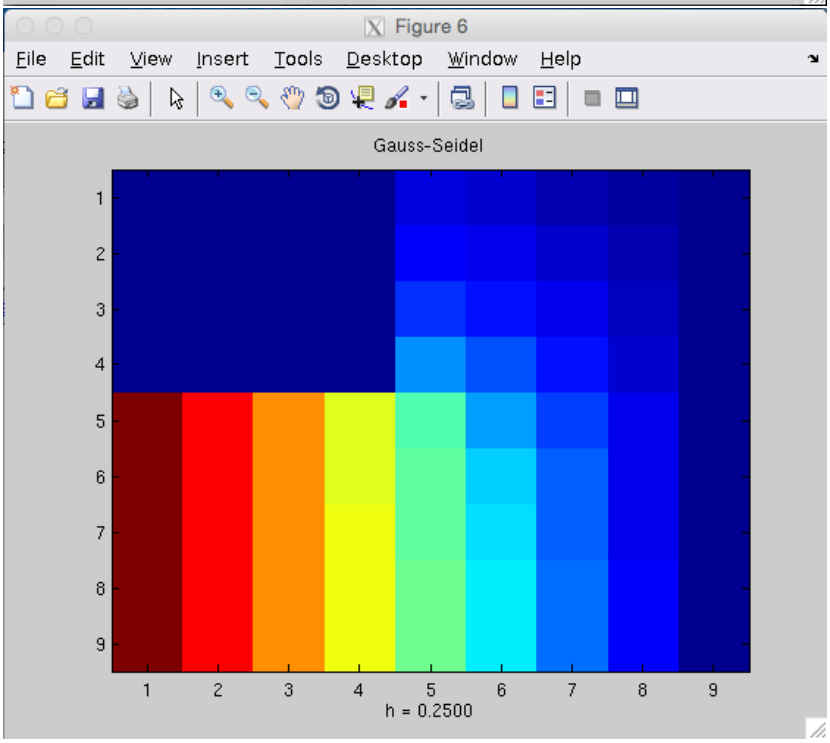


Figure 6
File Edit View Insert Tools Desktop Window Help
Gauss-Seidel
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
h = 0.2500

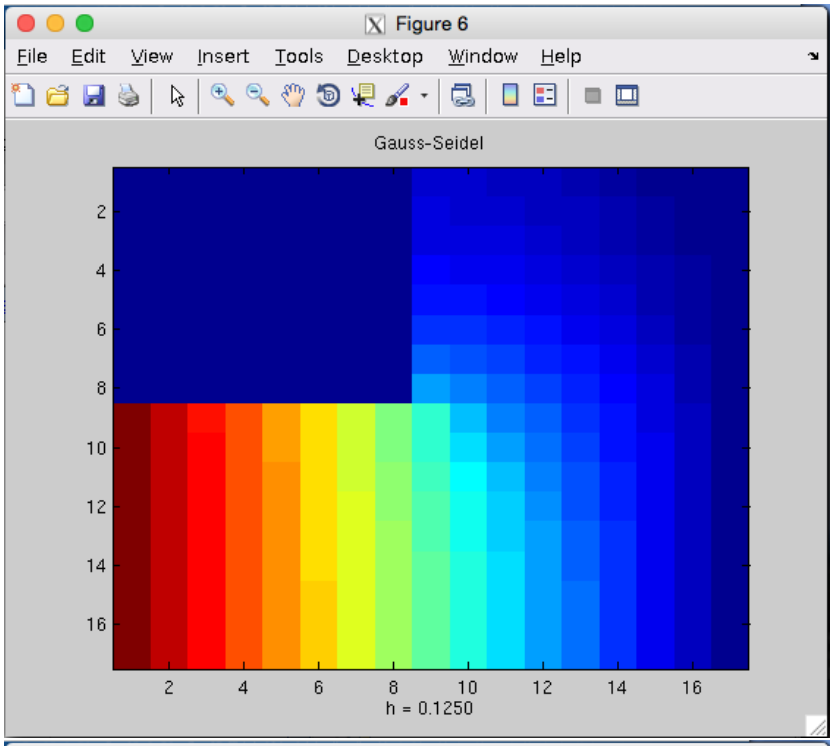


Figure 6
File Edit View Insert Tools Desktop Window Help
Gauss-Seidel
2
4
6
8
10
12
14
16
2 4 6 8 10 12 14 16
h = 0.1250

