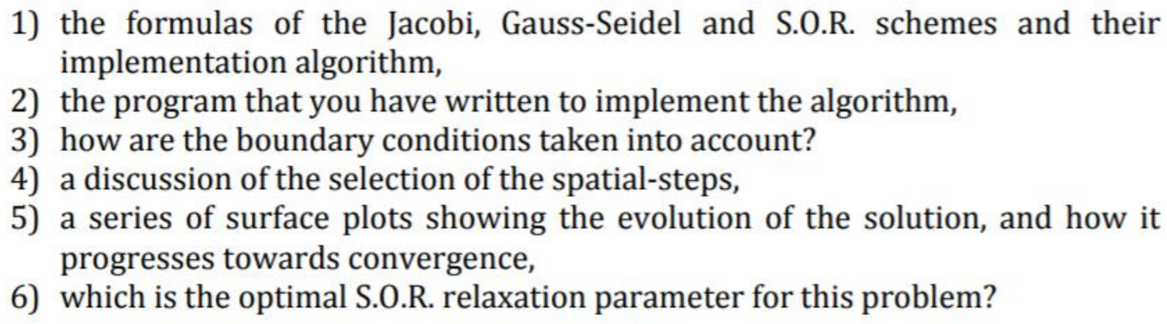
James Condie

Phys 404

HW 8

The requirements are: (taken from the assignment pdf)



1) the formulas of the Jacobi, Gauss-Seidel and S.O.R. schemes and their implementation algorithm,
2) the program that you have written to implement the algorithm,
3) how are the boundary conditions taken into account?
4) a discussion of the selection of the spatial-steps,
5) a series of surface plots showing the evolution of the solution, and how it progresses towards convergence,
6) which is the optimal S.O.R. relaxation parameter for this problem?

Section 1: Formulas for Jacobi, Gauss-Seidel and S.O.R. schemes and their implementation algorithm

Consider an x-y plane, and a two dimensional function $\phi(x,y)$. We can discretize the x-y plane as a mesh grid with indices ‘i’ an ‘j’, such that

$$\phi_{i,j} = \phi(x,y)$$

Jacobi, Gauss-Seidel and S.O.R. schemes are classified as relaxation methods (textbook, Garcia p. 252), which require an initial guess $\phi_{i,j}^{k}$. Iterations of the formula deliver an improved guess $\phi_{i,j}^{k+1}$.

Assuming our discretization choices for x and y have the same step size, the Jacobi formula for interior points of the mesh grid is given by:

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i-1,j}^{k} + \phi_{i+1,j}^{k} + \phi_{i,j-1}^{k} + \phi_{i,j+1}^{k})$$

Jacobi formula (equal spacing for Δx and Δy). This equation does not apply to points on boundary. Boundary points must be specified initially.

Notice the next guess for a data point is simply the average of the 4 adjacent neighboring points in the previous guess. As Garcia points out, the Laplace equation can be interpreted as the steady state solution to a diffusion equation as t approaches infinity and there's no longer any time dependence (Garcia, p.252).

The Gauss-Seidel formula is a minor improvement on the Jacobi formula. Unlike the Jacobi method which only uses points from the previous guess, the Gauss-Seidel method uses the most current data point available. This results in faster convergence. It looks like this:

$$\phi_{i,j}^{k+1} = \frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j-1}^{k+1} + \phi_{i,j+1}^{k})$$

Gauss-Seidel Formula: converges faster than Jacobi formula

The successive over relaxation method or SOR (the text calls it the simultaneous relaxation method) further improves upon the Gauss-Seidel formula by "overcorrecting the value" of phi at every iteration (text p.252). It looks like this:

$$\phi_{i,j}^{k+1} = (1-\omega)\phi_{i,j}^{k} + \omega\frac{1}{4}(\phi_{i-1,j}^{k+1} + \phi_{i+1,j}^{k} + \phi_{i,j-1}^{k+1} + \phi_{i,j+1}^{k})$$

Successive over relaxation (SOR) where ω is determined either by trial and error or in special cases theoretically. Stable values of ω are known to be within 1<ω<2.

All three methods follow roughly the same algorithm:

1. Discretize the two dimensional space of interest
2. Set initial conditions on all boundaries

3. Start with some initial guess for all interior points, there now should be values for every element in $\phi_{i,j}^{k}$ where k=1.
4. Use 1 of the 3 formulas: Jacobi, Gauss-Seidel, SOR, to populate values for the next iteration of guesses for $\phi_{i,j}^{k+1}$.
5. Compare current φ values with previous φ values. Stop iterations if guesses are converging to within specified fractional change.
6. The newest φ is the final numerical solution.

Section 2: The Program that has been written to implement the algorithm

I have written 3 separate programs for the 3 relaxation methods (they are all very similar).

Jacobi Method

```
%hw 8
%solving eliptical PDE using Jacobi,
% next versions will do Gauss_Seidel and successive
%over-relaxation (SOR)

clc
clear

%set steps and maximum iterations
steps=100;
max_iter=100000;
max_error=1e-12;
figures=10;

%set dimensions
size=3; %cm;


phi_new=ones(steps,steps);

%set initial conditions

%top border at 5 volts
phi_new(steps,2:steps-1)=5*ones(1,steps-2);
```

```
%other borders are at zero
phi_new(1,:)=zeros(1,steps);
phi_new(2:steps-1,1)=zeros(steps-2,1);
phi_new(2:steps-1,steps)=zeros(steps-2,1);

%create copy to keep track of old iterations
phi_old=phi_new;
done=false;

%Jacobi guesses for new iterations
for iter=1:max_iter
    if(done==true)
        break
    end
    for i=2:steps-1
        for j=2:steps-1
            phi_new(j,i)=.25*(phi_old(j,i-1)+phi_old(j,i+1)+phi_old(j-1,i)+phi_old(j+1,i));
        end
    end

    %check for convergence
    error_matrix=abs(phi_new-phi_old)./phi_old;
    error=max(error_matrix);
    if (error<max_error)
        done=true;
    end

    phi_old=phi_new;

    %plot various progressions
    %plots less often for higher iteration number

    if((iter<10 && mod(iter,2)==0)||(iter<100 && mod(iter,10)==0)||( iter<1000 &&
mod(iter,100)==0) || mod(iter, max_iter/figures)==0 || done==true)
        x=linspace(0,size,steps);
        y=linspace(0,size,steps);
     [X,Y]=meshgrid(x,y);
    figure
    C=phi_new;


    surf(X,Y,phi_new,C,'FaceAlpha',.79,'EdgeColor', 'none','FaceColor','interp')
    view(-48,25)
    c=colorbar;
    c.Label.String = 'Volts';
    camlight left
    axis tight
        if(done==true)
        title(['final Jacobi iteration: ',num2str(iter)])
        else
        title(['Jacobi iteration: ',num2str(iter)])
        end
    zlabel({'Volts'});
    ylabel({'cm'});
    xlabel({'cm'});
    end
end
```

Output is reserved for section 5.

Published with MATLAB® R2018a

Gauss-Seidel

```
%hw 8
%solving eliptical PDE using Gauss_Seidel

clc
clear

%set steps and maximum iterations
steps=100;
max_iter=100000;
max_error=1e-12;
figures=10;

%set dimensions
size=3; %cm;


phi_new=ones(steps,steps);

%set initial conditions

%top border at 5 volts
phi_new(steps,2:steps-1)=5*ones(1,steps-2);

%other borders are at zero
phi_new(1,:)=zeros(1,steps);
phi_new(2:steps-1,1)=zeros(steps-2,1);
phi_new(2:steps-1,steps)=zeros(steps-2,1);

%create copy to keep track of old iterations
phi_old=phi_new;
done=false;

%Gauss-Seidel guesses for new iterations
for iter=1:max_iter
    if(done==true)
        break
    end
    for i=2:steps-1
        for j=2:steps-1
```

```
                phi_new(j,i)=.25*(phi_new(j,i-1)+phi_old(j,i+1)+phi_new(j-1,i)+phi_old(j+1,i));
            end
        end

        %check for convergence
        error_matrix=abs(phi_new-phi_old)./phi_old;
        error=max(error_matrix);
        if (error<max_error)
            done=true;
        end

        phi_old=phi_new;

        %plot various progressions

        if((iter<10 && mod(iter,2)==0)||(iter<100 && mod(iter,10)==0)||( iter<1000 &&
mod(iter,100)==0) || mod(iter, max_iter/figures)==0 || done==true)
            x=linspace(0,size,steps);
            y=linspace(0,size,steps);
         [X,Y]=meshgrid(x,y);
        figure
        C=phi_new;


        surf(X,Y,phi_new,C,'FaceAlpha',.79,'EdgeColor', 'none','FaceColor','interp')
        view(-48,25)
        c=colorbar;
        c.Label.String = 'Volts';
        camlight left
        axis tight
            if(done==true)
            title(['final Gauss-Sidel iteration: ',num2str(iter)])
            else
            title(['Gauss-Sidel iteration: ',num2str(iter)])
            end
        zlabel({'Volts'});
        ylabel({'cm'});
        xlabel({'cm'});
        end
end
```

Output is reserved for section 5.

