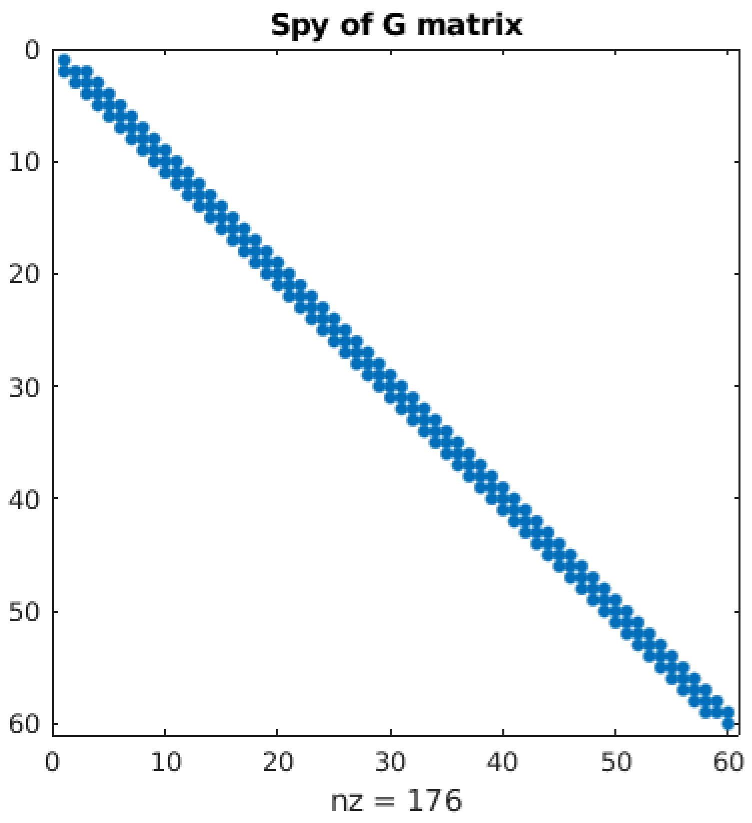
Table of Contents

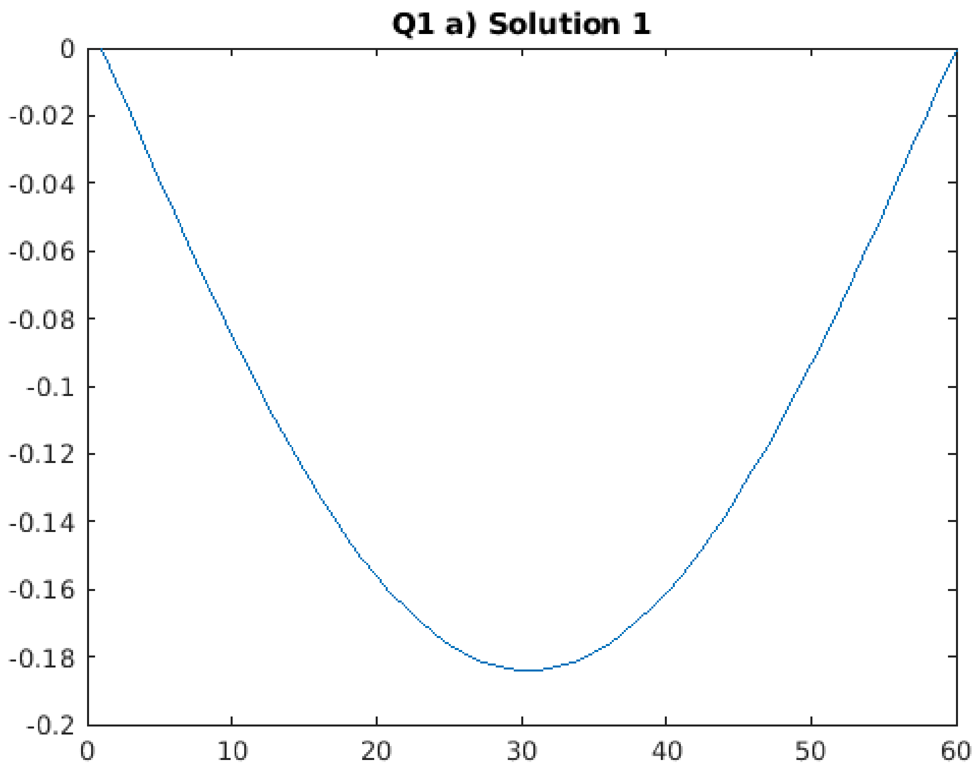
```
% setup
clearvars
clearvars -GLOBAL
close all
```

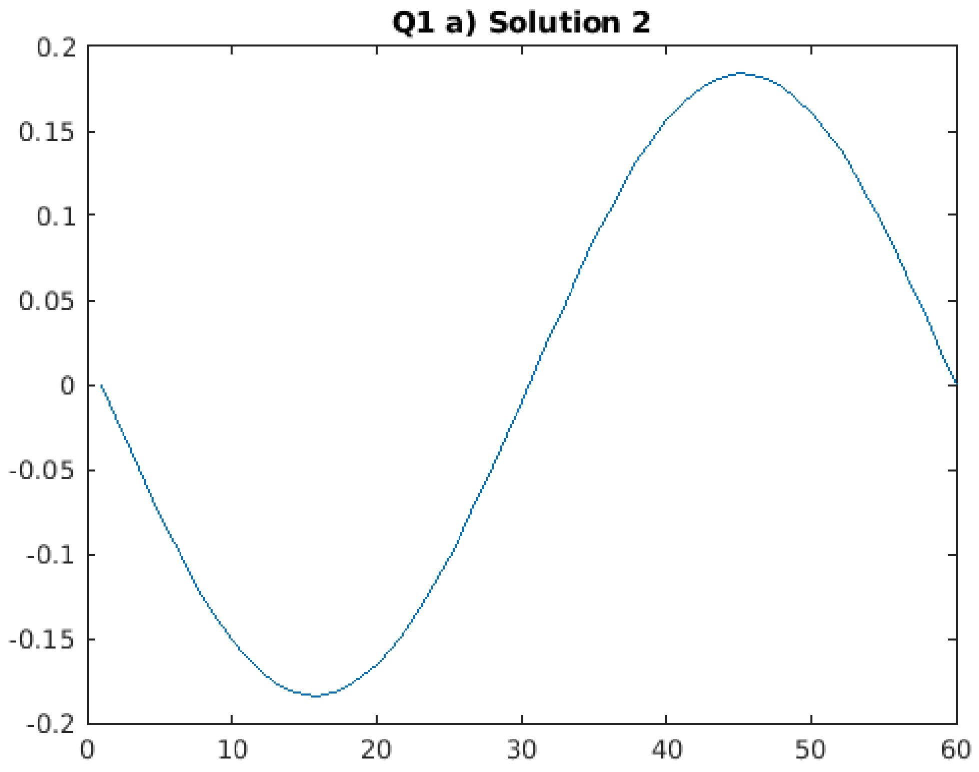
Question 1 a)

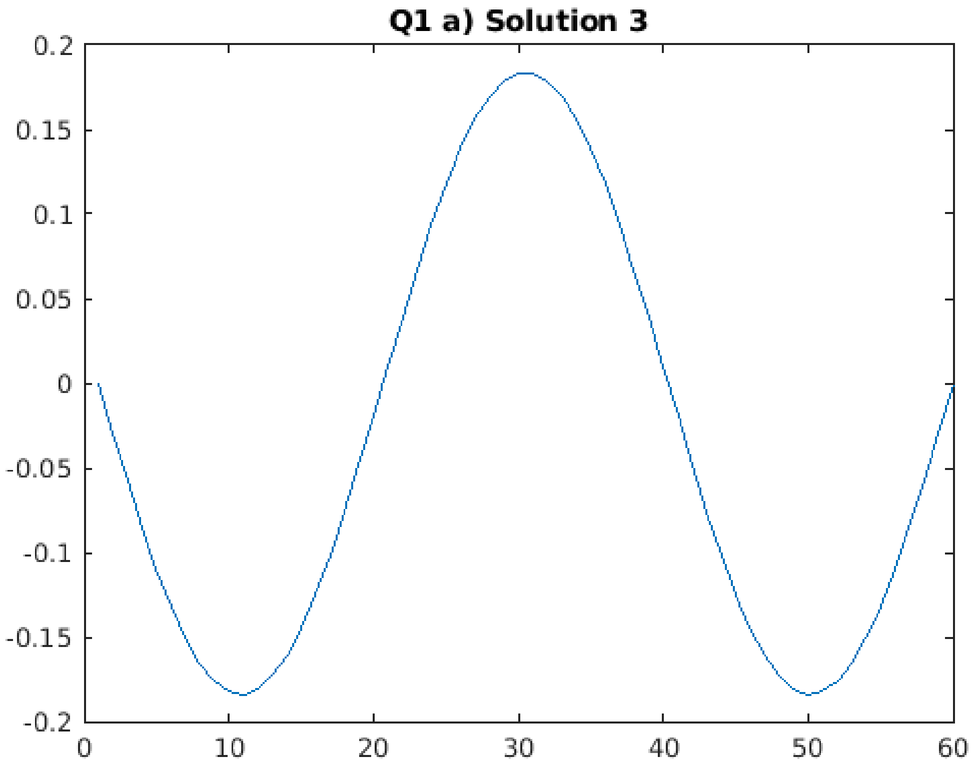
For 1 a), I will be focusing on the 1D case. So width will not be considered. Because of this, the G matrix will only have dimensions of nl by nl. This also changes the way the G matrix is populated compared to the EIPA. Since there is only one dimension being considered, the centermost n term will only have a -2. In addition to this nym and nyp also don't have to be considered. Note, setting one boundary to zero causes problems with the singularity of the matrix and breaks the eigs function. So to get around this, the boundary condition at x = L is instead set to something comparatively small (like 1) next to Vo (big, 100 in this case).

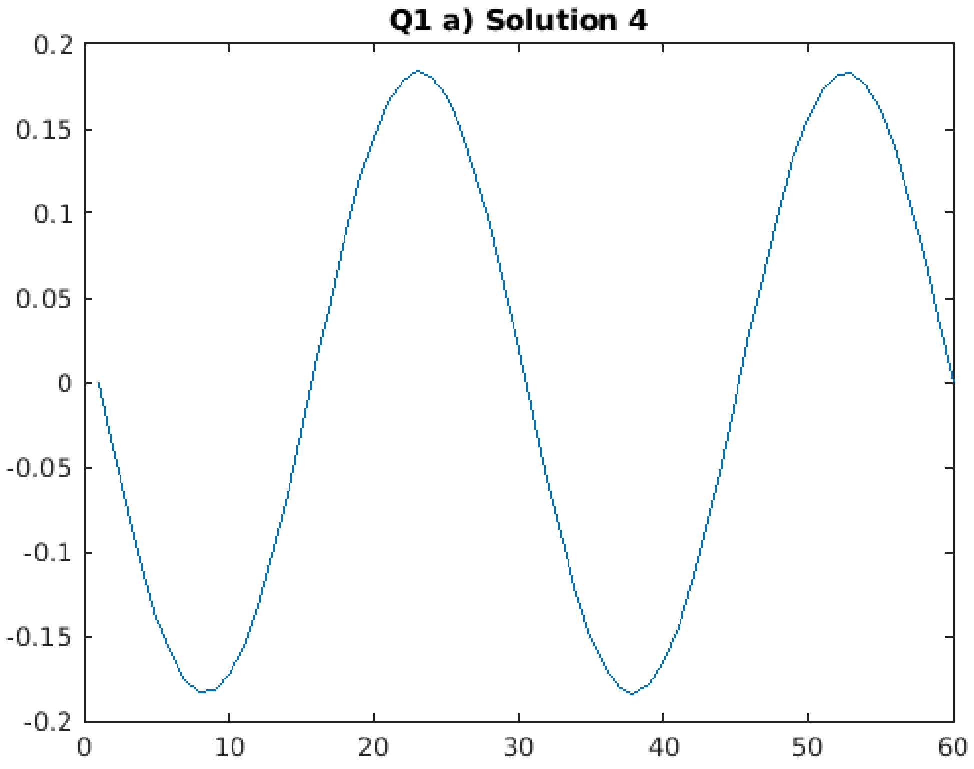
```
nw = 40;
nl = 60;
Vo = 100;
wanted_solutions = 4;

% initialize and populate the G matrix
G = zeros(nl, nl);
for i = 1:nl
    n = i;
    nxm = (i - 1);
    nxp = (i + 1);

    % check for left boundary condition
    if i == 1
        G(n, n) = Vo;

    % check for right boundary condition
    elseif i == nl
        G(n, n) = 1;

    % not at boundary, populate matrix normally
    else
        G(n, n) = -2;
        G(n, nxm) = 1;
        G(n, nxp) = 1;
    end
end

% make sure we have a "diagonal-ish" G matrix and call eigs
figure
spy(G)
title('Spy of G matrix')
[E, D] = eigs(G, wanted_solutions, 'SM');

% loop through solutions and plot them
for solution = 1:wanted_solutions
    map = zeros(nl,1);
    for count = 1:nl
        map(count) = E(count, solution);
    end
    figure
    plot(1:nl, map)
    title(sprintf('Q1 a) Solution %d', solution))
end
```




Q1 a) Solution 1
0
-0.02
-0.04
-0.06
-0.08
-0.1
-0.12
-0.14
-0.16
-0.18
-0.2
0
10
20
30
40
50
60


Q1 a) Solution 2
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
10
20
30
40
50
60


Q1 a) Solution 3
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
10
20
30
40
50
60


Q1 a) Solution 4
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
0
10
20
30
40
50
60

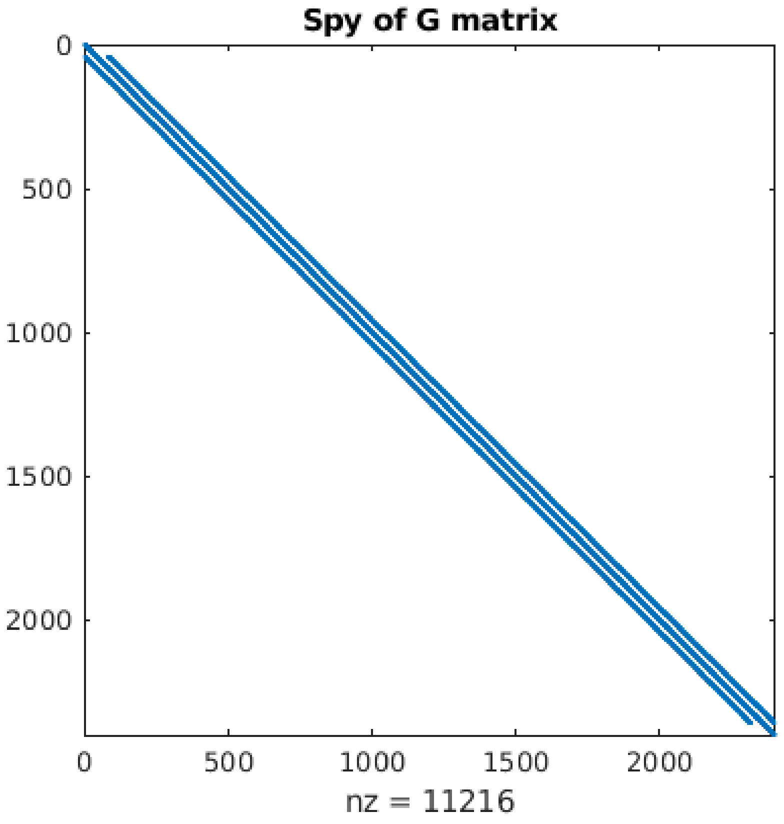
Question 1 b)