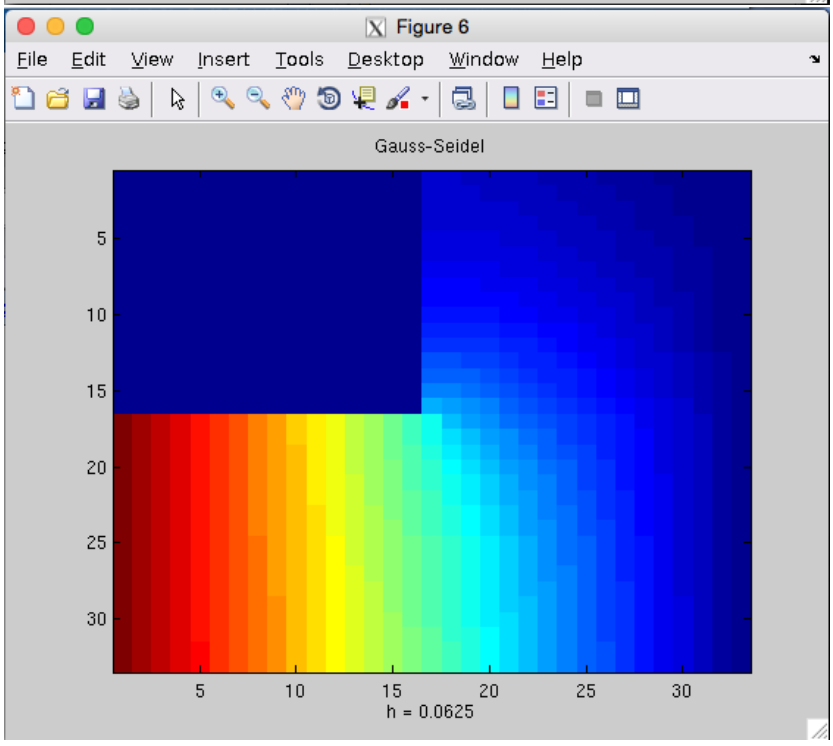
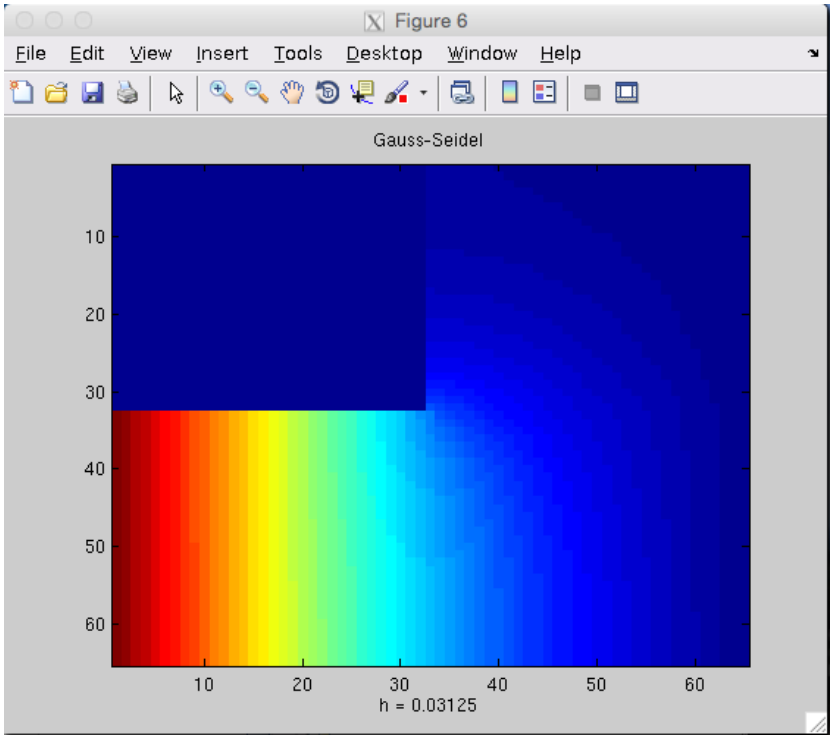


Figure 6
File Edit View Insert Tools Desktop Window Help
Gauss-Seidel
5
10
15
20
25
30
5 10 15 20 25 30
h = 0.0625



I also kept track of the CPU time for each of these methods for every value of h shown in the table below. We can see that for every method as the number of elements increases so does the CPU time. We obtain our best results as the number of objects increases, but the computation time takes too long for any smaller values of h than were tested in this experiment. Out of the three solvers that we tested, the Biconjugate Gradient solver was overall the fasted method and has the most visually pleasing solution.