SOR

```
%hw 8
%solving eliptical PDE using Successive Over Relaxation (SOR)

clc
clear
```

```
%numerical parameters
steps=100;
max_iter=100000;
max_error=1e-12;
figures=10;
omega=2/(1+sin(pi/steps));  %stable for 1<omega<2

%set dimensions
size=3; %cm;


phi_new=1*ones(steps,steps);

%set initial conditions

%top border at 5 volts
phi_new(steps,:)=5*ones(1,steps);

%other borders are at zero
phi_new(1,:)=zeros(1,steps);
phi_new(2:steps-1,1)=zeros(steps-2,1);
phi_new(2:steps-1,steps)=zeros(steps-2,1);

%create copy to keep track of old iterations
phi_old=phi_new;


%Gauss-Seidel guesses for new iterations
done=false;
for iter=1:max_iter
    if(done==true)
        break
    end

    for i=2:steps-1
        for j=2:steps-1
            phi_new(j,i)=(1-omega)*phi_old(j,i)+0.25*omega*(phi_new(j,i-
1)+phi_old(j,i+1)+phi_new(j-1,i)+phi_old(j+1,i));
        end
    end

    %check for convergence
    error_matrix=abs(phi_new-phi_old)./phi_old;
    error=max(max(error_matrix));
    if (error<max_error)
        done=true;
    end

    phi_old=phi_new;

    %plot various progressions

    %plots less often for higher order iterations
     if(iter==1 || (iter<10 && mod(iter,2)==0)||(iter<100 && mod(iter,10)==0)||( iter<1000 &&
mod(iter,100)==0) || mod(iter, max_iter/figures)==0 || done==true)
        x=linspace(0,size,steps);
        y=linspace(0,size,steps);
     [X,Y]=meshgrid(x,y);

     figure
    C=phi_new;
    surf(X,Y,phi_new,C,'FaceAlpha',.79,'EdgeColor', 'none','FaceColor','interp')
    view(-48,25)
    c=colorbar;
    c.Label.String = 'Volts';
    camlight left
    axis tight
        if(done==true)
        title(['final SOR iteration: ',num2str(iter)])
        else
        title(['SOR iteration: ',num2str(iter)])
        end
    zlabel({'Volts'});
    ylabel({'cm'});
    xlabel({'cm'});
    end
end
```

Output is reserved for section 5.

Published with MATLAB® R2018a

Section 3: How are the boundary conditions taken into account?

The boundary conditions are ambiguous at corner points of the grid. To illustrate, consider a mesh grid that's 3x3 for simplicity where each number is an electric potential in Volts. With the given boundary conditions, there are 2 different interpretations.

Interpretation A: where initial rows get precedence

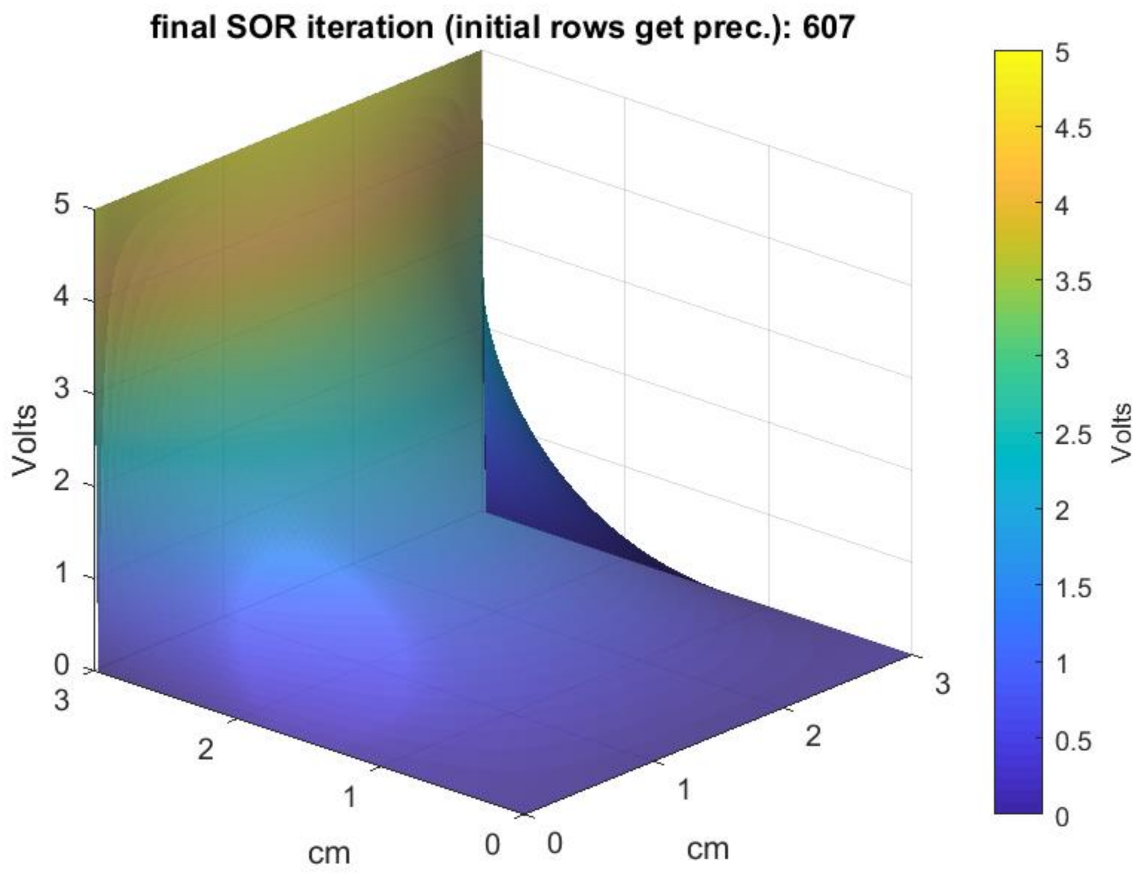
5	5	5
0	guess	0
0	0	0

The code for interpretation A where initial rows get precedence looks like:

```
%top border at 5 volts
phi_new(steps,:)=5*ones(1,steps);

%other borders are at zero
phi_new(1,:)=zeros(1,steps);
phi_new(2:steps-1,1)=zeros(steps-2,1);
phi_new(2:steps-1,steps)=zeros(steps-2,1);
```

The output looks like this:



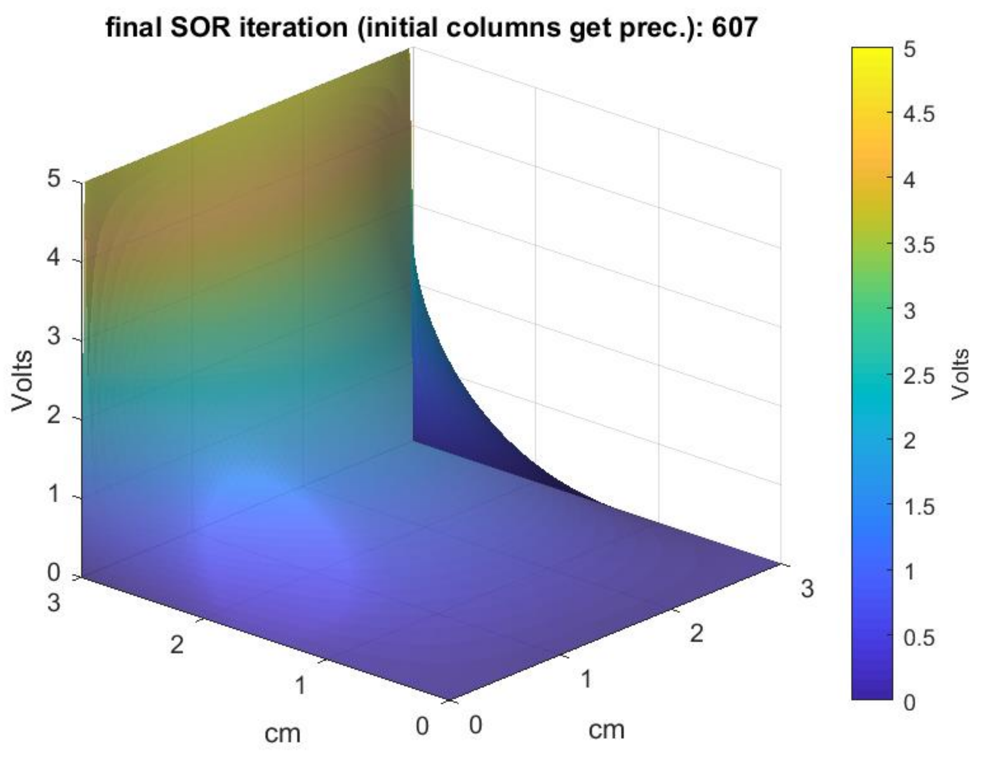
Interpretation B where initial columns get precedence looks like this:

0	5	0
0	guess	0
0	0	0

The code for interpretation B (initial columns get precedence looks like)

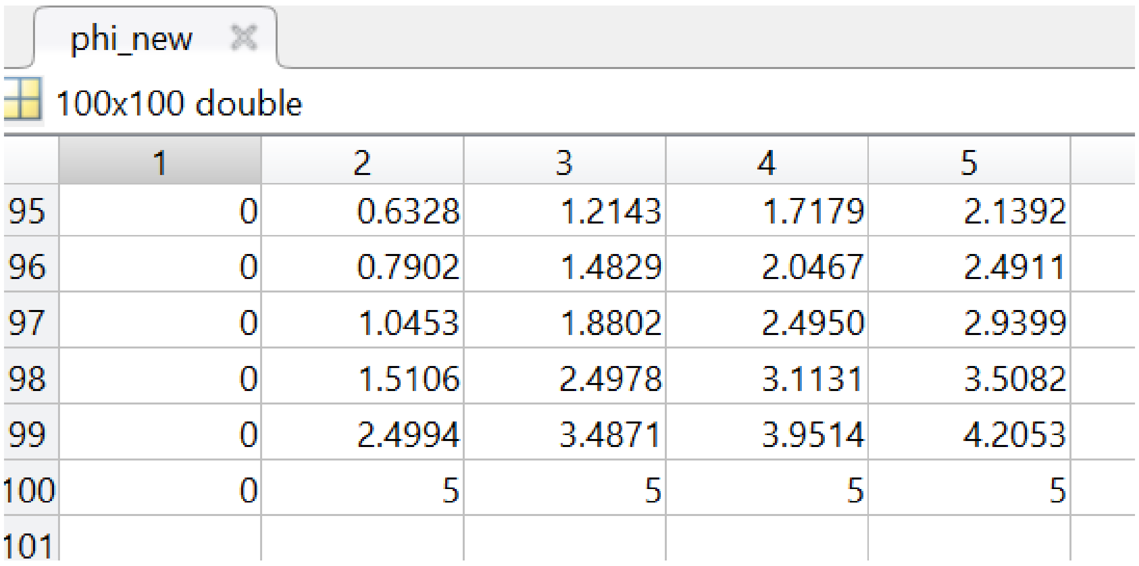
```
%top border at 5 volts
phi_new(steps,2:steps-1)=5*ones(1,steps-2);

%other borders are at zero
phi_new(1,:)=zeros(1,steps);
phi_new(2:steps,1)=zeros(steps-1,1);
phi_new(2:steps,steps)=zeros(steps-1,1);
```



There's no apparent difference in the two output plots. They even converge in the same number of guesses (or iterations). Let's investigate the final result in tabular form and compare.