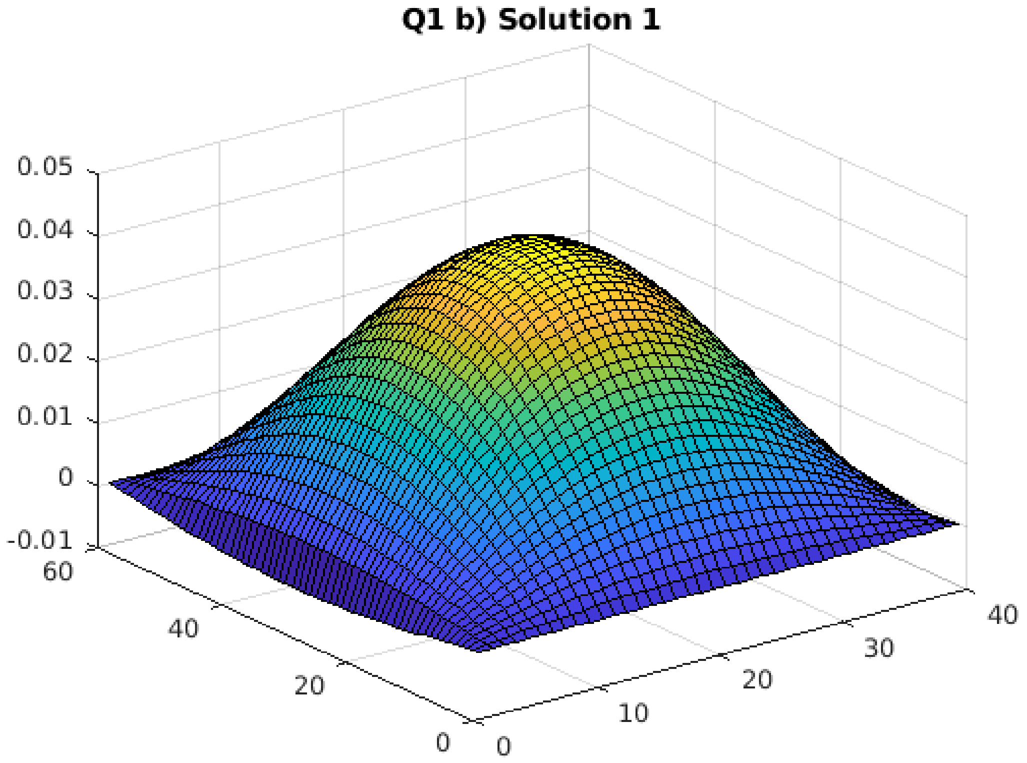
The following snippet solves for the case where V = Vo (set to 100) at x = 0, and V = 0 at x = L. Note, actually setting a boundary condition to zero causes problems for the eigs function like in 1 a). This part is pretty similar to a), the only difference being we have two dimesnions like the EIPA. So we now have a nested for loop populating the G matrix with nyp and nym now being considered.

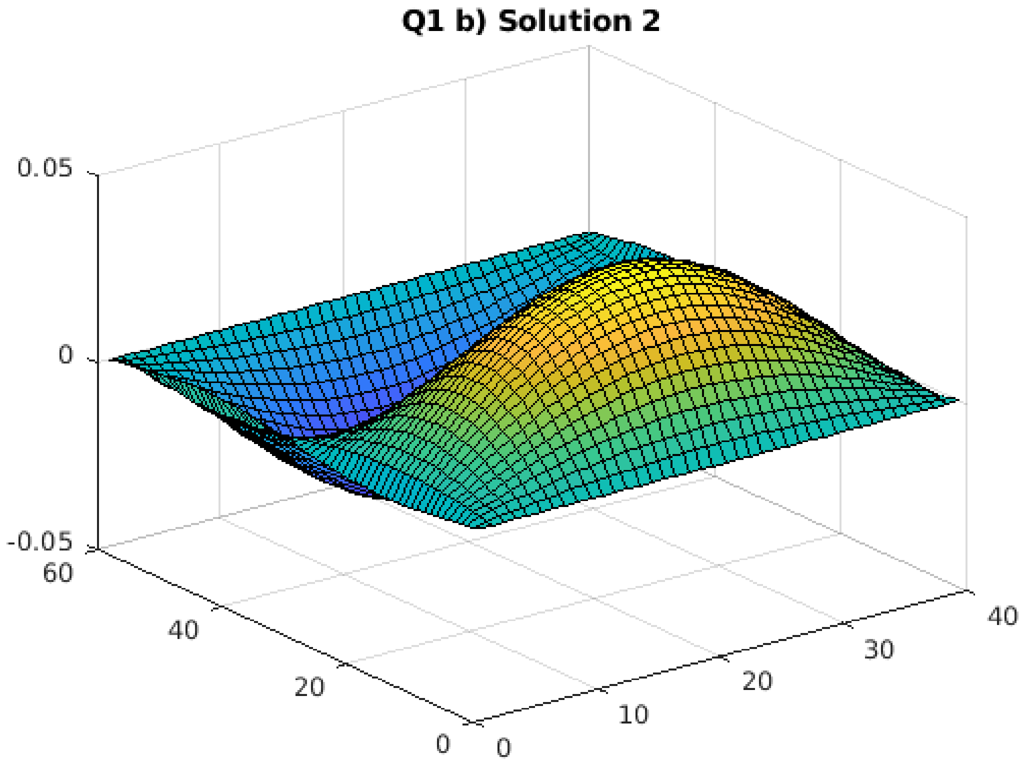
Comparison with Analytical Series: I have included the code to populate a voltage matrix based of the analytical equation, and included a 3D plot, but I don't feel the results make sense. There isn't much to compare. Because of the sin(n*pi*y/a) term, this will always be zero. Since for the first solution n = 1, and the step size, a = 1, the argument is integers of pi, making it always zero. I might be misunderstanding how the series works, but populating a matrix requires feeding this integers. Investigating the matrix with a breakpoint shows that values are making it in to the matrix. Stepping through the formula, it appears to be because cosh generates massive numbers, which should be multiplied by zero, but are instead multiplied by the sine of Matlab's rounding error on pi. This is super cool, but not relevant at all to the assignment.

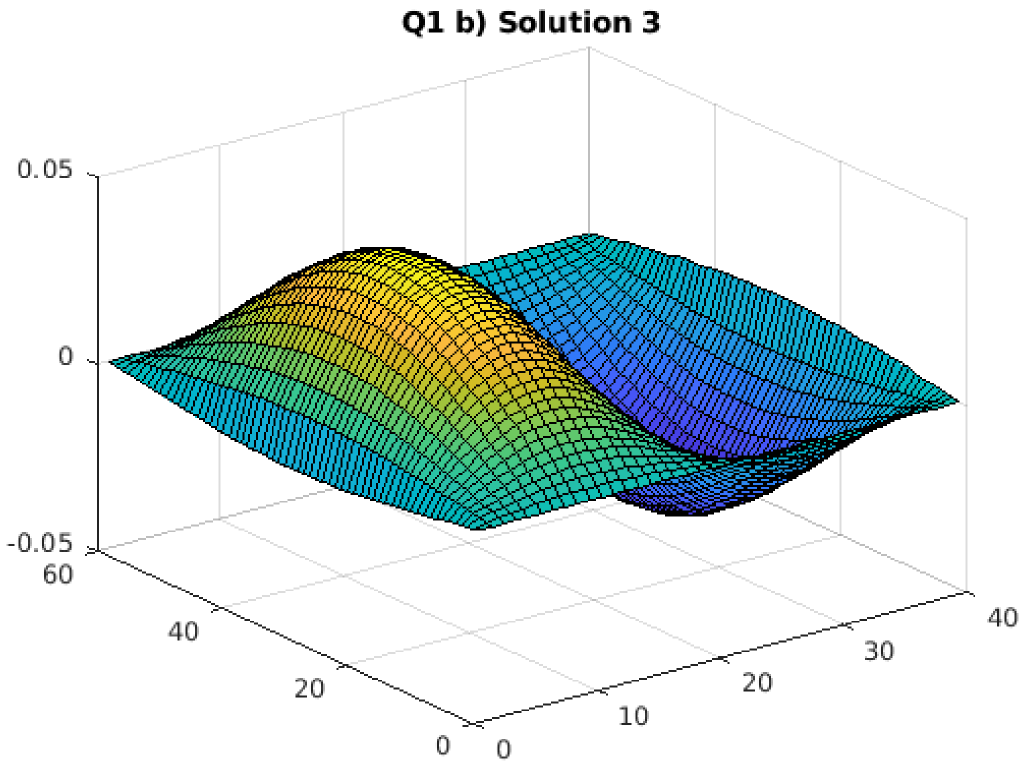
```
% initialize and populate the G matrix
G = zeros(nl*nw, nl*nw);
for i = 1:nl
    for j = 1:nw
        n = j + (i - 1)*nw;
        nxm = j + (i - 2)*nw;
        nxp = j + i*nw;
        nym = (j - 1) + (i - 1)*nw;
        nyp = (j + 1) + (i - 1)*nw;

        % check if we are at a boundary, setting to zero breaks eigs
 so we get as close as we can
        if i == 1
            G(n, n) = Vo;
        elseif i == nl
            G(n, n) = 1;
        elseif j == 1 || j == nw
            G(n, n) = 1;

        % not at a boundary, populate G matrix
        else
            G(n, n) = -4;
            G(n, nxm) = 1;
            G(n, nxp) = 1;
            G(nyp, n) = 1;
            G(nym, n) = 1;
        end
    end
end

% check if diagonal-ish and call eigs
figure
spy(G)
title('Spy of G matrix')
[E, D] = eigs(G, wanted_solutions, 'SM');

% loop though solutions and plot them (surface this time)
```

```
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
    for count = 1:nw*nl
        map(i, j) = E(count, solution);
        if j == nw
            j = 1;
            i = i + 1;
        else
            j = j + 1;
        end
    end
    figure
    surf(map)
    title(sprintf('Q1 b) Solution %d', solution))
    axis([0 nw 0 nl])
end

% populate matrix with analytical solution
% I will compare the first valid solution, n = 1, step size a = b = 1
V = zeros(nl, nw);
for i = 1:nl
    for j = 1:nw
        V(i, j) = (4 * Vo / pi) * (cosh(pi * i)/cosh(pi)) * sin(pi *
 j);
    end
end

figure
surf(V)
title('Plot of Analytical Solution for 1 b)')
```

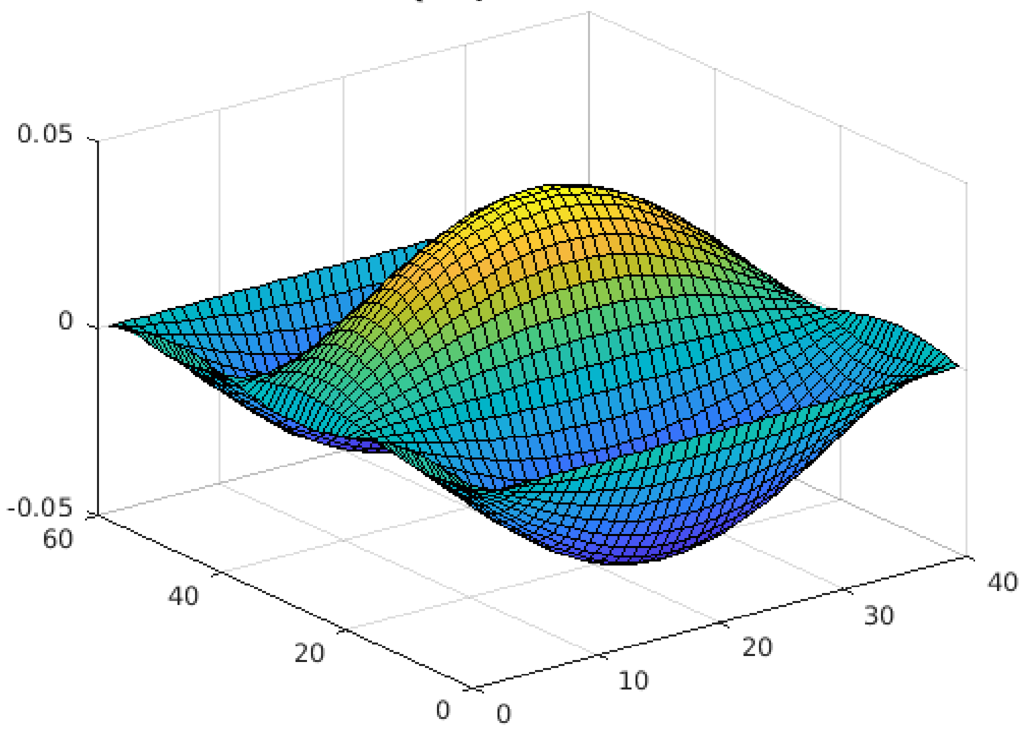

Spy of G matrix
0
500
1000
1500
2000
0
500
1000
1500
2000
nz = 11216


Q1 b) Solution 1
0.05
0.04
0.03
0.02
0.01
0
-0.01
60
40
20
0
0
10
20
30
40

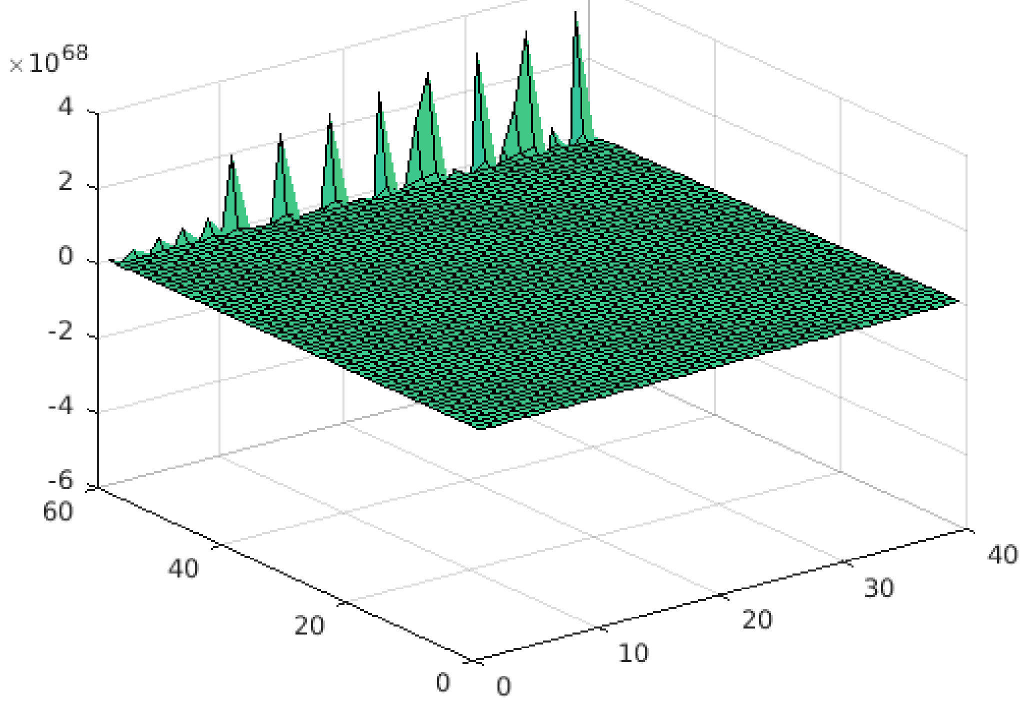

Q1 b) Solution 2
0.05
0
-0.05
60
40
20
0
0
10
20
30
40


Q1 b) Solution 3
0.05
0
-0.05
60
40
20
0
0
10
20
30
40

Q1 b) Solution 4



Plot of Analytical Solution for 1 b)