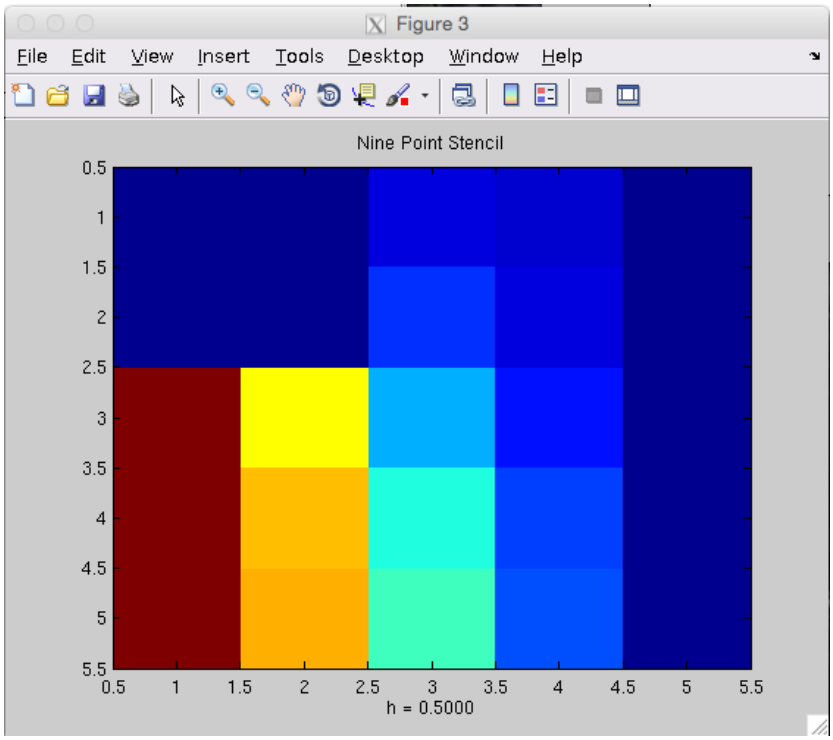
Solver	Cputime	h
BackSlash	0.22	0.5
	8.54	0.25
	3.74	0.125
	8.2	0.0625
	50.56	0.03125
Bicg	0.01	0.5
	0.01	0.25
	1.3	0.125
	33.42	0.0625
	1781.4	0.03125
Jacobi	0	0.5
	0.01	0.25
	7.35	0.125
	213.05	0.0625
	6633	0.03125
Gauss-Seidel	0.04	0.5
	0.15	0.25
	7.94	0.125
	48.68	0.0625
	1139.0	0.03125

4 Reworking Code to Solve a Linear System

Rework your code so that it now uses the nine-point finite difference approximation for the Laplacian operator. Solve the previous problem (from end-to-end) using several different values of h. Again, graph the results. Compare the results from the nine-point and five-point difference formulas in terms of accuracy and computational costs.

In modifying my code into a nine-point stencil, different weights than the five-point

stencil were applied to the current node and the eight surrounding nodes. The corner of the L also had to be slightly modified. The A and b matrices were built in the same fashion as the five-point stencil. After A and b were constructed, I used the backslash operator in MATLAB to quickly solve this linear system and visualize my results. The results were visualized for five different values of h shown in the following figures. The code to create the stencil follows.