First for column precedence, we see the following internal values are found.

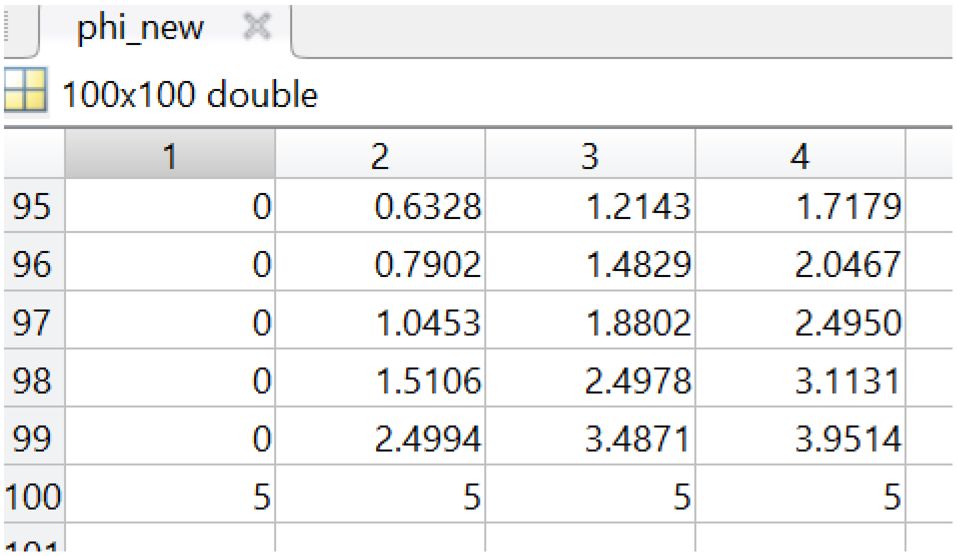


phi_new

100x100 double

	1	2	3	4	5
95	0	0.6328	1.2143	1.7179	2.1392
96	0	0.7902	1.4829	2.0467	2.4911
97	0	1.0453	1.8802	2.4950	2.9399
98	0	1.5106	2.4978	3.1131	3.5082
99	0	2.4994	3.4871	3.9514	4.2053
100	0	5	5	5	5
101					

For row precedence we still get the exact same internal points in the final result.



phi_new

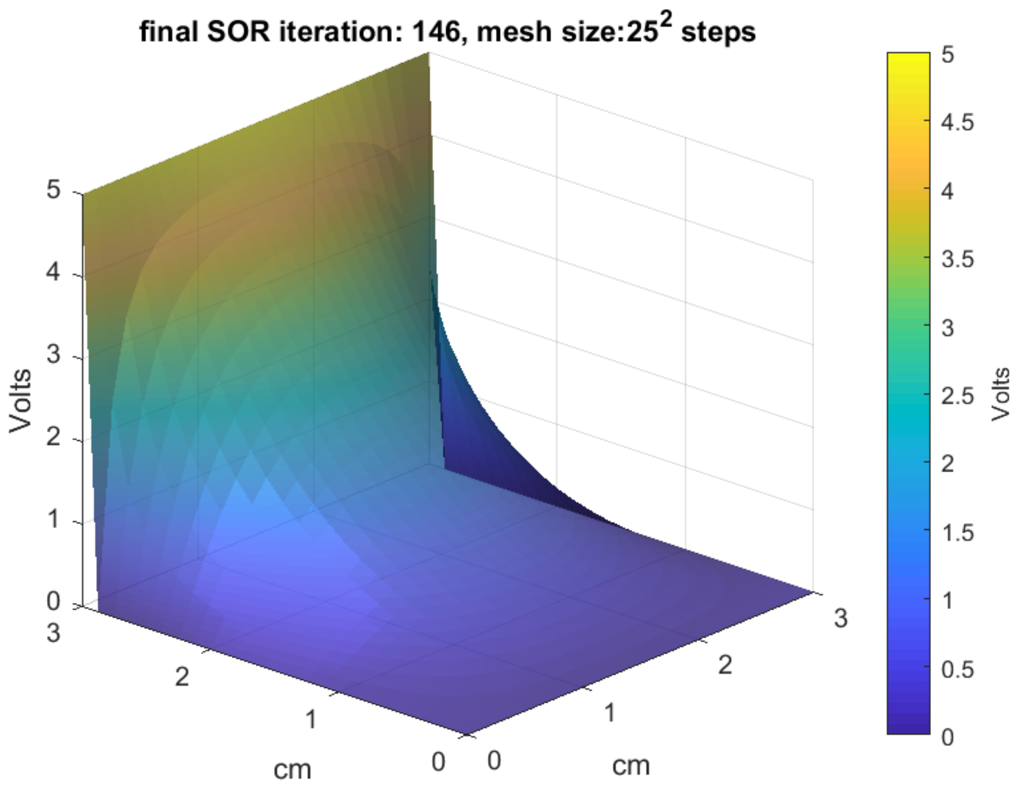
100x100 double

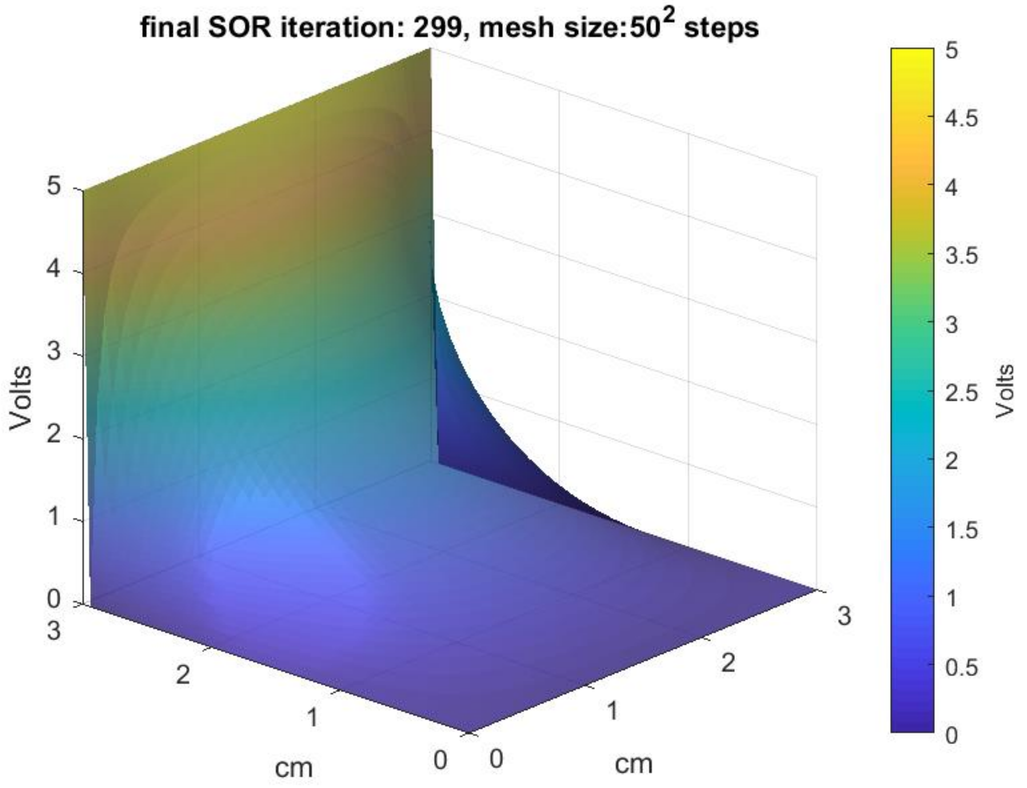
	1	2	3	4
95	0	0.6328	1.2143	1.7179
96	0	0.7902	1.4829	2.0467
97	0	1.0453	1.8802	2.4950
98	0	1.5106	2.4978	3.1131
99	0	2.4994	3.4871	3.9514
100	5	5	5	5

It seems to be of no matter how we interpret the corners. But I only claim so for this particular problem at this particular level of accuracy.

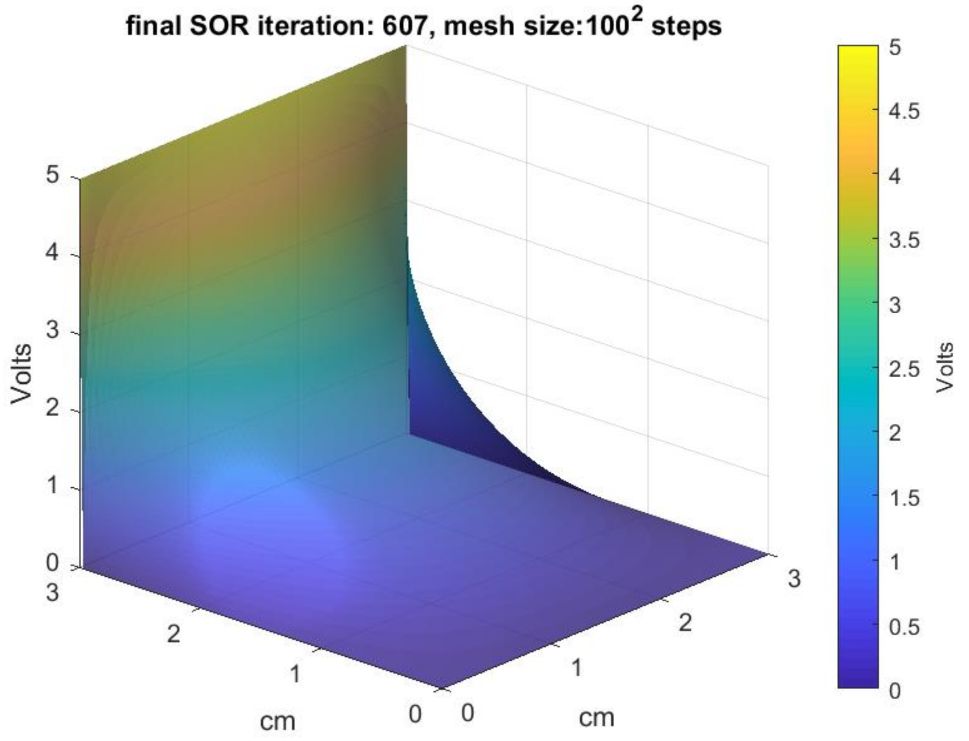
Section 4: A discussion of the selection of spatial steps

Below I show final surface plots for various mesh sizes. The titles the final number of iterations and mesh size. As usual, if one wants a more accurate result and smoother plot they should choose a larger mesh size (which gives smaller step size).