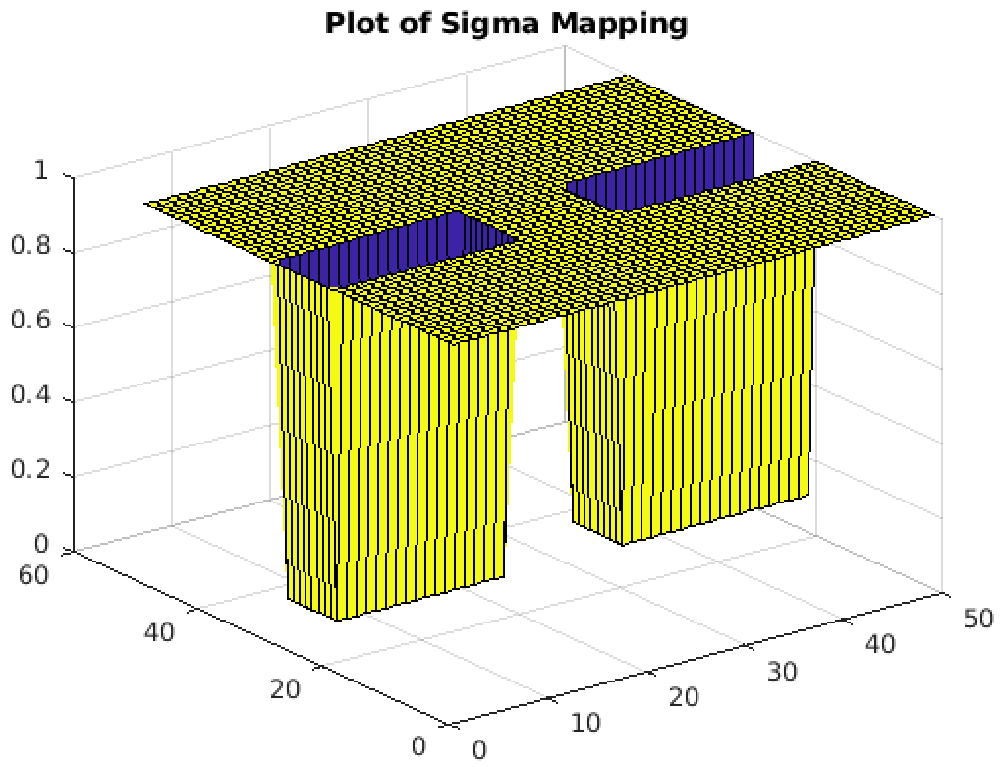


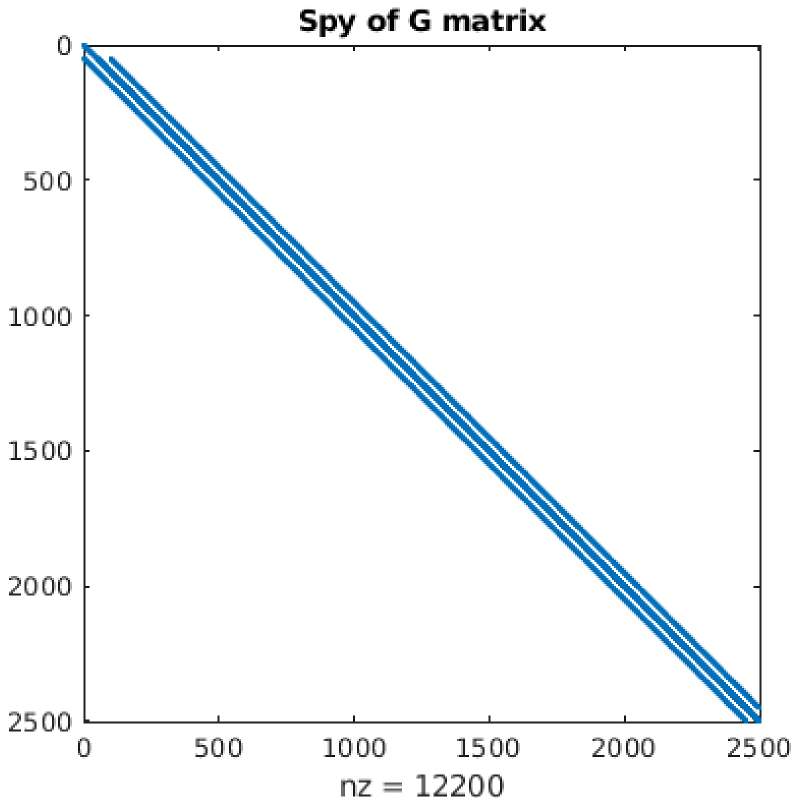
Question 2 a)

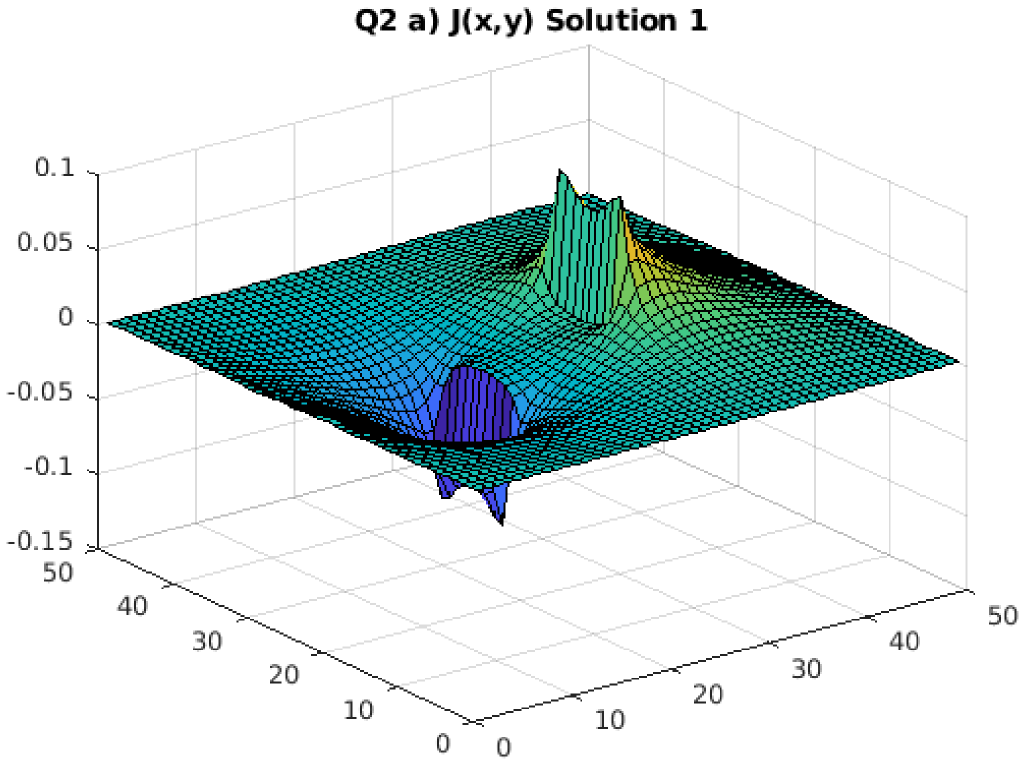
This problem investigates current flow through an area with a bottleneck by using the same finite difference method used in Q1, and applying different conductivities to boxes within the area (1 for outside the box, 10^-2 for inside). To model the current flow, a voltage was assumed to be applied to one side of the area. I decided to set Vo = 10 at x = 0. The boxes with lower conductivity are set for 20 < x < 30, and both 0 < y < 20, and 30 < y < W, where L and W are the length and width of the area respectively and were chosen to both be 50 for simplicity. The following snippet sets up the problem with the constants required.

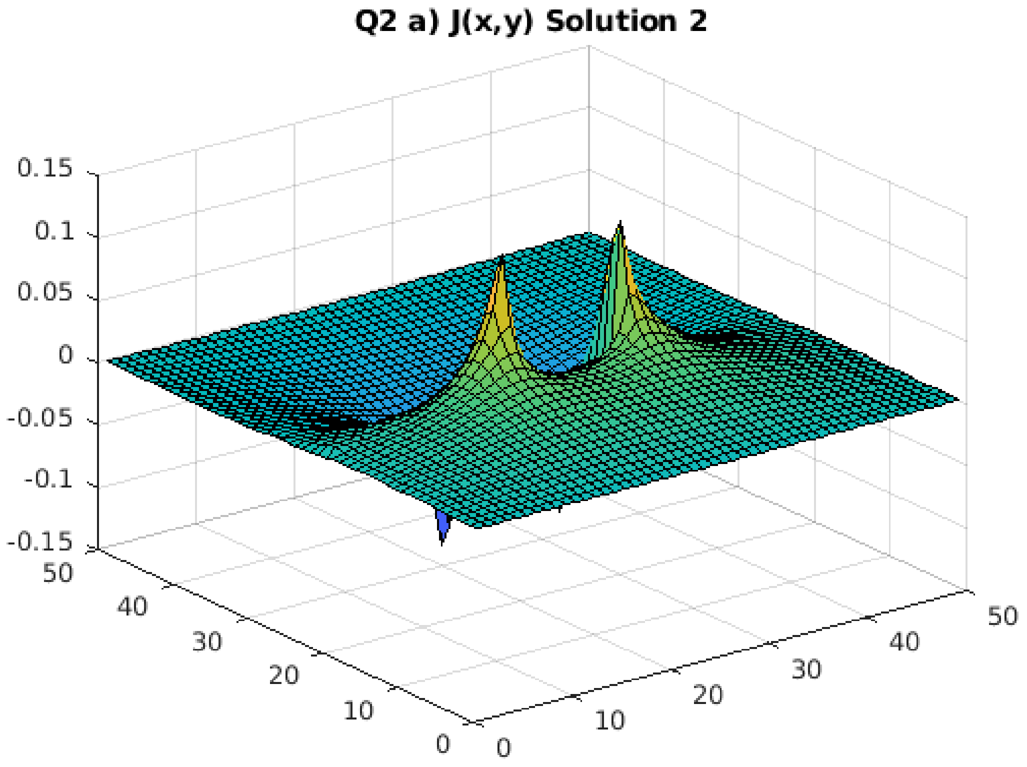
```
nw = 50;
nl = 50;
Vo = 10;
sigma_in = 10E-2;
sigma_out = 1;
wanted_solutions = 4;
sigma = zeros(nl, nw);

% build our sigma matrix and plot it
for i = 1:nl
    for j = 1:nw
        % check if inside box
        if (i > 20 && i < 30) && ((j > 1 && j < 20) || (j > 30))
            sigma(i, j) = sigma_in;
        % otherwise, we are outside the box
        else
            sigma(i, j) = sigma_out;
        end
    end
end
figure
surf(sigma)
title('Plot of Sigma Mapping')

% initialize and populate the G matrix
G = zeros(nl*nw, nl*nw);
for i = 1:nl
    for j = 1:nw
        n = j + (i - 1) * nw;
        nxm = j + (i - 2) * nw;
        nxp = j + i * nw;
        nyp = (j + 1) + (i - 1) * nw;
        nym = (j - 1) + (i - 1) * nw;

        % check if we are at the l = 0 case, apply boundary
        if i == 1
            G(n, n) = Vo;

        % we still need to check if we are at an edge to know which
        % terms contribute to G. Corners are most specific cases, we
 will
        % cover those first, except for x = 0 because that is tied to
 a
        % boundary condition and will always be Vo
        % top right corner, no nxp or nym
        elseif i == nl && j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % bottom right corner, no nxp or nyp
        elseif i == nl && j == nw
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);

        % now we cover the cases along the edge. We already know we
 aren't
        % at a corner at this point

        % along right edge, so no nxp term
        elseif i == nl
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % along top edge, no nym
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % along bottom edge, no nyp
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % otherwise, we're somewhere in the middle, all terms
 considered
        else
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        end
    end
end

% make sure we have the diagonal correct, and find eigen[vectors/
values]
figure
spy(G)
title('Spy of G matrix')
[E, D] = eigs(G, wanted_solutions, 'SM');

% plot and pray, starting with J(x,y)
```