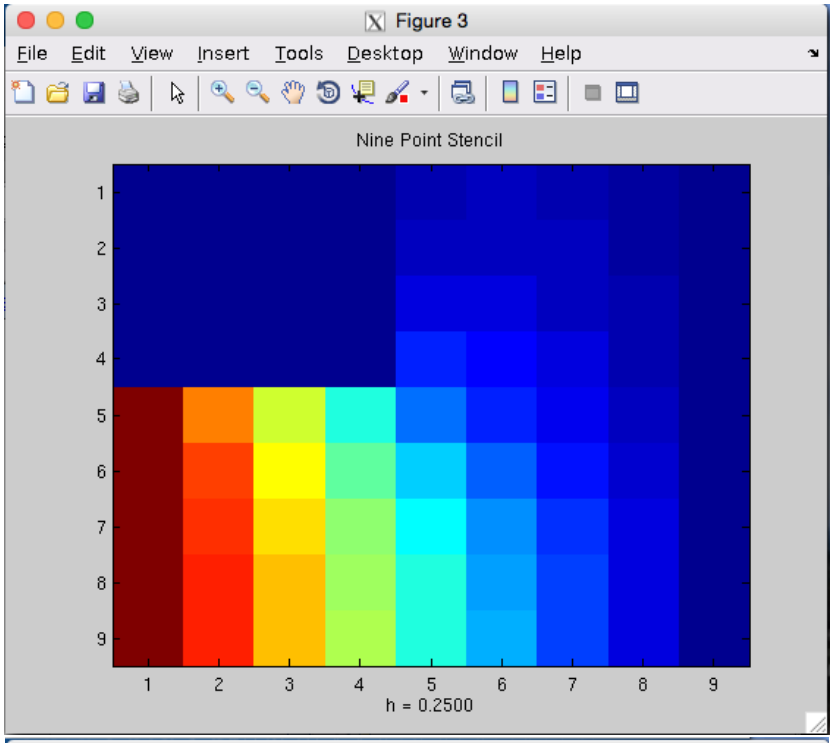


Figure 3
File Edit View Insert Tools Desktop Window Help
Nine Point Stencil
1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9
h = 0.2500

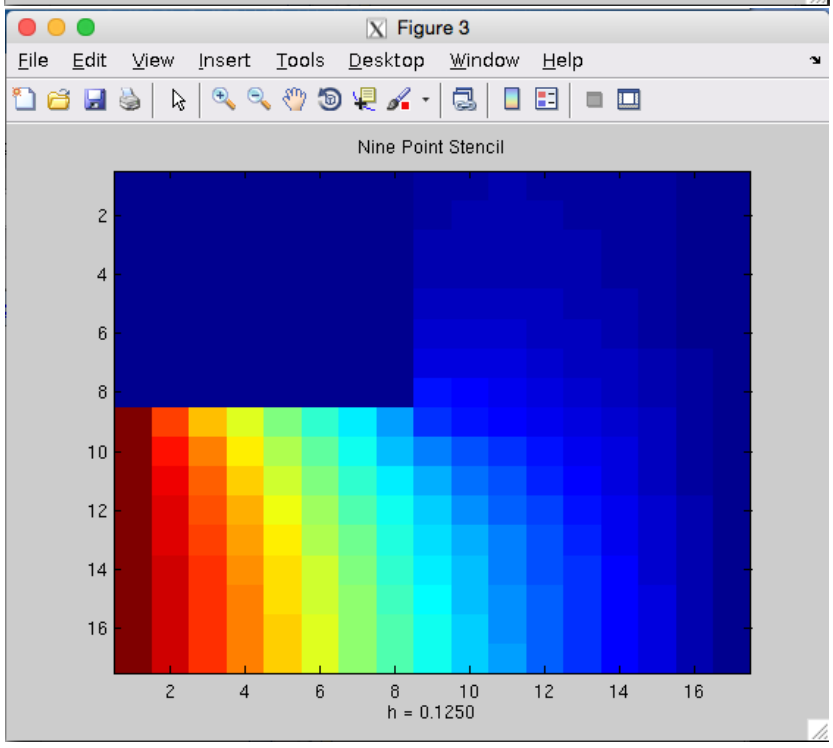


Figure 3
File Edit View Insert Tools Desktop Window Help
Nine Point Stencil
2 4 6 8 10 12 14 16
2 4 6 8 10 12 14 16
h = 0.1250