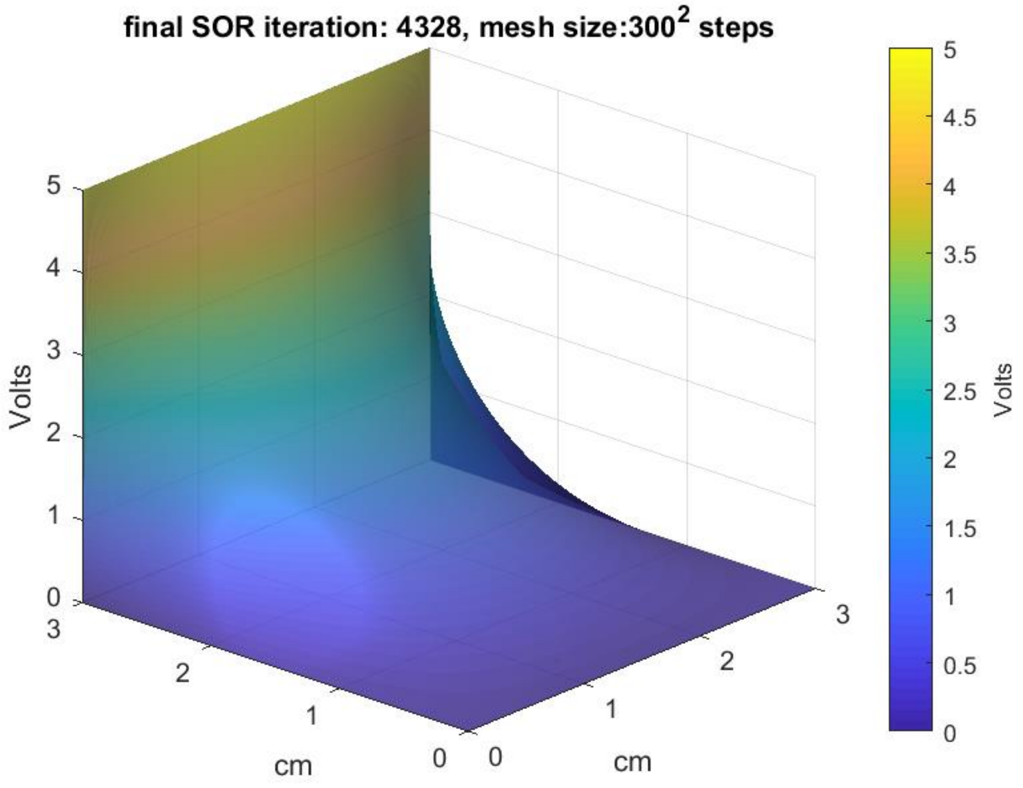


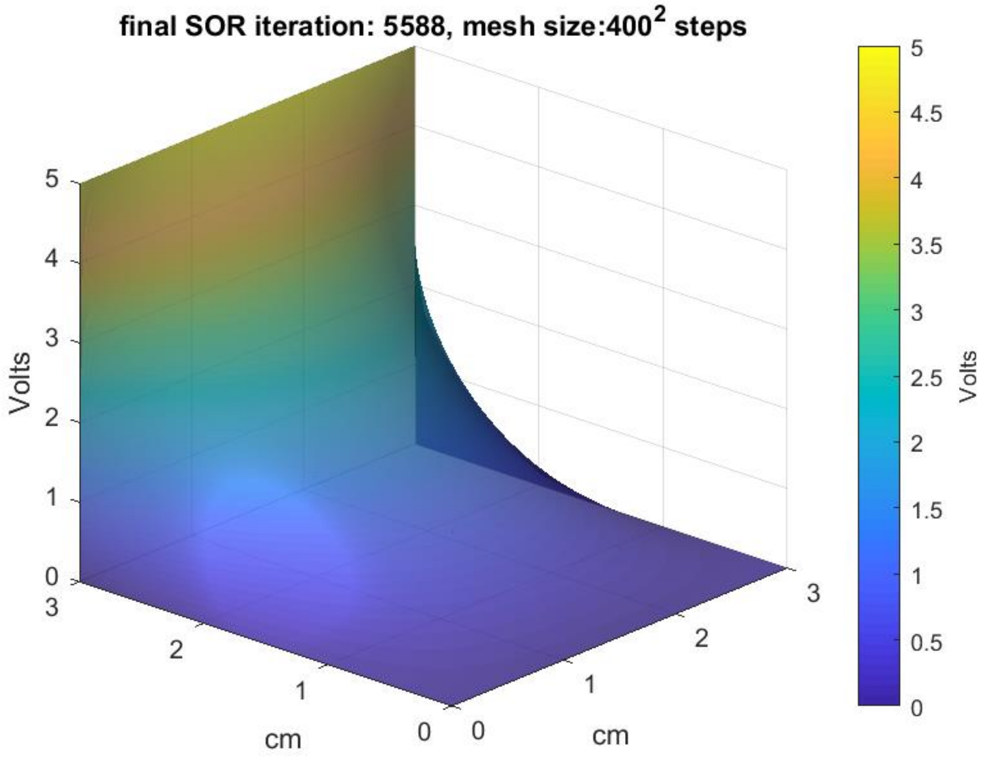


final SOR iteration: 299, mesh size:50^2 steps
Volts
cm
cm
Volts



final SOR iteration: 607, mesh size:100^2 steps
Volts
cm
cm
Volts

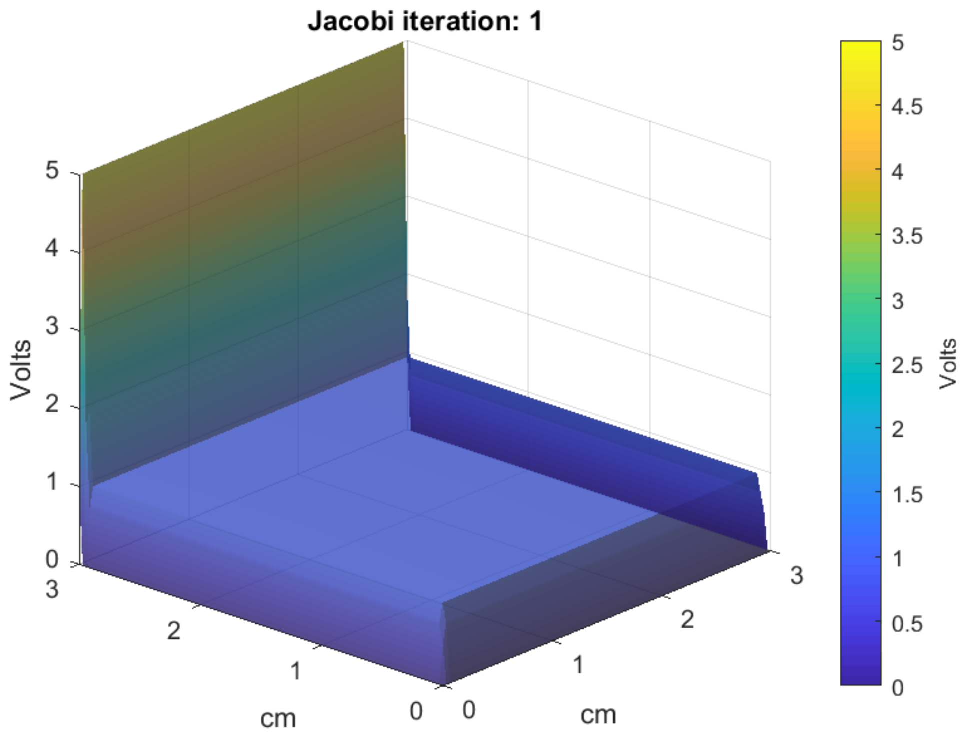

final SOR iteration: 4328, mesh size:300^2 steps
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

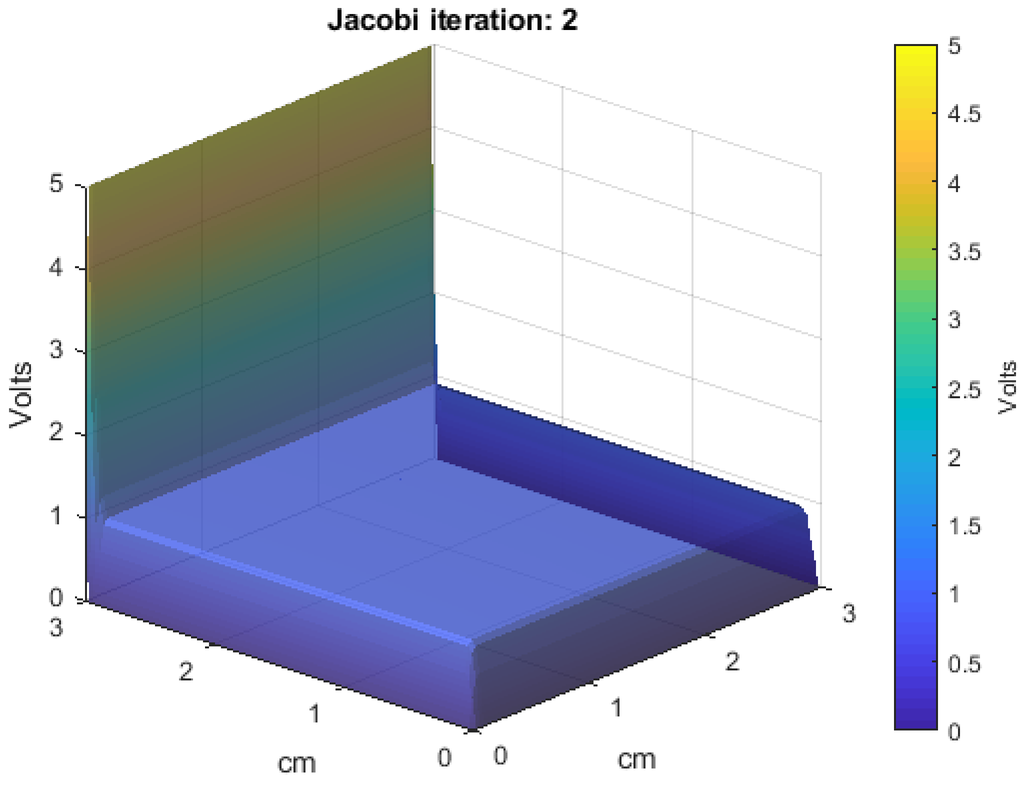

final SOR iteration: 5588, mesh size:400^2 steps
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