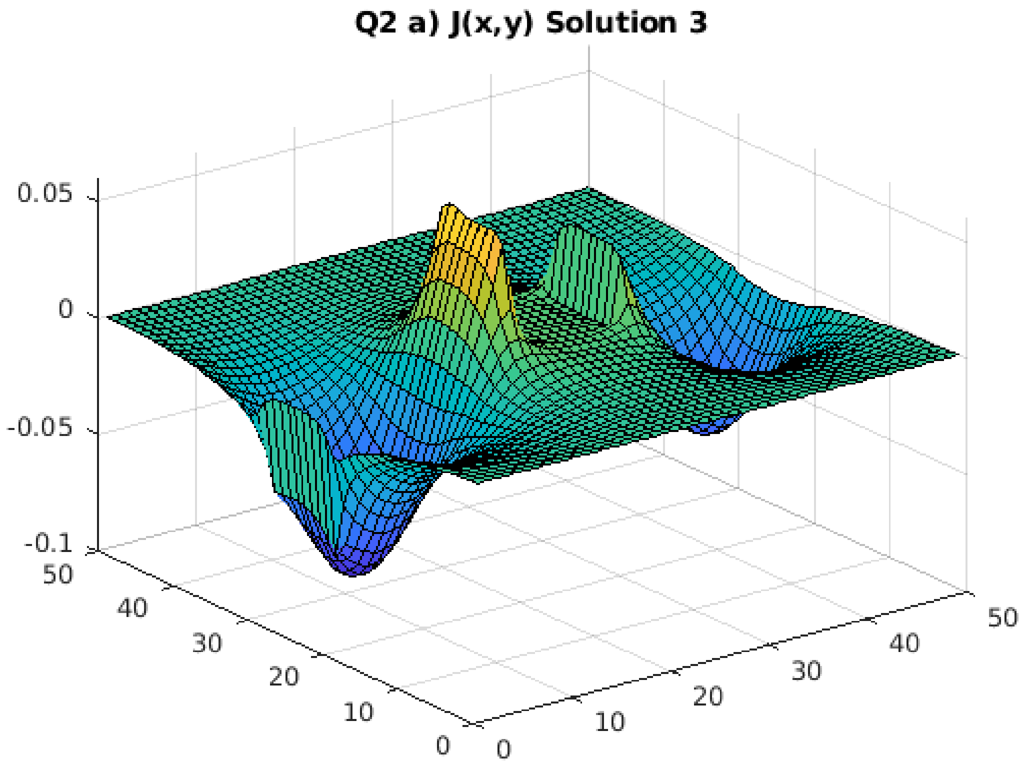
```
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
    for count = 1:nw*nl
        map(i, j) = E(count, solution);
        if j == nw
            j = 1;
            i = i + 1;
        else
            j = j + 1;
        end
    end
    figure
    surf(map)
    title(sprintf('Q2 a) J(x,y) Solution %d', solution))
    axis([0 nw 0 nl])
end

% plot and pray, the sequel: V(x,y)
% pretty much the same plots, but we divide by conductivity
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
    for count = 1:nw*nl
        map(i, j) = E(count, solution);
        if j == nw
            j = 1;
            i = i + 1;
        else
            j = j + 1;
        end
    end
    map = map ./ sigma;
    figure
    surf(map)
    title(sprintf('Q2 a) V(x,y) Solution %d', solution))
    axis([0 nw 0 nl])
end

% plot and pray, round three: E(x,y)
% pretty much the same plots, but multiply V(x, y) by a test charge
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
    for count = 1:nw*nl
        map(i, j) = E(count, solution);
        if j == nw
            j = 1;
            i = i + 1;
        else
            j = j + 1;
        end
    end
    map = map .* 1.602E-19;
    figure
    surf(map)
    title(sprintf('Q2 a) E(x,y) Solution %d', solution))
    axis([0 nw 0 nl])
end
```

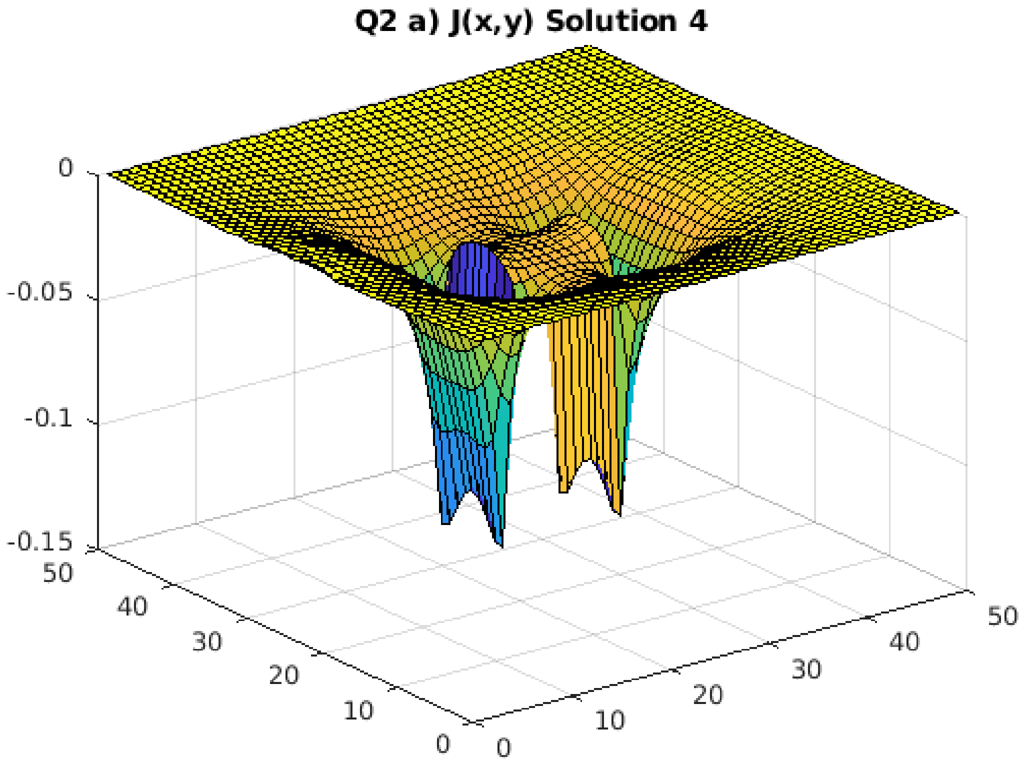


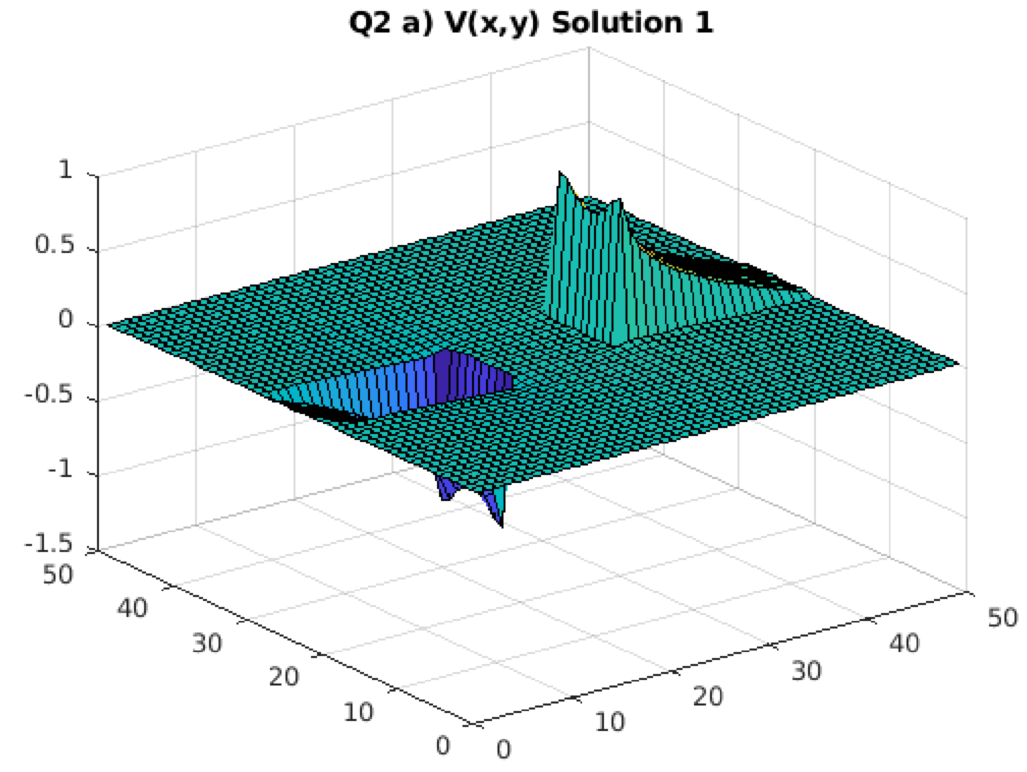

Spy of G matrix
0
500
1000
1500
2000
2500
0
500
1000
1500
2000
2500
nz = 12200

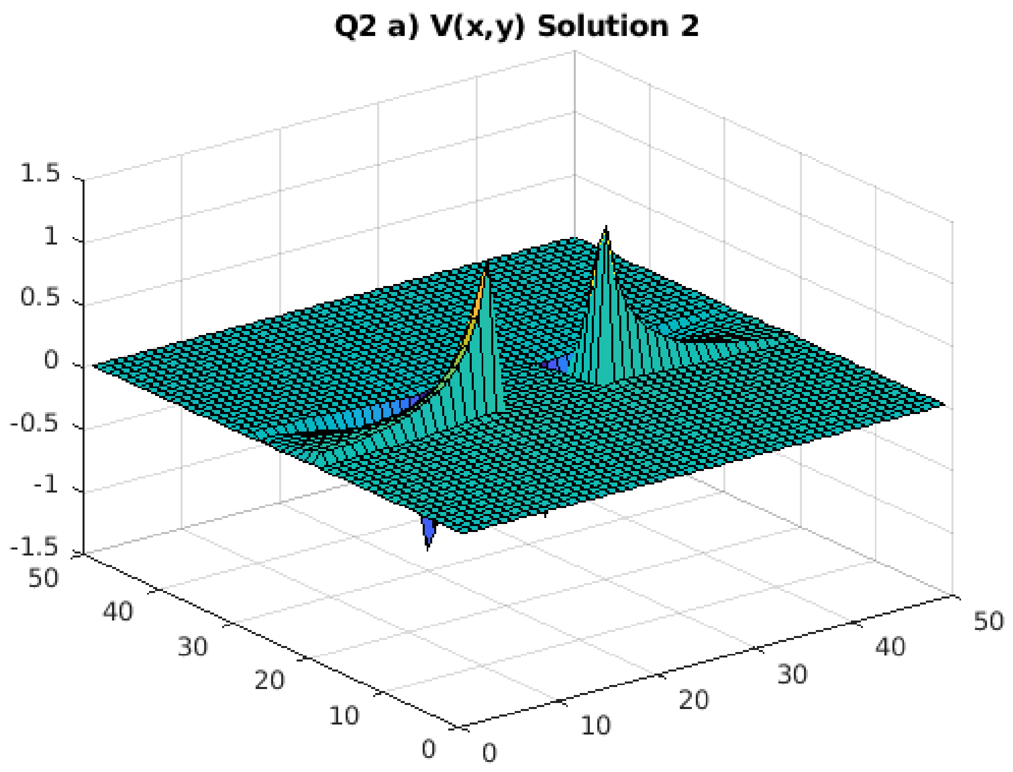

Q2 a) J(x,y) Solution 1
0.1
0.05
0
-0.05
-0.1
-0.15
50
40
30
20
10
0
0
10
20
30
40
50

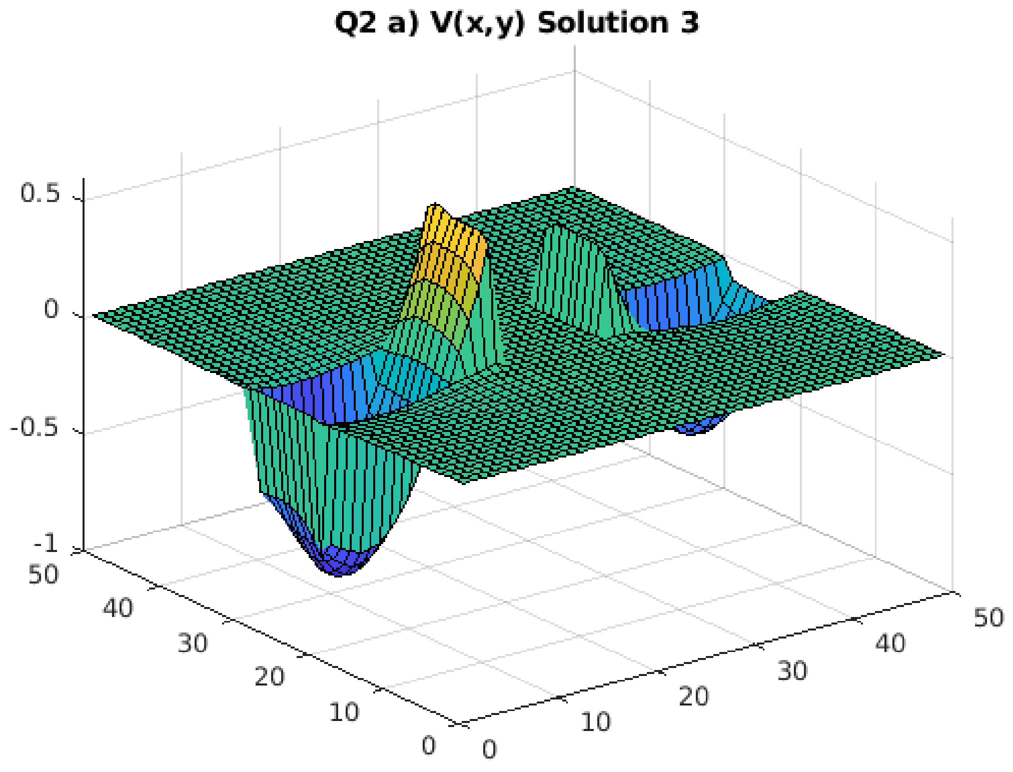

Q2 a) J(x,y) Solution 2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
50
40
30
20
10
0
0
10
20
30
40
50