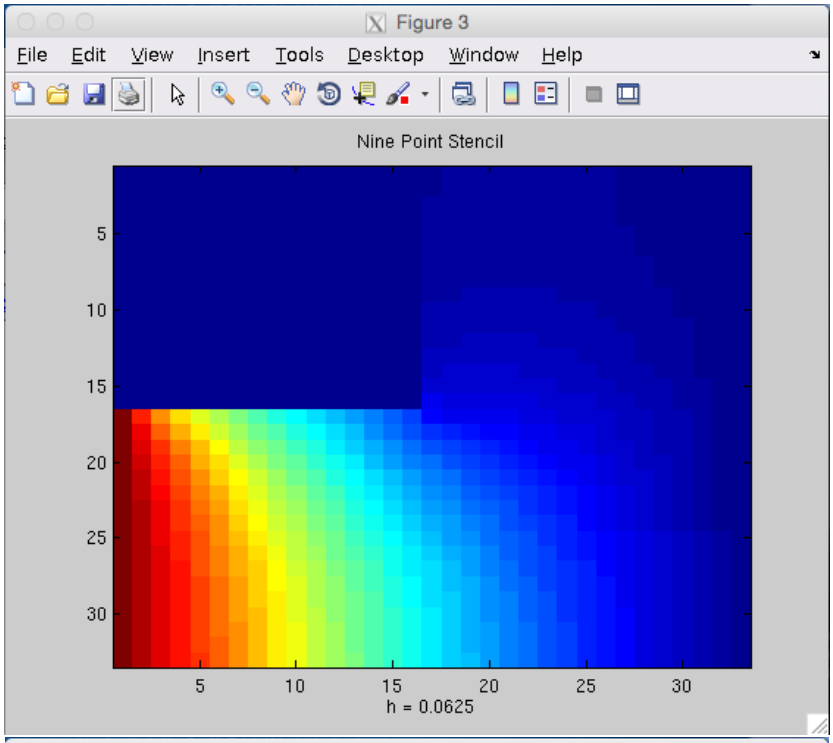


Figure 3
File Edit View Insert Tools Desktop Window Help
Nine Point Stencil
5
10
15
20
25
30
5 10 15 20 25 30
h = 0.0625