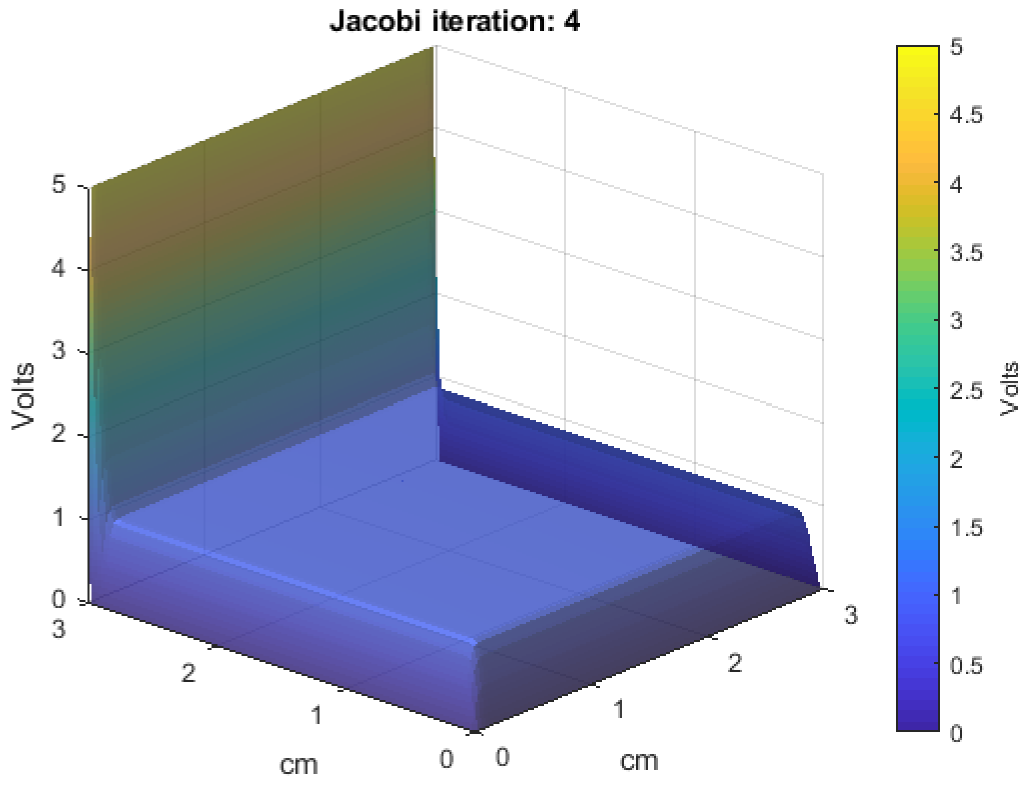
If one wants a general idea of the shape of the plot, a mesh dimension of 100x100 steps or beyond seems to offer a relatively detailed plot. Any smaller dimension gives coarse detail. I find 100x100 dimension favorable because it converges almost instantaneously using my university's remote server. As noted in Section 6, I'm using a formula to optimize my relaxation parameter, but I found that this formula breaks down for mesh dimensions higher than 300x300 steps. For these reasons, if one just wants a general idea of shape, choosing a mesh square with side dimension of 100 to 300 points seems like a good middle ground between quick results and quality results.

Section 5: A series of surface plots showing the evolution of the solution and how it progresses towards convergence

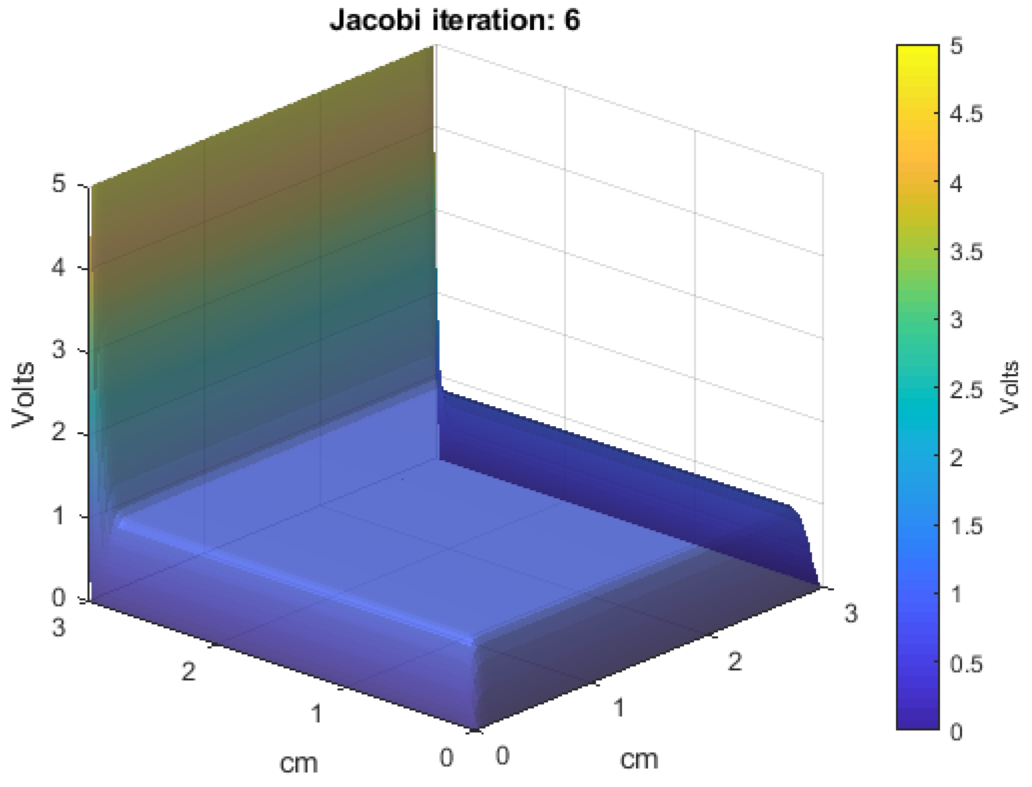
First sequence is for Jacobi method, Steps=100, Initial guess $\varphi=1$ for all internal points.


Jacobi iteration: 1
Volts
cm
cm
Volts
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

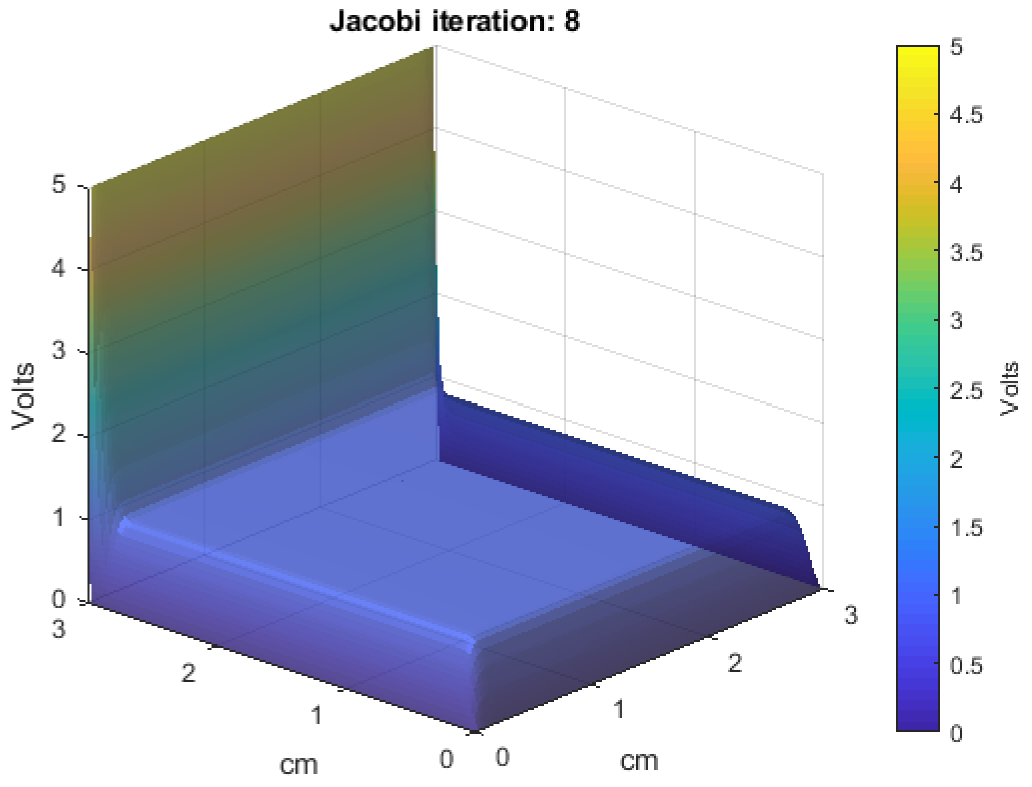

Jacobi iteration: 2
Volts
cm
cm
Volts
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