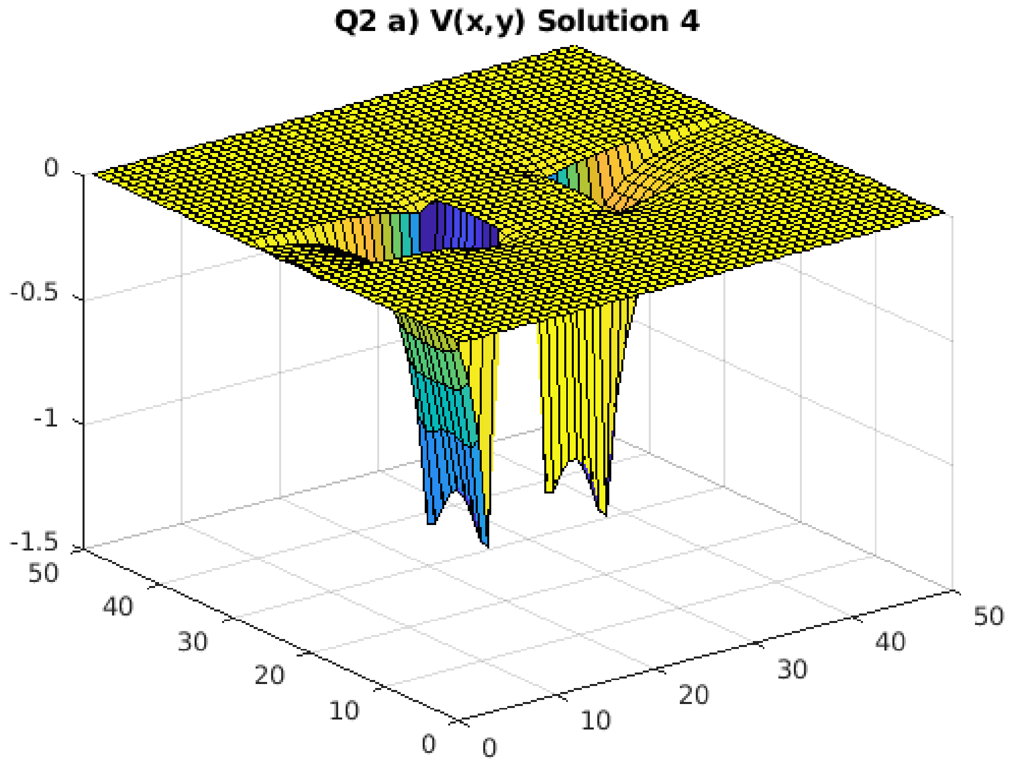

Q2 a) J(x,y) Solution 3
0.05
0
-0.05
-0.1
50
40
30
20
10
0
0
10
20
30
40
50


Q2 a) J(x,y) Solution 4
0
-0.05
-0.1
-0.15
50
40
30
20
10
0
0
10
20
30
40
50

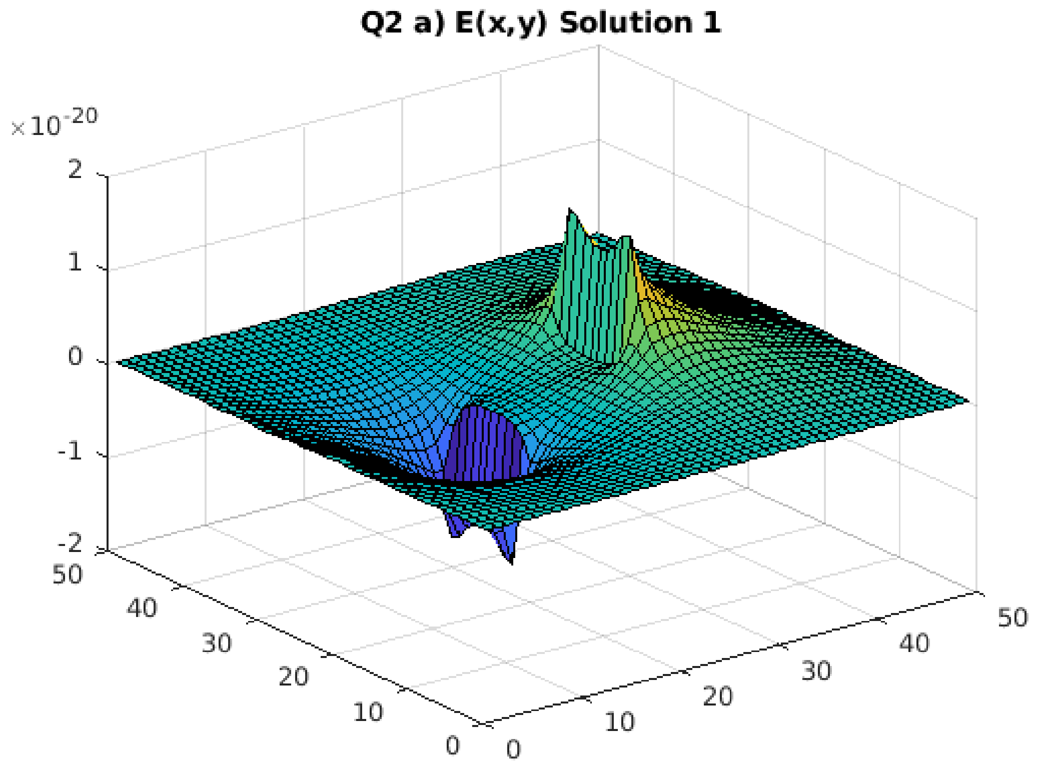

Q2 a) V(x,y) Solution 1
1
0.5
0
-0.5
-1
-1.5
50
40
30
20
10
0
0
10
20
30
40
50


Q2 a) V(x,y) Solution 2
1.5
1
0.5
0
-0.5
-1
-1.5
50
40
30
20
10
0
0
10
20
30
40
50

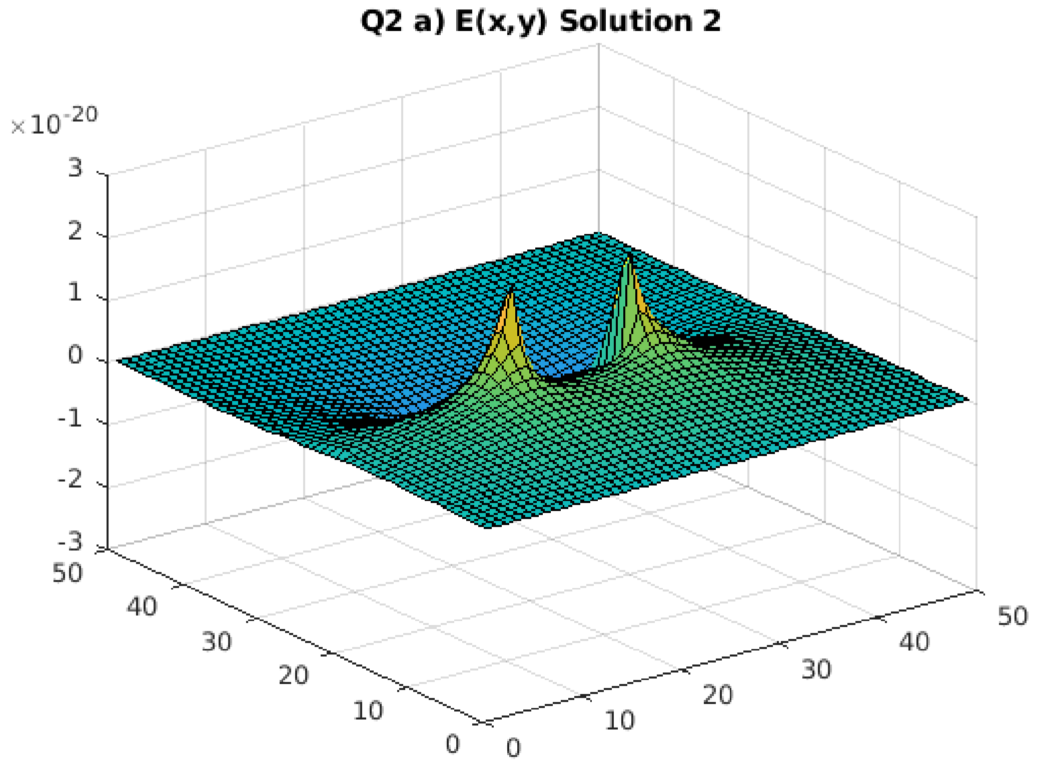

Q2 a) V(x,y) Solution 3
0.5
0
-0.5
-1
50
40
30
20
10
0
0
10
20
30
40
50

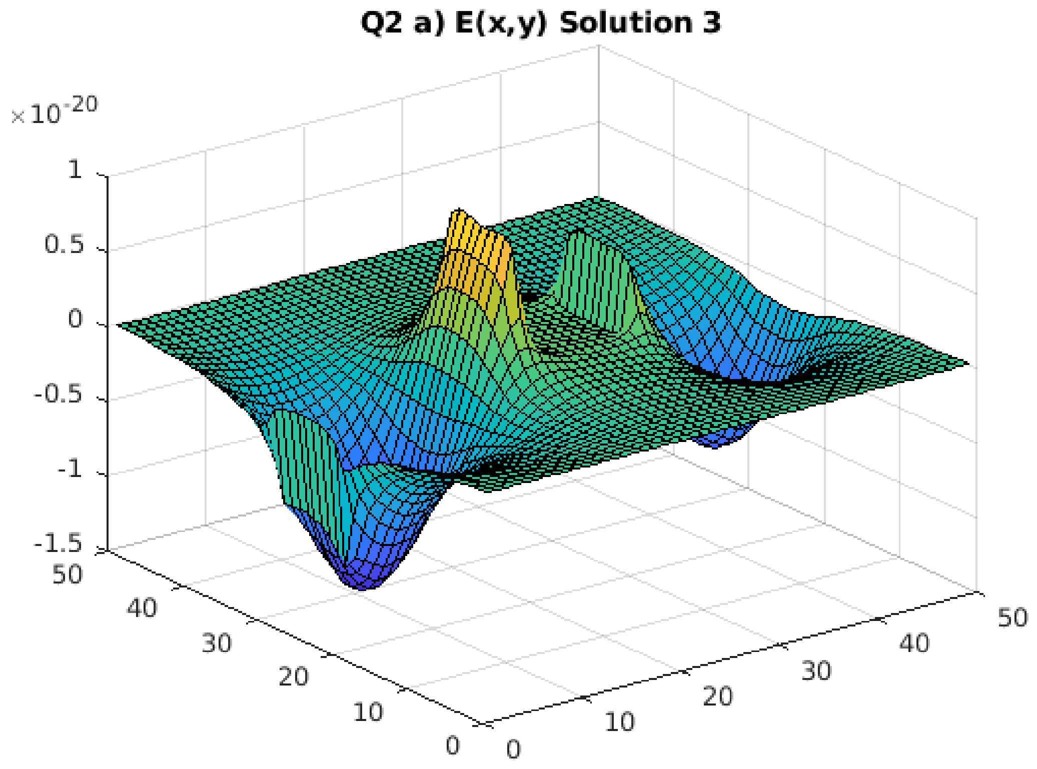


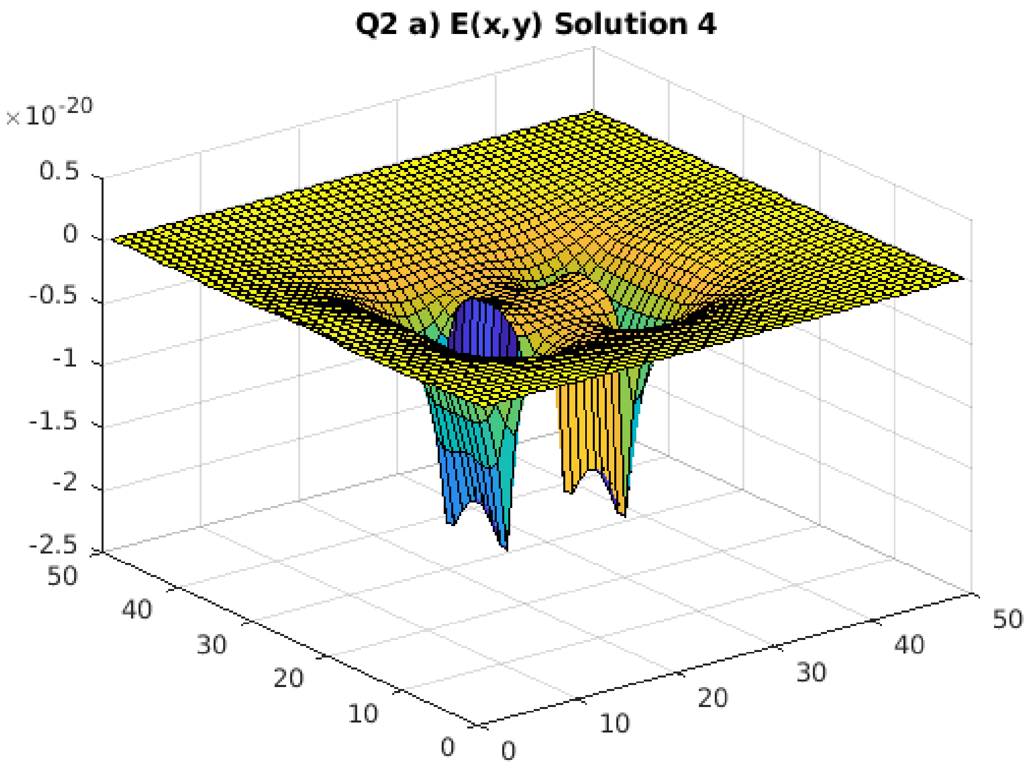
Q2 a) V(x,y) Solution 4
0
-0.5
-1
-1.5
50
40
30
20
10
0
0
10
20
30
40
50



Q2 a) E(x,y) Solution 1
$\times 10^{-20}$
2
1
0
-1
-2
50
40
30
20
10
0
0
10
20
30
40
50


Q2 a) E(x,y) Solution 2
×10^-20
3
2
1
0
-1
-2
-3
50
40
30
20
10
0
0
10
20
30
40
50


Q2 a) E(x,y) Solution 3
×10^-20
1
0.5
0
-0.5
-1
-1.5
50
40
30
20
10
0
0
10
20
30
40
50



Question 2 b)

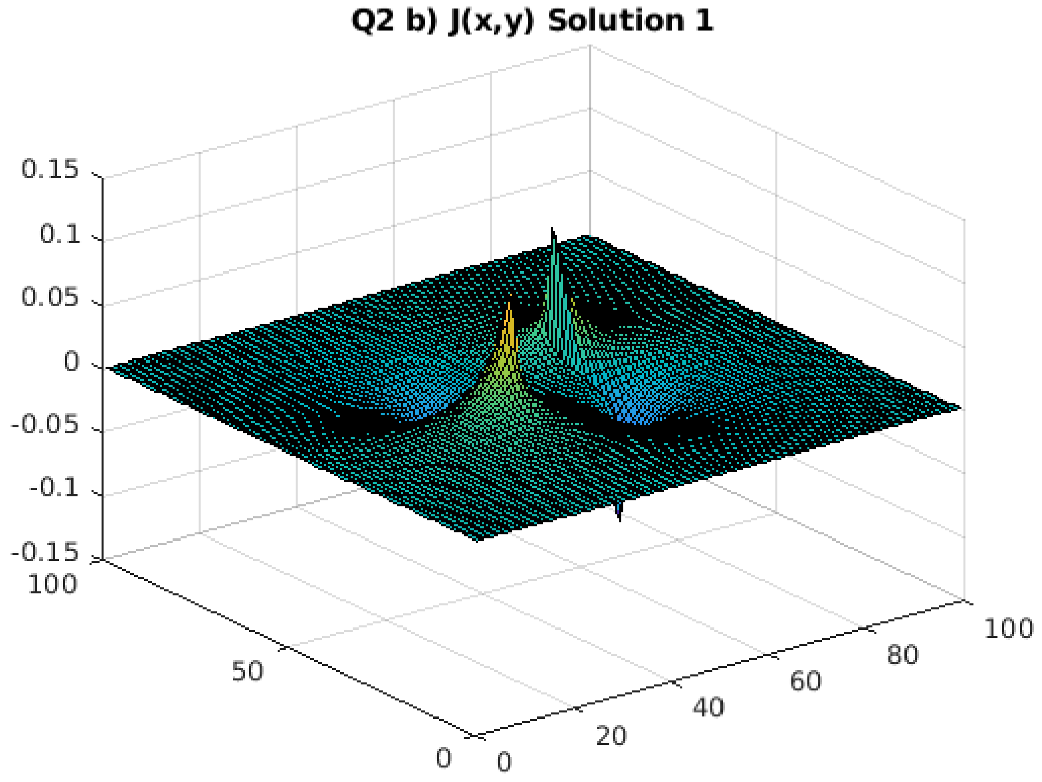
In this section, we investigate the effect of mesh density. To do this, we set W and L to 100, double what they were in part a). The bottleneck was also increased in size, so the geometry stays constant. Ultimately, this results in the same shape, but with double the points considered in both directions.