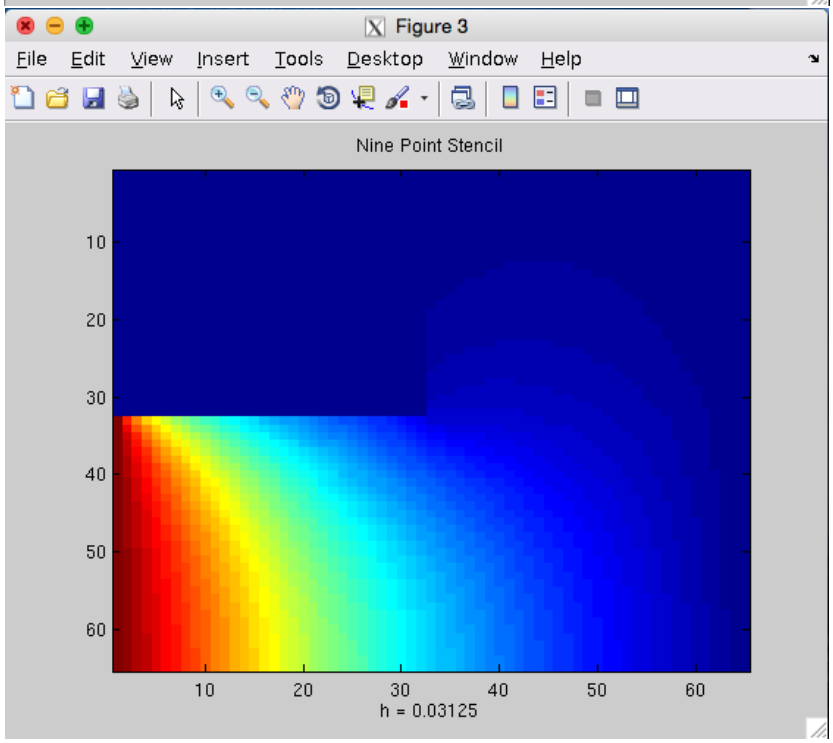


Figure 3
File Edit View Insert Tools Desktop Window Help
Nine Point Stencil
10
20
30
40
50
60
10 20 30 40 50 60
h = 0.03125

```
function [A,b,e] = matrix9(hIn)
%Converts the grid information into a matrix A and b.
t = cputime;

%Gets the spatial info from the Grid function.
[xPosArr,gridSize] = makeGrid(hIn);

%Initialize A and b matrices.
A = zeros(length(xPosArr));
b = zeros(length(xPosArr),1);

inx = 1;

%Makes the A and b arrays for the bottom half of the grid.
%Different conditions based on where you are in the space.
for i = 1:ceil(gridSize/2)
    for j = 1:gridSize
        if xPosArr(inx) == 0
            b(inx) = 1;
            A(inx,inx) = 1;
        elseif xPosArr(inx) == 2
            A(inx,inx) = 1;
        elseif i == 1
            A(inx,inx) = -20;
            A(inx,inx-1) = 4;
            A(inx,inx+1) = 4;
            A(inx,inx+gridSize) = 8;
            A(inx,inx+(gridSize+1)) = 2;
            A(inx,inx+(gridSize-1)) = 2;
        elseif xPosArr(inx) < 1 && i == ceil(gridSize/2)
            A(inx,inx) = -20;
            A(inx,inx+1) = 4;
            A(inx,inx-1) = 4;
            A(inx,inx-gridSize) = 8;
            A(inx,inx-(gridSize+1)) = 1;
            A(inx,inx-(gridSize-1)) = 2;
            A(inx,inx+(gridSize+1)) = 1;
        elseif xPosArr(inx) == 1 && i == ceil(gridSize/2)
            A(inx,inx) = -20;
            A(inx,inx+1) = 4;
            A(inx,inx-1) = 4;
            A(inx,inx+gridSize) = 4;
            A(inx,inx-gridSize) = 4;
            A(inx,inx-(gridSize+1)) = 1;
            A(inx,inx-(gridSize-1)) = 2;
            A(inx,inx+(gridSize+1)) = 1;
        else
            A(inx,inx) = -20;
            A(inx,inx-1) = 4;
            A(inx,inx+1) = 4;
            A(inx,inx+gridSize) = 4;
            A(inx,inx-gridSize) = 4;
            A(inx,inx+(gridSize-1)) = 1;
            A(inx,inx+(gridSize+1)) = 1;
            A(inx,inx-(gridSize-1)) = 1;
            A(inx,inx-(gridSize+1)) = 1;
        end
        inx = inx + 1;
    end
end
```

```
%Makes the A and b arrays for the top half of the grid.
for i = ceil(gridSize/2)+1:gridSize
    for j = ceil(gridSize/2):gridSize
        if xPosArr(inx) == 2
            A(inx,inx) = 1;
        elseif xPosArr(inx) == 1 && i == gridSize
            A(inx,inx) = -20;
            A(inx,inx+1) = 8;
            A(inx,inx-ceil(gridSize/2)) = 8;
            A(inx,inx-(ceil(gridSize/2)+1)) = 4;
        elseif xPosArr(inx) == 1
            A(inx,inx) = -20;
            A(inx,inx+1) = 8;
            A(inx,inx-ceil(gridSize/2)) = 4;
            A(inx,inx+ceil(gridSize/2)) = 4;
            A(inx,inx-(ceil(gridSize/2)+1)) = 1;
            A(inx,inx-(ceil(gridSize/2)-1)) = 1;
            A(inx,inx+(ceil(gridSize/2)+1)) = 2;
        elseif i == gridSize
            A(inx,inx) = -20;
            A(inx,inx+1) = 4;
            A(inx,inx-1) = 4;
            A(inx,inx-ceil(gridSize/2)) = 8;
            A(inx,inx-(ceil(gridSize/2)+1)) = 2;
            A(inx,inx-(ceil(gridSize/2)-1)) = 2;
        else
            A(inx,inx) = -20;
            A(inx,inx-1) = 4;
            A(inx,inx+1) = 4;
            A(inx,inx+ceil(gridSize/2)) = 4;
            A(inx,inx-ceil(gridSize/2)) = 4;
            A(inx,inx+(ceil(gridSize/2)+1)) = 1;
            A(inx,inx+(ceil(gridSize/2)-1)) = 1;
            A(inx,inx-(ceil(gridSize/2)+1)) = 1;
            A(inx,inx-(ceil(gridSize/2)-1)) = 1;
        end
        inx = inx + 1;
    end
end

e = cputime - t;

end
```

I also kept track of the CPU time for both the five-point and the nine-point stencil for every value of h shown in the table below. These results were surprising in the beginning because the nine-point stencil is faster than the five-point, but as the number of elements increases the CPU time between the two methods gets closer together. I would assume that there is a certain number of elements were the five-point stencil is consistently faster than the nine-point stencil. We can see from the images that the nine-point stencil is a better solution.

Stencil	Cputime	h
Five-point	0.22	0.5
	8.54	0.25
	3.74	0.125
	8.2	0.0625
	50.56	0.03125
NinePoint	0.11	0.5
	0.52	0.25
	2.36	0.125
	7.19	0.0625
	50.97	0.03125

*All of the code for this assignment is included in the submission.