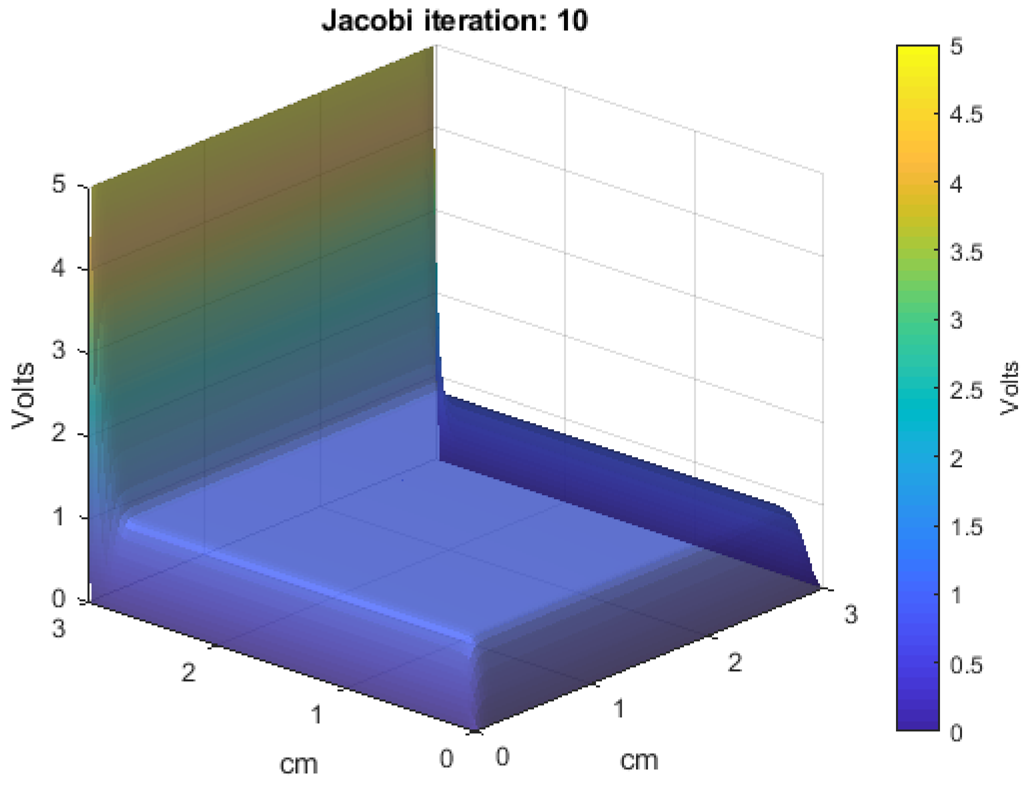


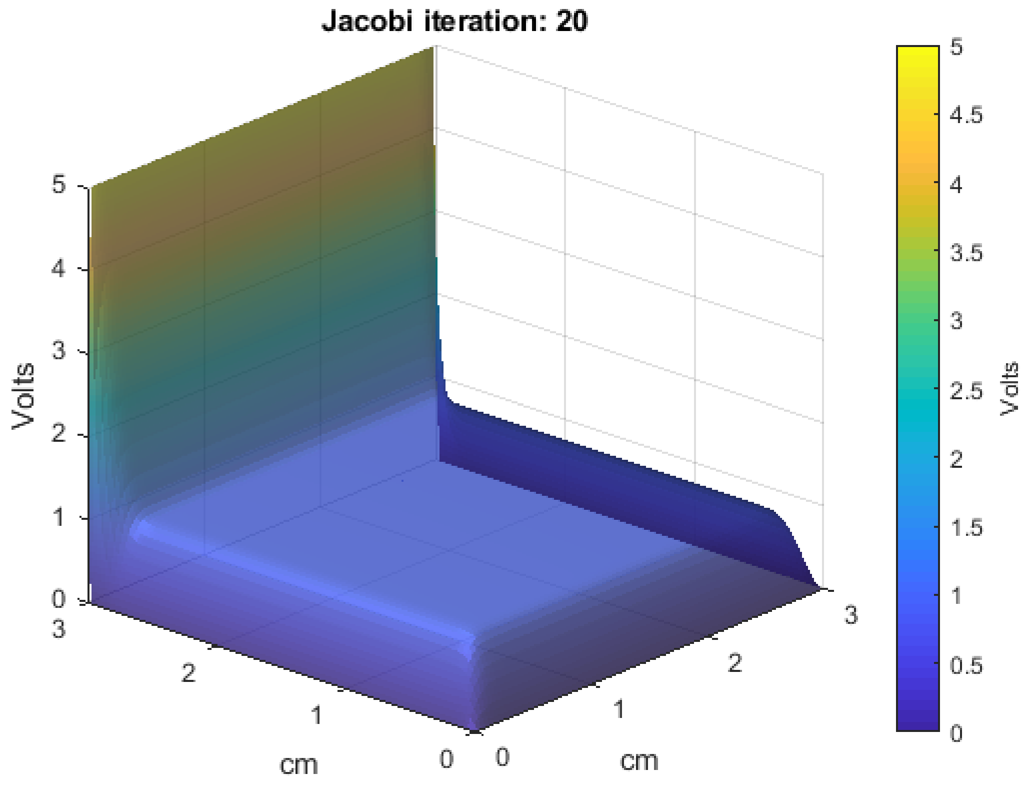
Jacobi iteration: 4
Volts
0
1
2
3
4
5
3
2
1
0
cm
0
1
2
3
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

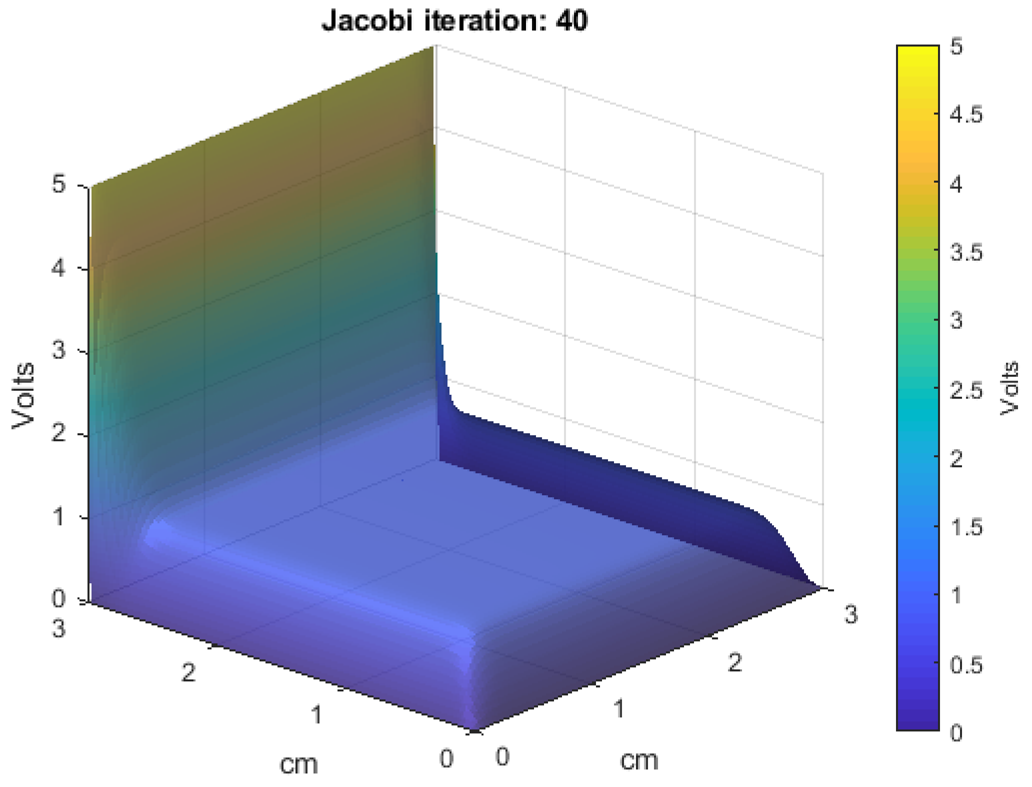


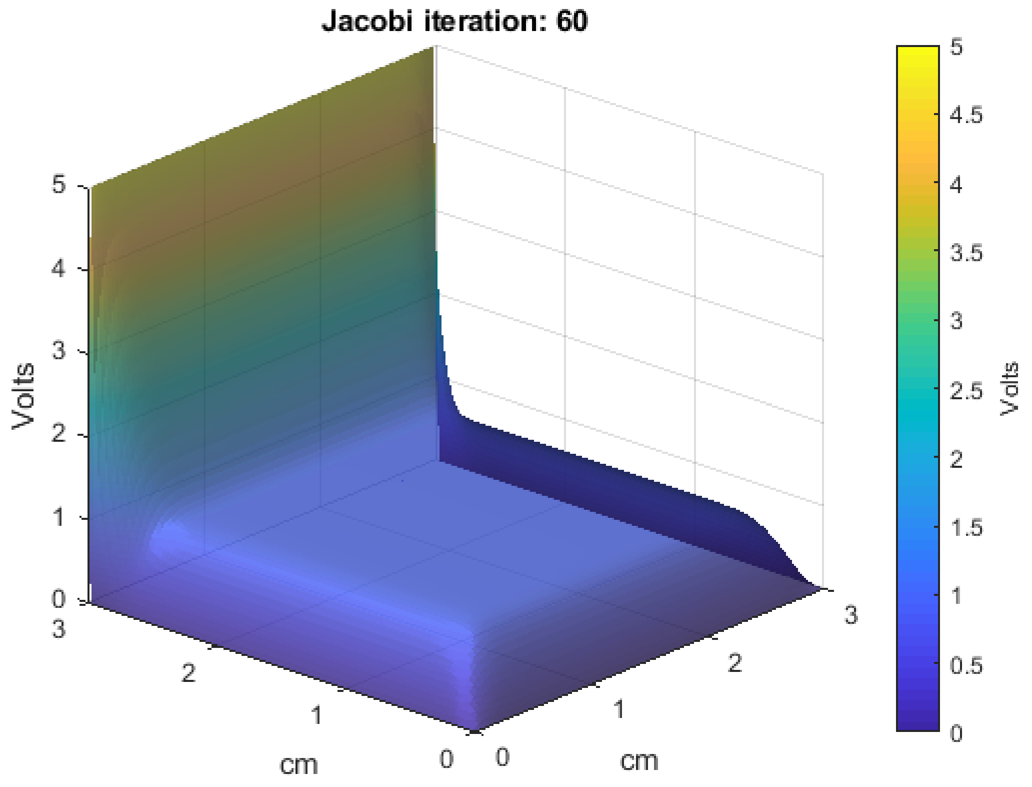
Jacobi iteration: 6
Volts
0
1
2
3
4
5
3
2
1
0
cm
0
1
2
3
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Jacobi iteration: 8
Volts
cm
cm
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Volts