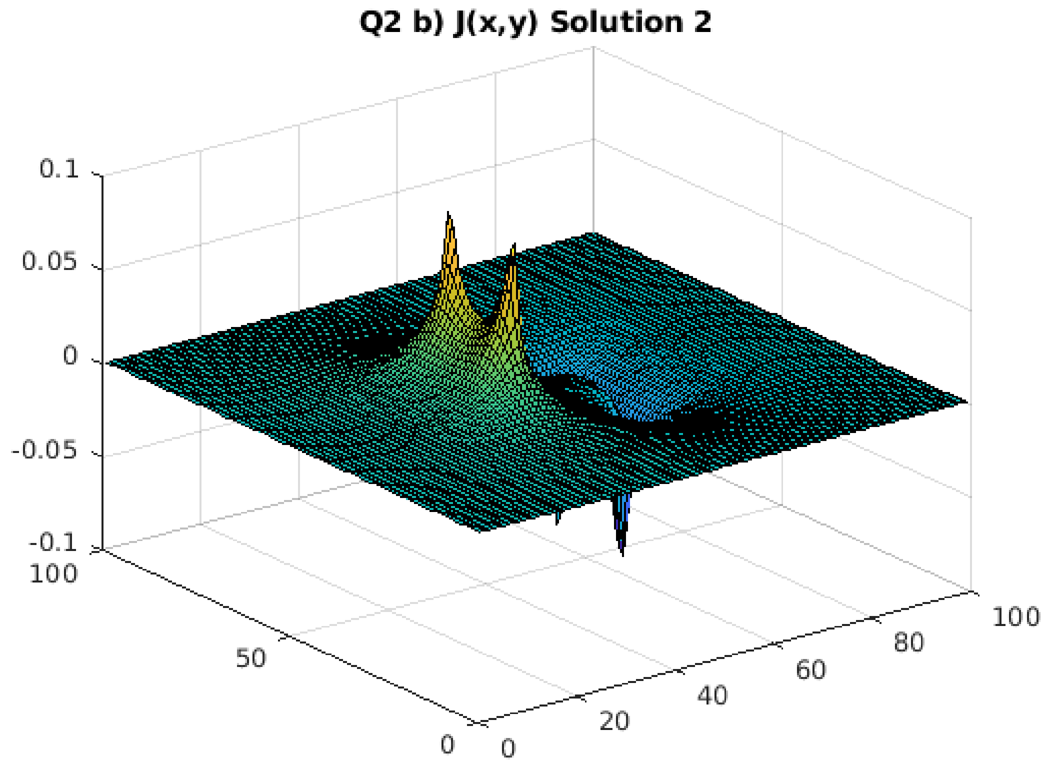
Unfortunately, I have to copy and paste the whole code from part a) here because Matlab insists that functions must be at the end of the file. If I used the function, the code won't be displayed where I want when I publish, so to the TA marking this, I'm sorry for the redundant clutter :(I've commented out some of the plots that are less essential like the sigma and spy, which won't change much from a) to trim a page from the report.

Observations of the higher mesh density: It didn't seem to improve much, 50x50 was already good enough to see what was going on and 100x100 made the computation take a few seconds longer. I also tried with 25x25, and while noticably less detailed and jagged, particularly around the boxes, it would still be acceptable for some applications. There appears to be a tradeoff between computation time, and simulation resolution.

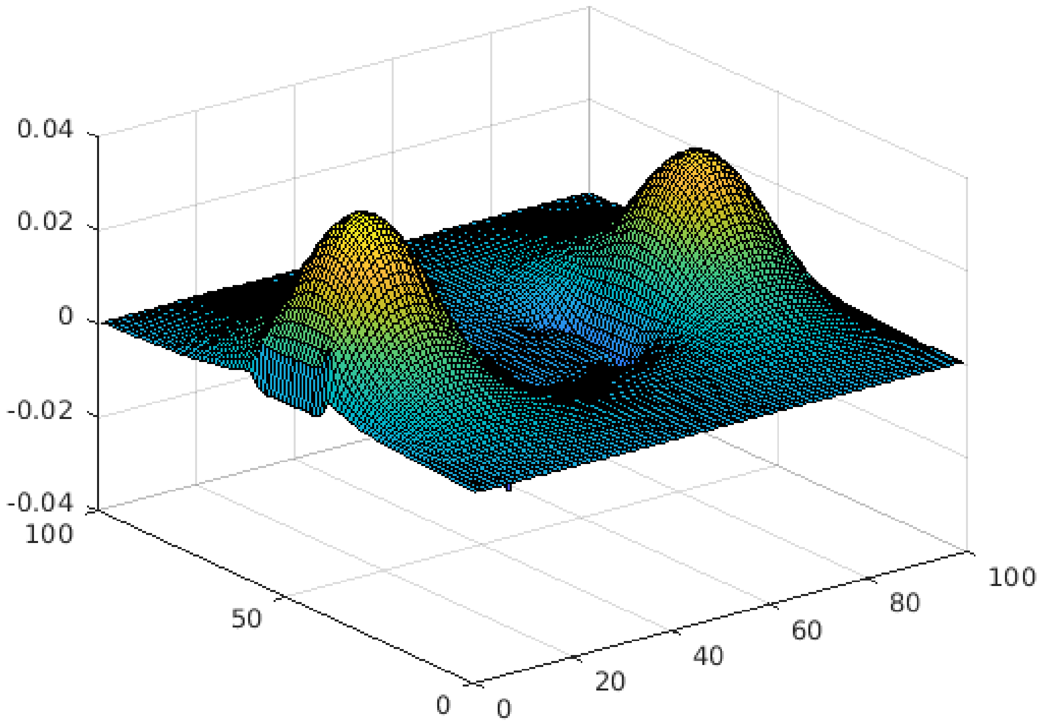
```
nw = 100;
nl = 100;
Vo = 10;
sigma_in = 10E-2;
sigma_out = 1;
wanted_solutions = 4;
sigma = zeros(nl, nw);

% build our sigma matrix and plot it
```

```
for i = 1:nl
    for j = 1:nw
        % check if inside box
        if (i > 40 && i < 60) && ((j > 1 && j < 40) || (j > 60))
            sigma(i, j) = sigma_in;
        % otherwise, we are outside the box
        else
            sigma(i, j) = sigma_out;
        end
    end
end

% initialize and populate the G matrix
G = zeros(nl*nw, nl*nw);
for i = 1:nl
    for j = 1:nw
        n = j + (i - 1) * nw;
        nxm = j + (i - 2) * nw;
        nxp = j + i * nw;
        nyp = (j + 1) + (i - 1) * nw;
        nym = (j - 1) + (i - 1) * nw;

        % check if we are at the l = 0 case, apply boundary
        if i == 1
            G(n, n) = Vo;

        % we still need to check if we are at an edge to know which
        % terms contribute to G. Corners are most specific cases, we
 will
        % cover those first, except for x = 0 because that is tied to
 a
        % boundary condition and will always be Vo

        % top right corner, no nxp or nym
        elseif i == nl && j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % bottom right corner, no nxp or nyp
        elseif i == nl && j == nw
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);

        % now we cover the cases along the edge. We already know we
 aren't
        % at a corner at this point

        % along right edge, so no nxp term
        elseif i == nl
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % along top edge, no nym
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % along bottom edge, no nyp
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % otherwise, we're somewhere in the middle, all terms
 considered
        else
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        end
    end
end

[E, D] = eigs(G, wanted_solutions, 'SM');

% plot J(x,y)
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
    for count = 1:nw*nl
        map(i, j) = E(count, solution);
        if j == nw
            j = 1;
            i = i + 1;
        else
            j = j + 1;
        end
    end
    figure
    surf(map)
    title(sprintf('Q2 b) J(x,y) Solution %d', solution))
    axis([0 nw 0 nl])
end
```

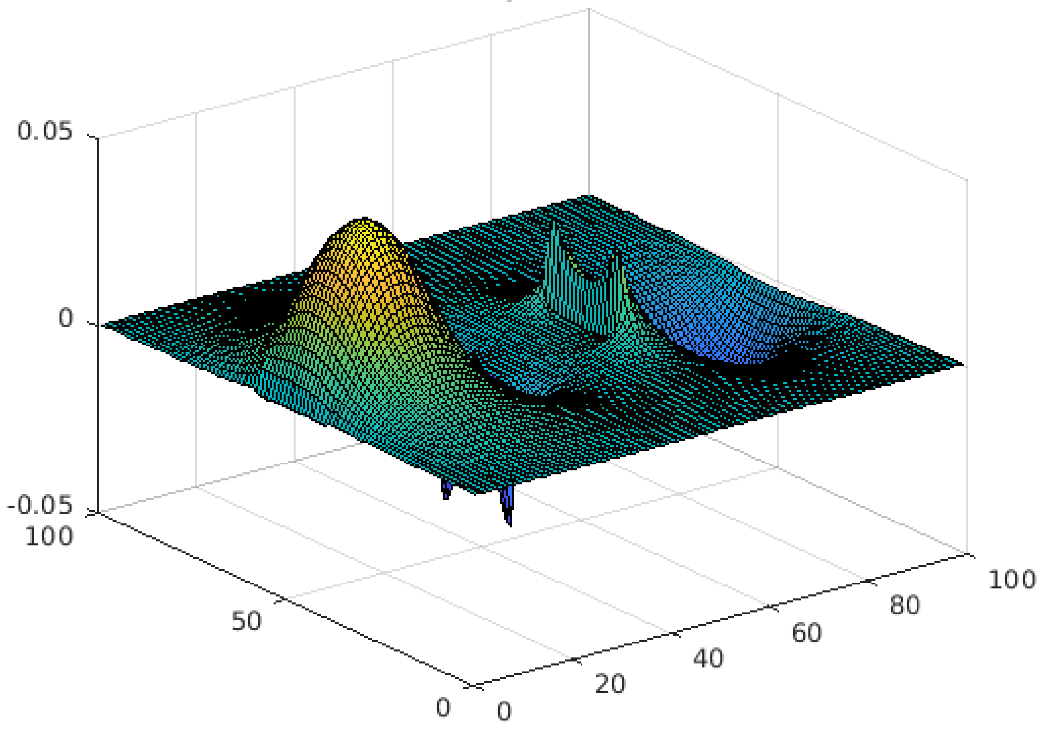

Q2 b) J(x,y) Solution 1
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
100
50
0
0
20
40
60
80
100


Q2 b) J(x,y) Solution 2
0.1
0.05
0
-0.05
-0.1
100
50
0
0
20
40
60
80
100

Q2 b) J(x,y) Solution 3



Q2 b) J(x,y) Solution 4



Question 2 c)

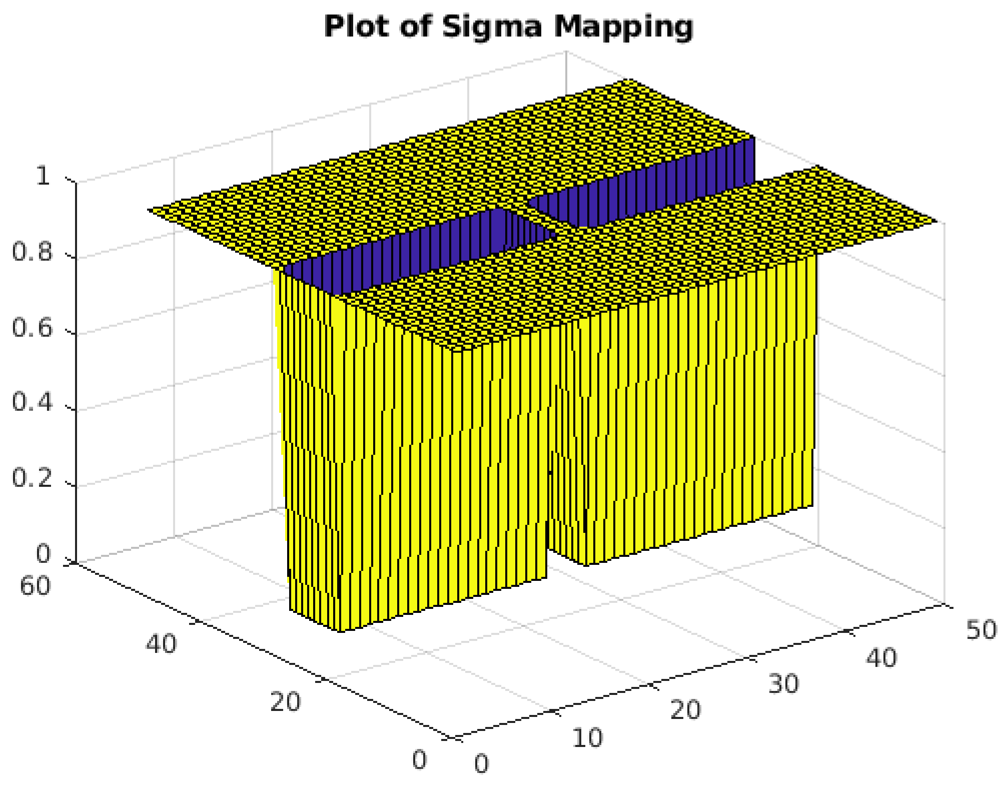
In this section, we investigate the narrowing of the bottleneck. We will return to the 50x50 mesh size used in a) because it's faster and has an acceptable level of detail. The bottleneck will still cover 20 < x < 30, like in a), however, it will extend from 0 < y < 24, and 26 < y < W. This leaves a space of 2 in the bottleneck.