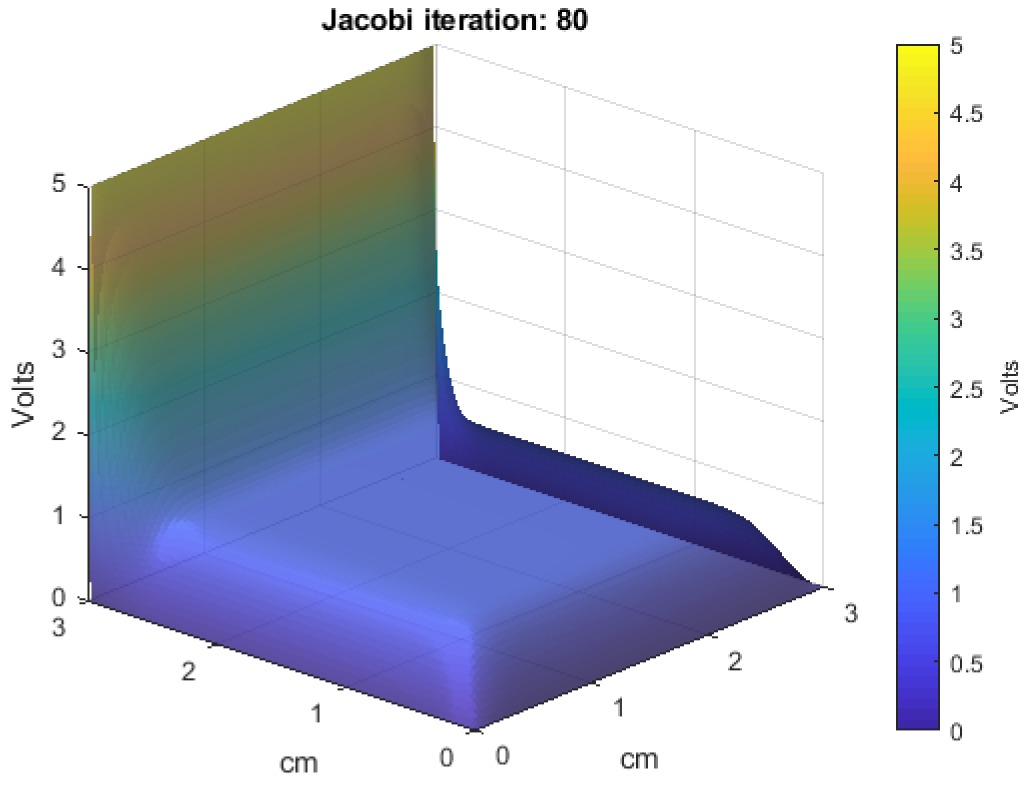

Jacobi iteration: 10
Volts
cm
cm
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
Volts


Jacobi iteration: 20
Volts
cm
cm
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

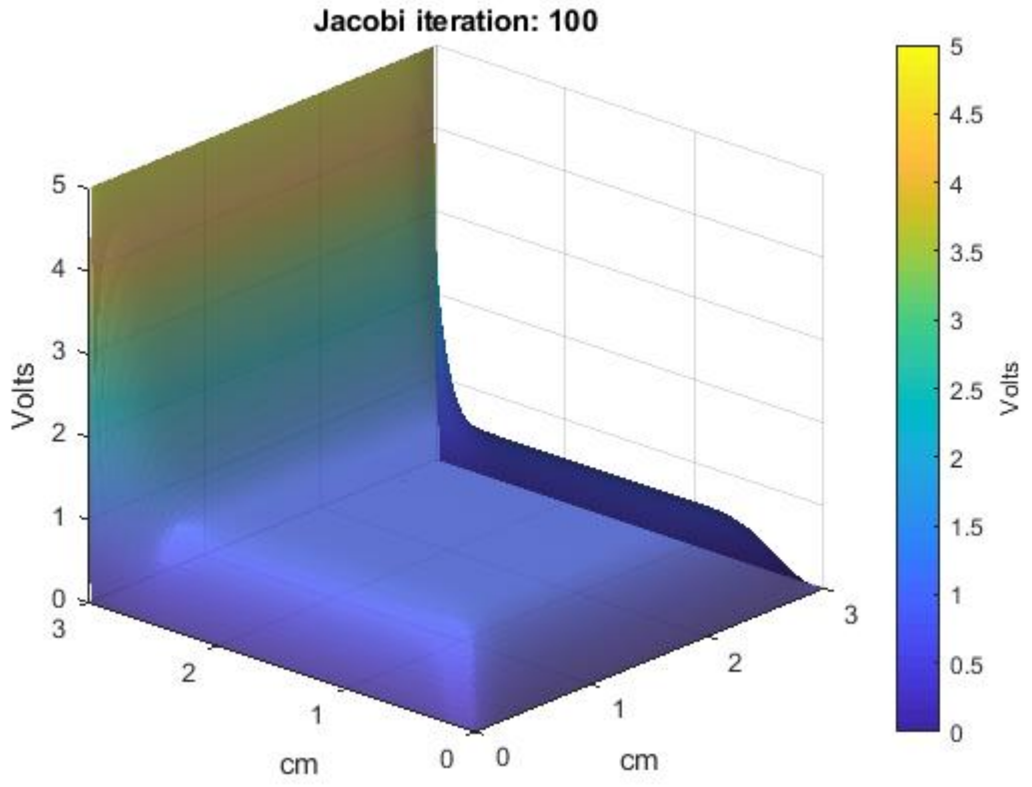

Jacobi iteration: 40
Volts
cm
cm
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