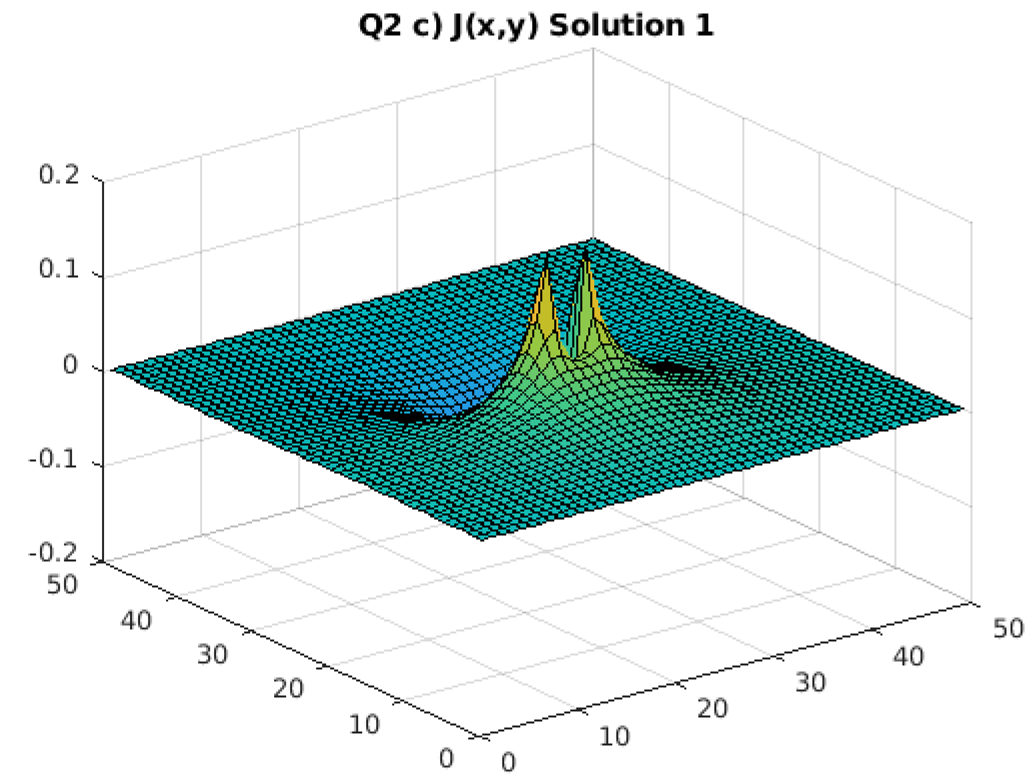
Observations: This makes the current density spike way up in the small area between the boxes. This makes sense, because we've essentially made a tiny resistor in parallel with a massive one. All the current will go through the smaller resistor.

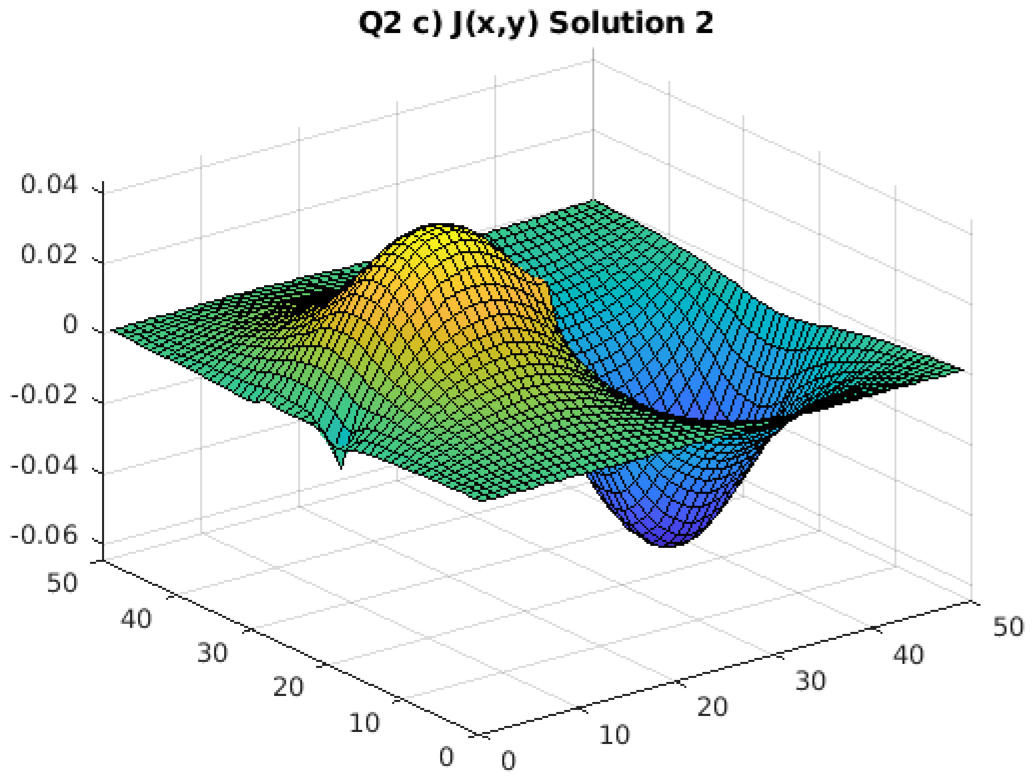
```
nw = 50;
nl = 50;
Vo = 10;
sigma_in = 10E-2;
sigma_out = 1;
wanted_solutions = 4;
sigma = zeros(nl, nw);

% build our sigma matrix
for i = 1:nl
    for j = 1:nw
        % check if inside box
        if (i > 20 && i < 30) && ((j > 1 && j < 24) || (j > 26))
            sigma(i, j) = sigma_in;
        % otherwise, we are outside the box
        else
            sigma(i, j) = sigma_out;
        end
    end
end
figure
surf(sigma)
title('Plot of Sigma Mapping')

% initialize and populate the G matrix
G = zeros(nl*nw, nl*nw);
for i = 1:nl
    for j = 1:nw
        n = j + (i - 1) * nw;
        nxm = j + (i - 2) * nw;
        nxp = j + i * nw;
        nyp = (j + 1) + (i - 1) * nw;
        nym = (j - 1) + (i - 1) * nw;

        % check if we are at the l = 0 case, apply boundary
        if i == 1
            G(n, n) = Vo;

        % we still need to check if we are at an edge to know which
        % terms contribute to G. Corners are most specific cases, we
 will
        % cover those first, except for x = 0 because that is tied to
 a
```

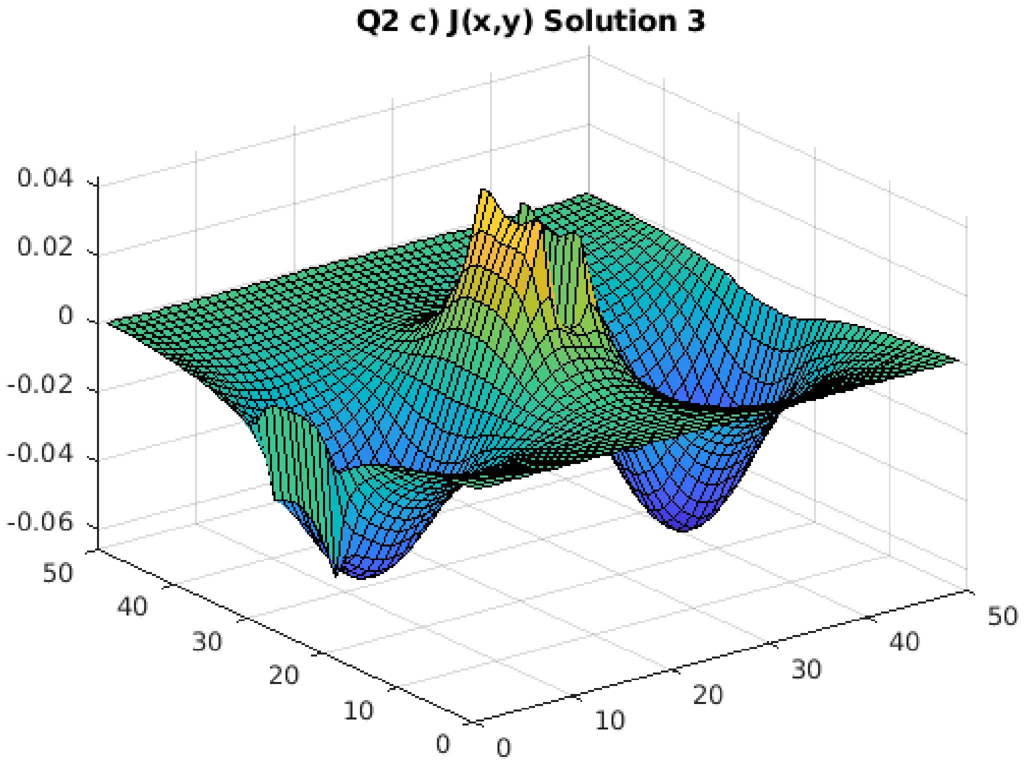
```
        % boundary condition and will always be Vo

        % top right corner, no nxp or nym
        elseif i == nl && j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % bottom right corner, no nxp or nyp
        elseif i == nl && j == nw
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);

        % now we cover the cases along the edge. We already know we
 aren't
        % at a corner at this point

        % along right edge, so no nxp term
        elseif i == nl
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % along top edge, no nym
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % along bottom edge, no nyp
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % otherwise, we're somewhere in the middle, all terms
 considered
        else
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        end
    end
end

[E, D] = eigs(G, wanted_solutions, 'SM');

% plot J(x,y)
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
```

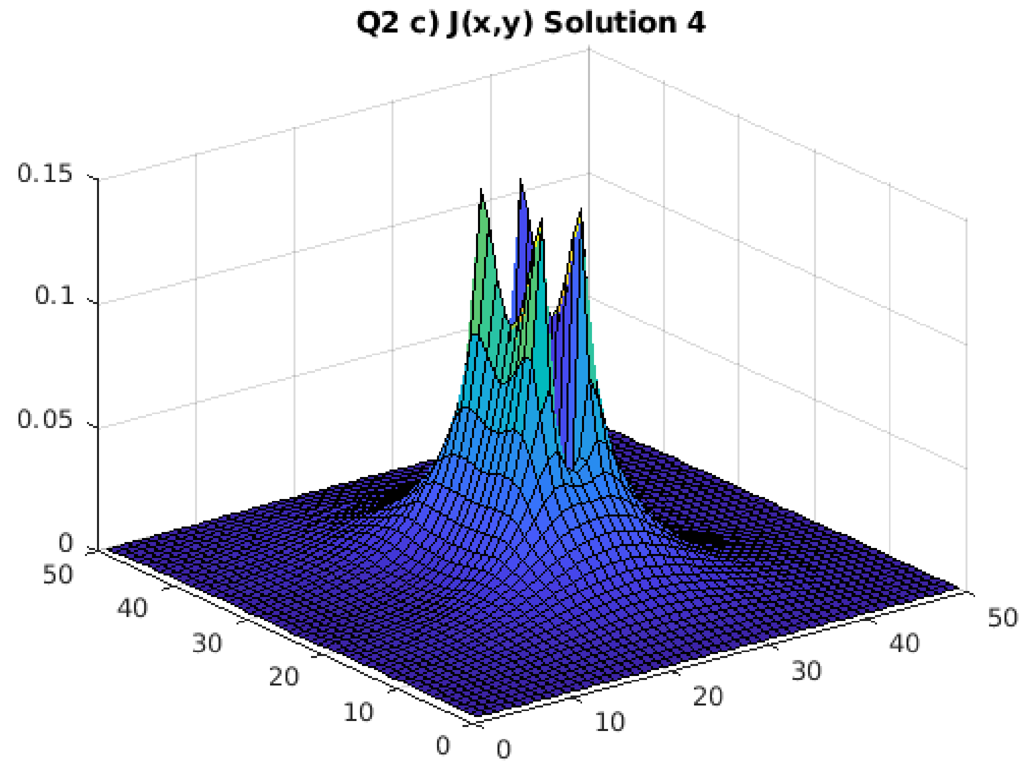
```
        for count = 1:nw*nl
            map(i, j) = E(count, solution);
            if j == nw
                j = 1;
                i = i + 1;
            else
                j = j + 1;
            end
        end
        figure
        surf(map)
        title(sprintf('Q2 c) J(x,y) Solution %d', solution))
        axis([0 nw 0 nl])
    end
```




Q2 c) J(x,y) Solution 1
0.2
0.1
0
-0.1
-0.2
50
40
30
20
10
0
0
10
20
30
40
50


Q2 c) J(x,y) Solution 2
0.04
0.02
0
-0.02
-0.04
-0.06
50
40
30
20
10
0
0
10
20
30
40
50


Q2 c) J(x,y) Solution 3
0.04
0.02
0
-0.02
-0.04
-0.06
50
40
30
20
10
0
0
10
20
30
40
50


Q2 c) J(x,y) Solution 4
0.15
0.1
0.05
0
50
40
30
20
10
0
0
10
20
30
40
50

Question 2 d)

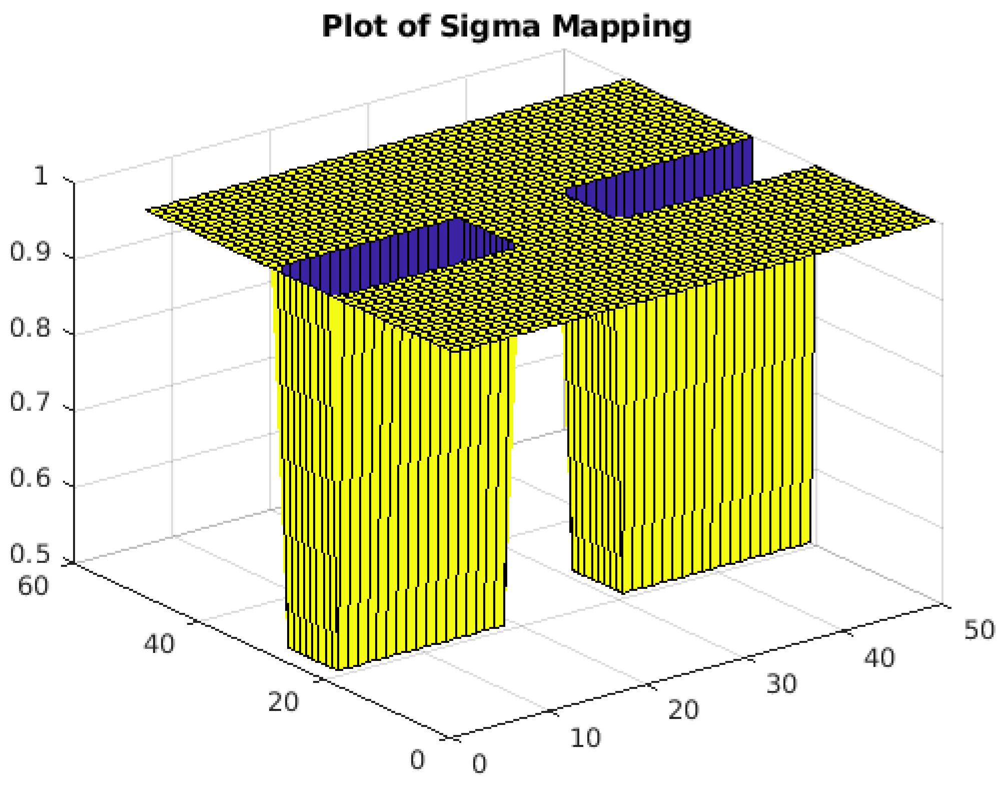
In this section, we vary the conductivity of the boxes. In this case, we will try raising it to only be a little bit lower than the rest of the area. We will be returning to the geometry and mesh density used in a).

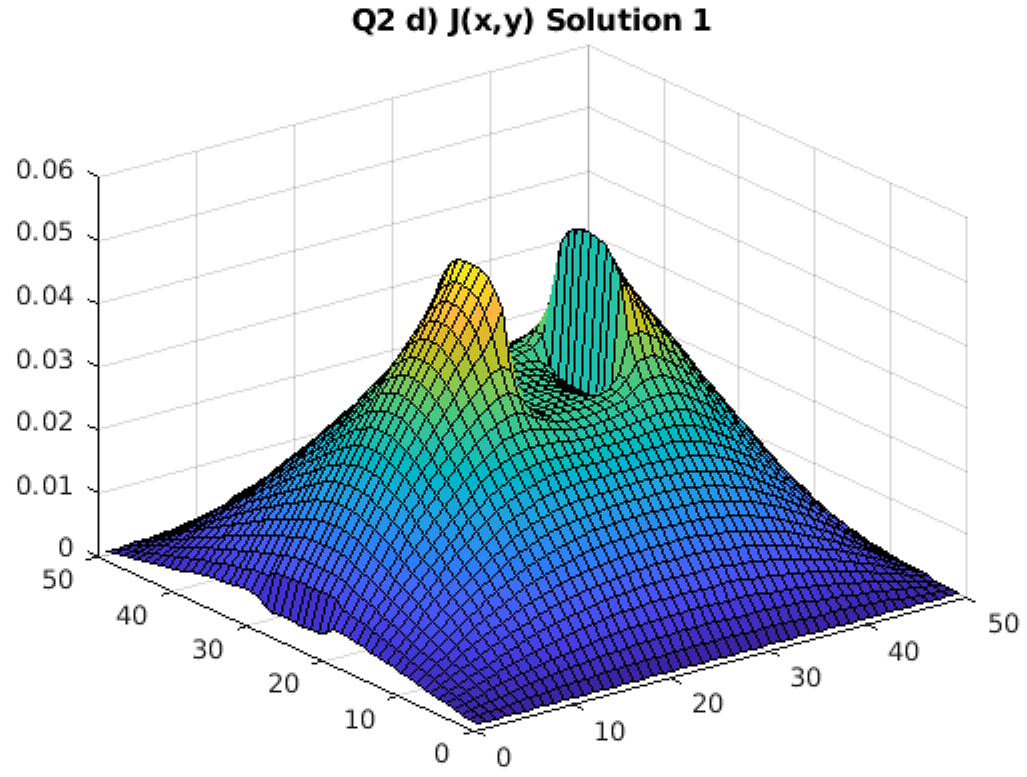
Observations: It looks like more current is now getting through the boxes, as expected. The plots look to be smoothing slowly to match the case with no bottleneck at all, but with minor blips on the edges of the box. I would expect these to become smaller and smother as the conductivity of the boxes gets closer to matching the surrounding area. The solutions atart looking closer to what was seen in Q1.

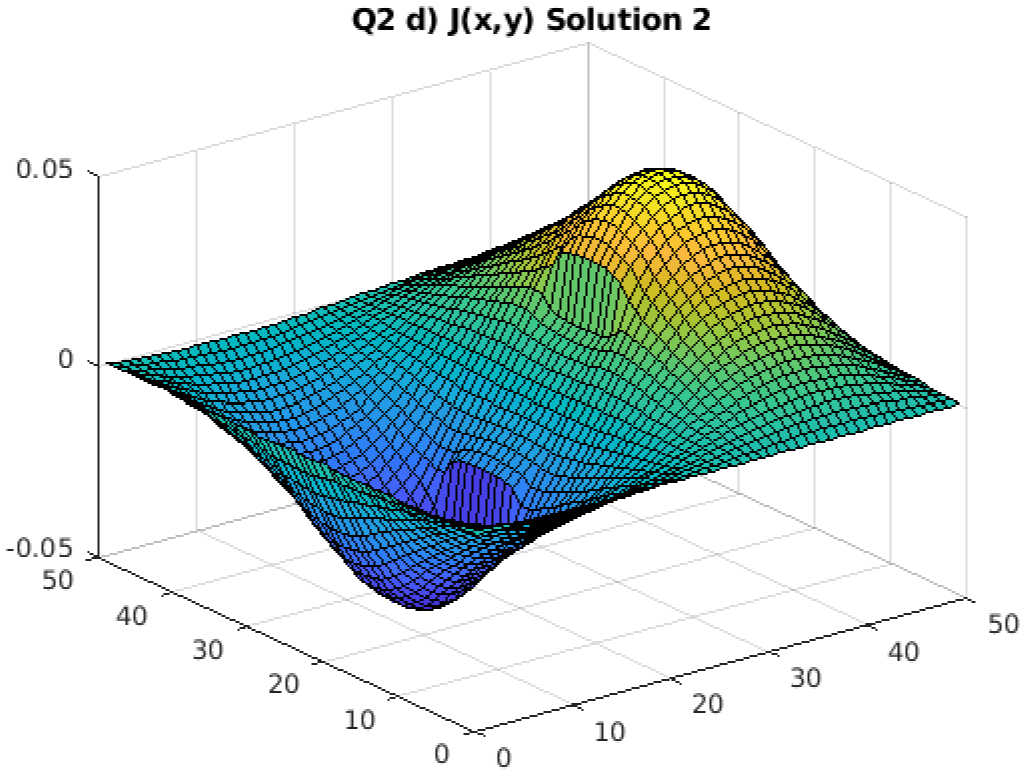
```
nw = 50;
nl = 50;
Vo = 10;
sigma_in = 0.5;      % boxes are half as conductive as outside
sigma_out = 1;
wanted_solutions = 4;
sigma = zeros(nl, nw);

% build our sigma matrix and plot it
for i = 1:nl
    for j = 1:nw
        % check if inside box
        if (i > 20 && i < 30) && ((j > 1 && j < 20) || (j > 30))
            sigma(i, j) = sigma_in;
        % otherwise, we are outside the box
        else
            sigma(i, j) = sigma_out;
        end
    end
end
figure
surf(sigma)
title('Plot of Sigma Mapping')

% initialize and populate the G matrix
G = zeros(nl*nw, nl*nw);
for i = 1:nl
    for j = 1:nw
        n = j + (i - 1) * nw;
        nxm = j + (i - 2) * nw;
        nxp = j + i * nw;
        nyp = (j + 1) + (i - 1) * nw;
        nym = (j - 1) + (i - 1) * nw;

        % check if we are at the l = 0 case, apply boundary
        if i == 1
            G(n, n) = Vo;

        % we still need to check if we are at an edge to know which
        % terms contribute to G. Corners are most specific cases, we
 will
        % cover those first, except for x = 0 because that is tied to
 a
```

```
        % boundary condition and will always be Vo

        % top right corner, no nxp or nym
        elseif i == nl && j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % bottom right corner, no nxp or nyp
        elseif i == nl && j == nw
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);

        % now we cover the cases along the edge. We already know we
 aren't
        % at a corner at this point

        % along right edge, so no nxp term
        elseif i == nl
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % along top edge, no nym
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
        % along bottom edge, no nyp
        elseif j == 1
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        % otherwise, we're somewhere in the middle, all terms
 considered
        else
            G(n, n) = -4*sigma(i, j);
            G(n, nxm) = 1*sigma(i, j);
            G(n, nxp) = 1*sigma(i, j);
            G(nyp, n) = 1*sigma(i, j);
            G(nym, n) = 1*sigma(i, j);
        end
    end
end

[E, D] = eigs(G, wanted_solutions, 'SM');

% plot and pray, starting with J(x,y)
for solution = 1:wanted_solutions
    i = 1;
    j = 1;
    map = zeros(nw,nl);
```

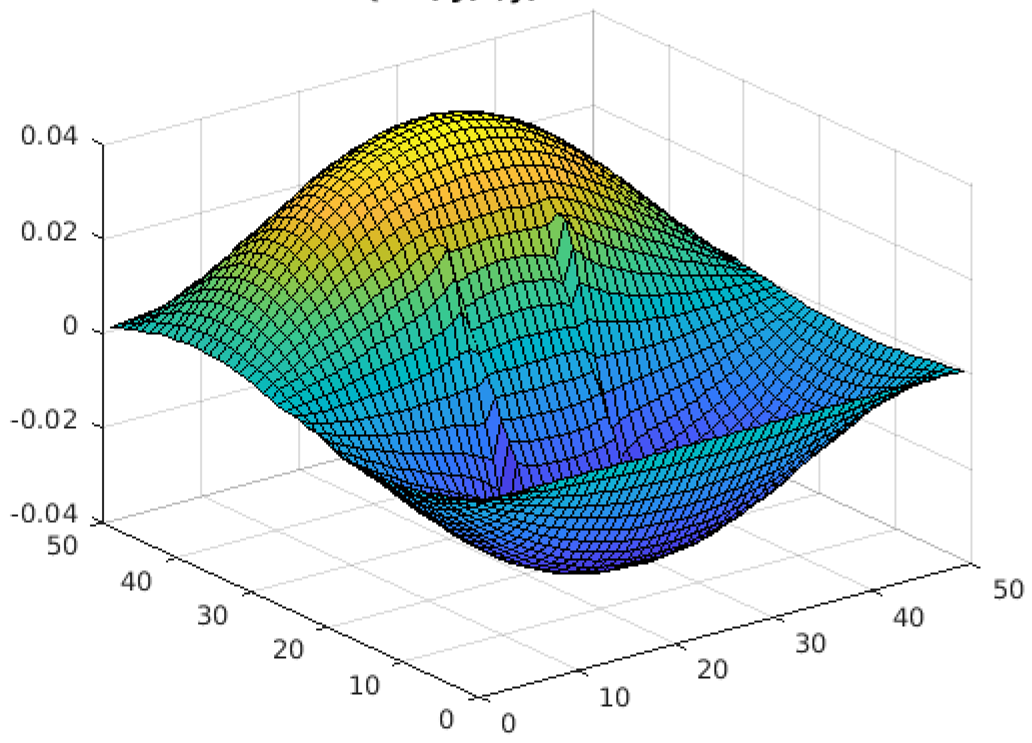
```
        for count = 1:nw*nl
            map(i, j) = E(count, solution);
            if j == nw
                j = 1;
                i = i + 1;
            else
                j = j + 1;
            end
        end
        figure
        surf(map)
        title(sprintf('Q2 d) J(x,y) Solution %d', solution))
        axis([0 nw 0 nl])
    end
```



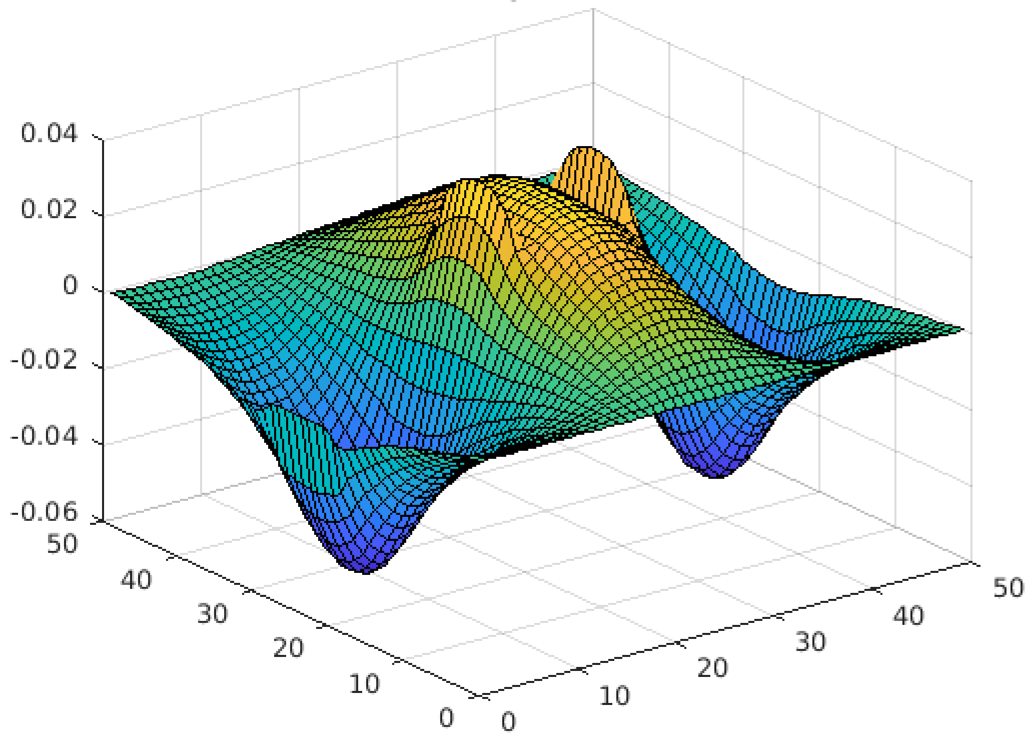

Q2 d) J(x,y) Solution 1
0.06
0.05
0.04
0.03
0.02
0.01
0
50
40
30
20
10
0
0
10
20
30
40
50


Q2 d) J(x,y) Solution 2
0.05
0
-0.05
50
40
30
20
10
0
0
10
20
30
40
50

Q2 d) J(x,y) Solution 3



Q2 d) J(x,y) Solution 4



Published with MATLAB® R2019b