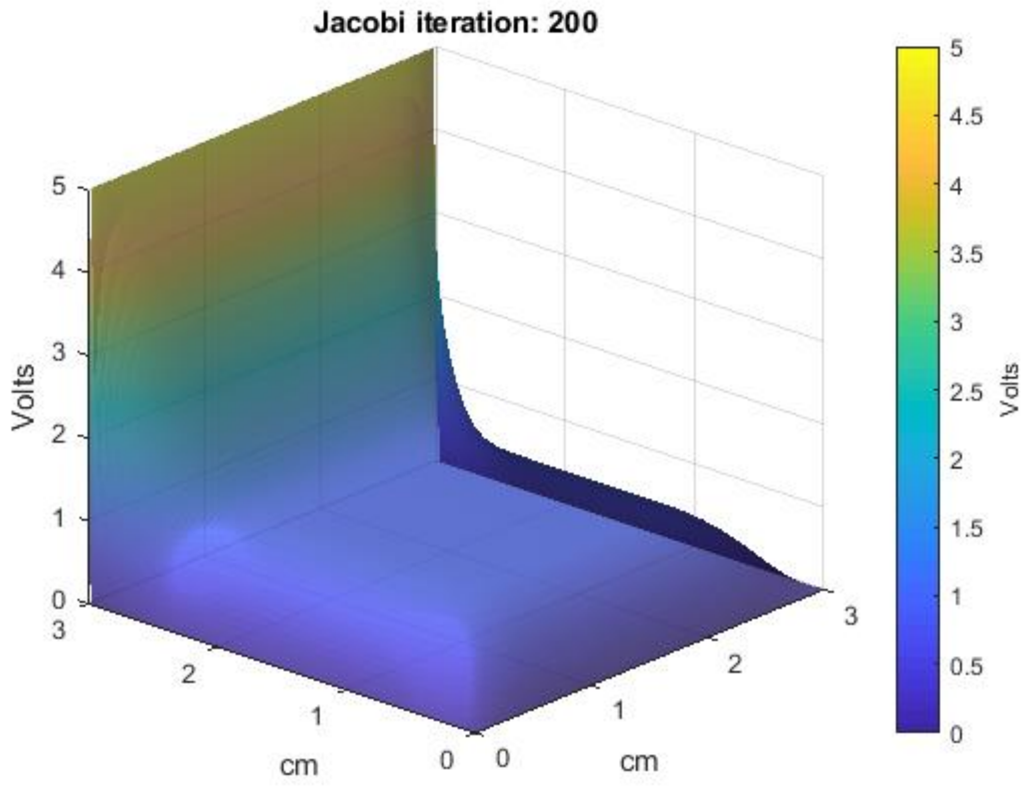


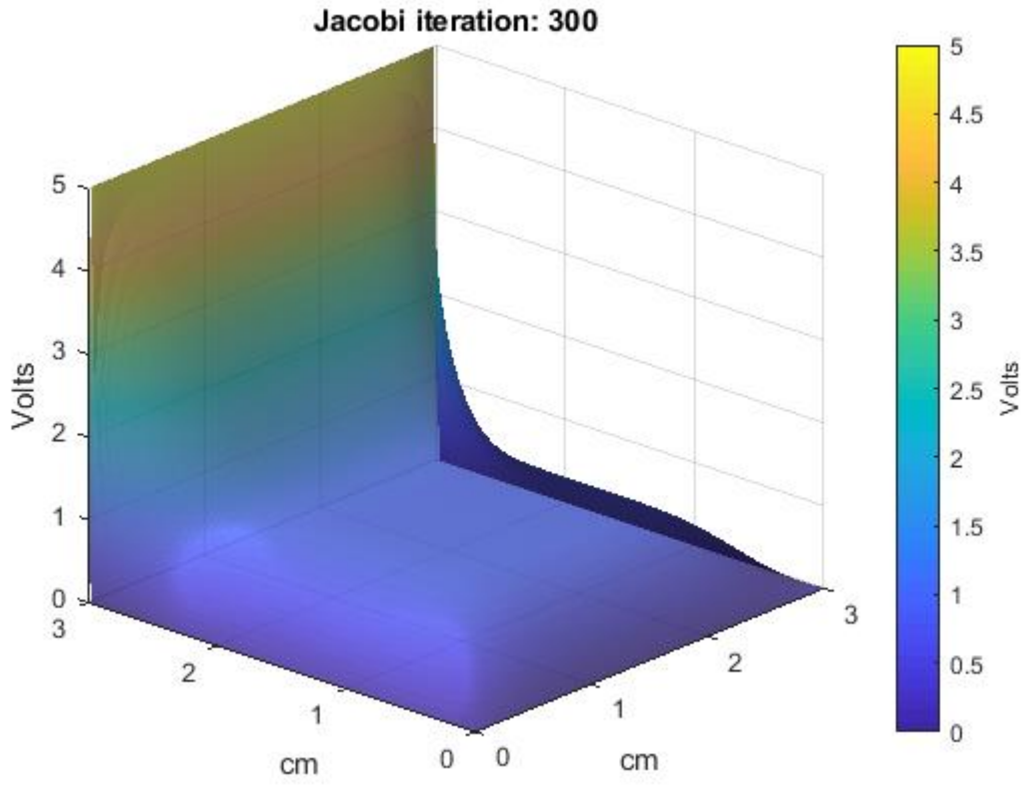
Jacobi iteration: 60
Volts
cm
cm
Volts
0
1
2
3
4
5
0.5
1.5
2.5
3.5
4.5



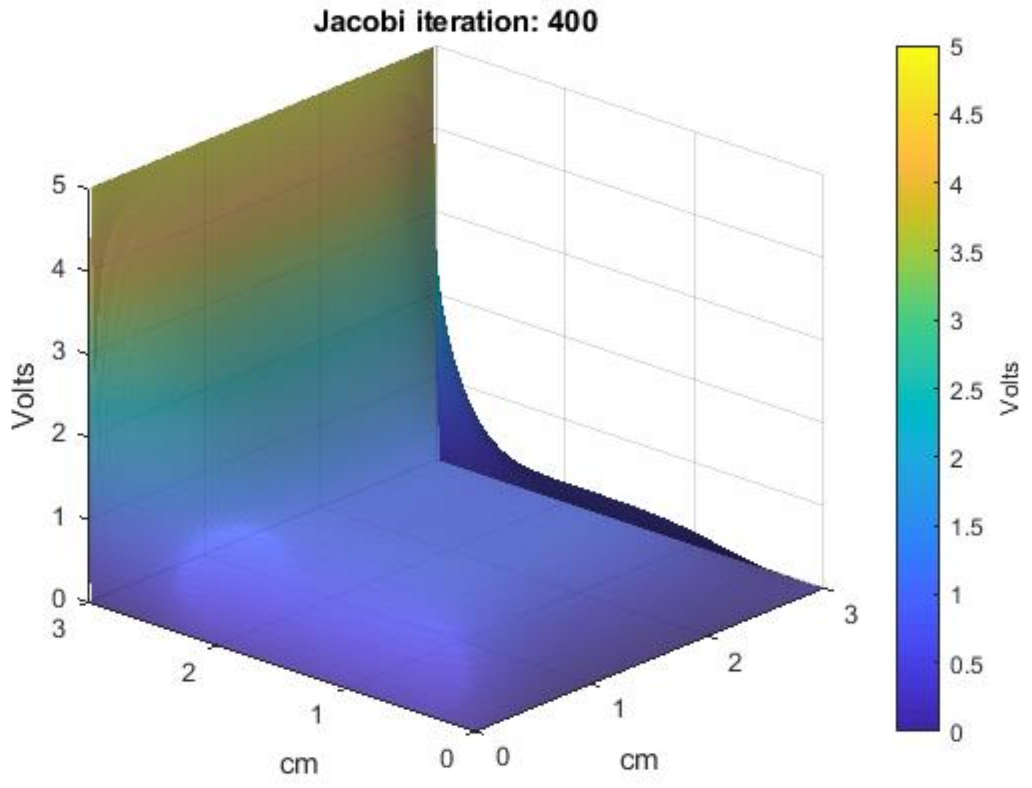
Jacobi iteration: 80
Volts
cm
cm
Volts
0
1
2
3
4
5
0.5
1.5
2.5
3.5
4.5


Jacobi iteration: 100
Volts
cm
cm
Volts
0
1
2
3
4
5
0.5
1.5
2.5
3.5
4.5

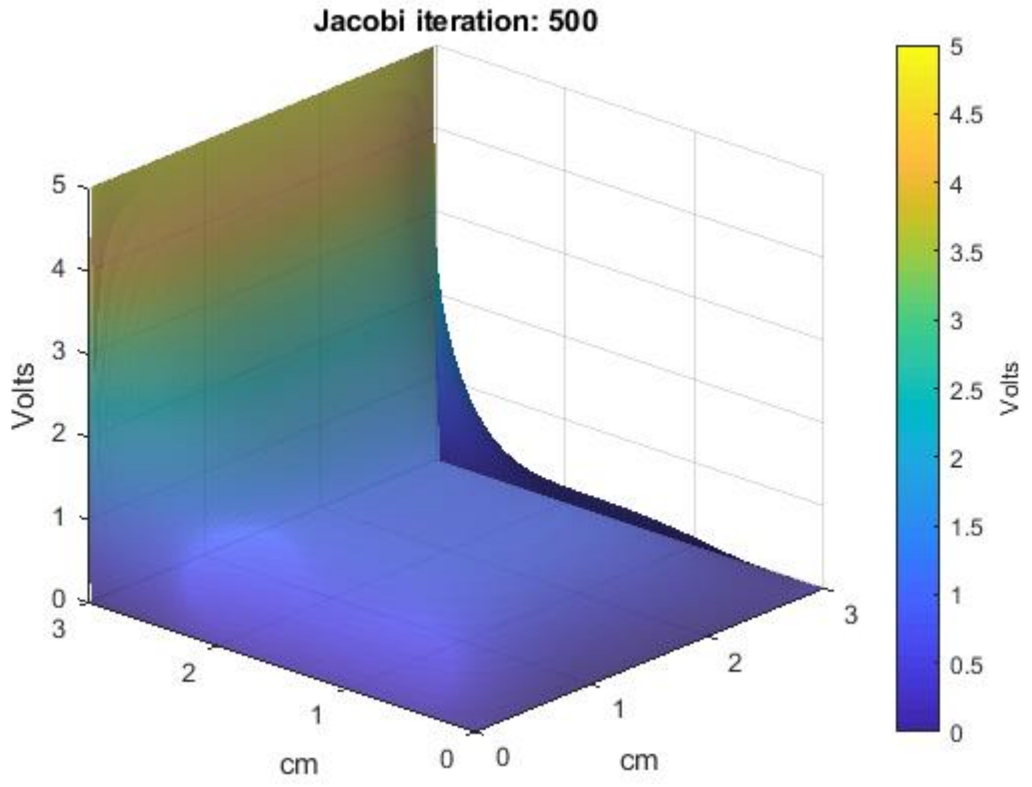

Jacobi iteration: 200
Volts
cm
cm
Volts
0
1
2
3
4
5
0.5
1.5
2.5
3.5
4.5

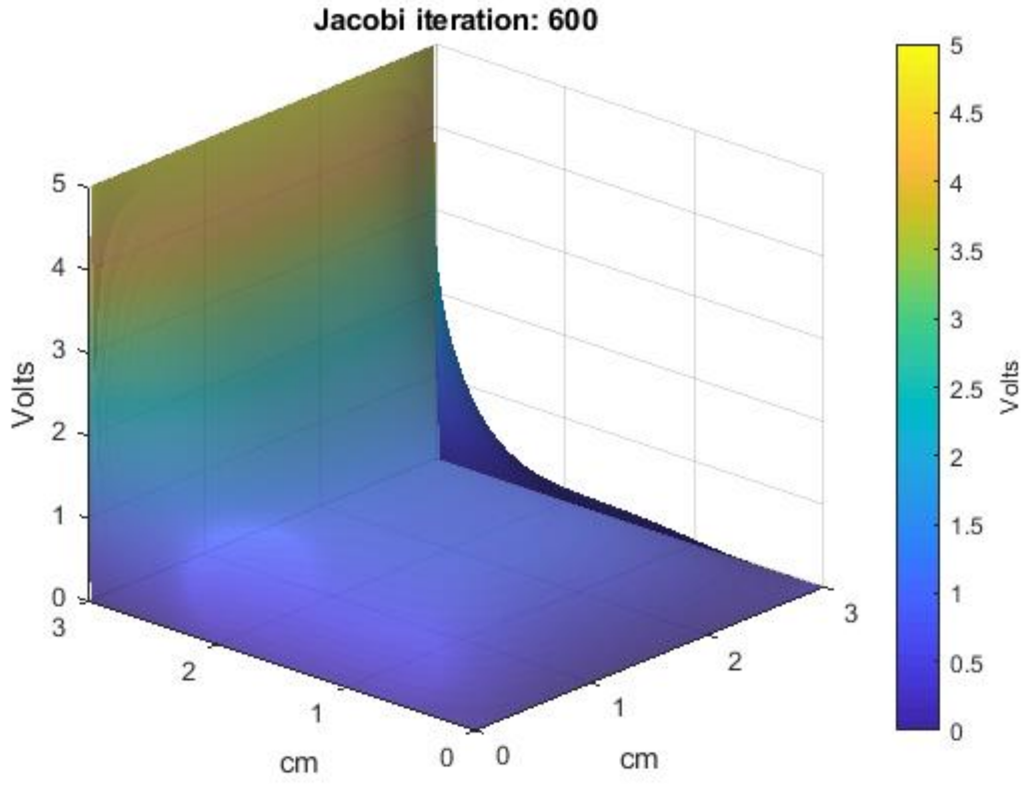


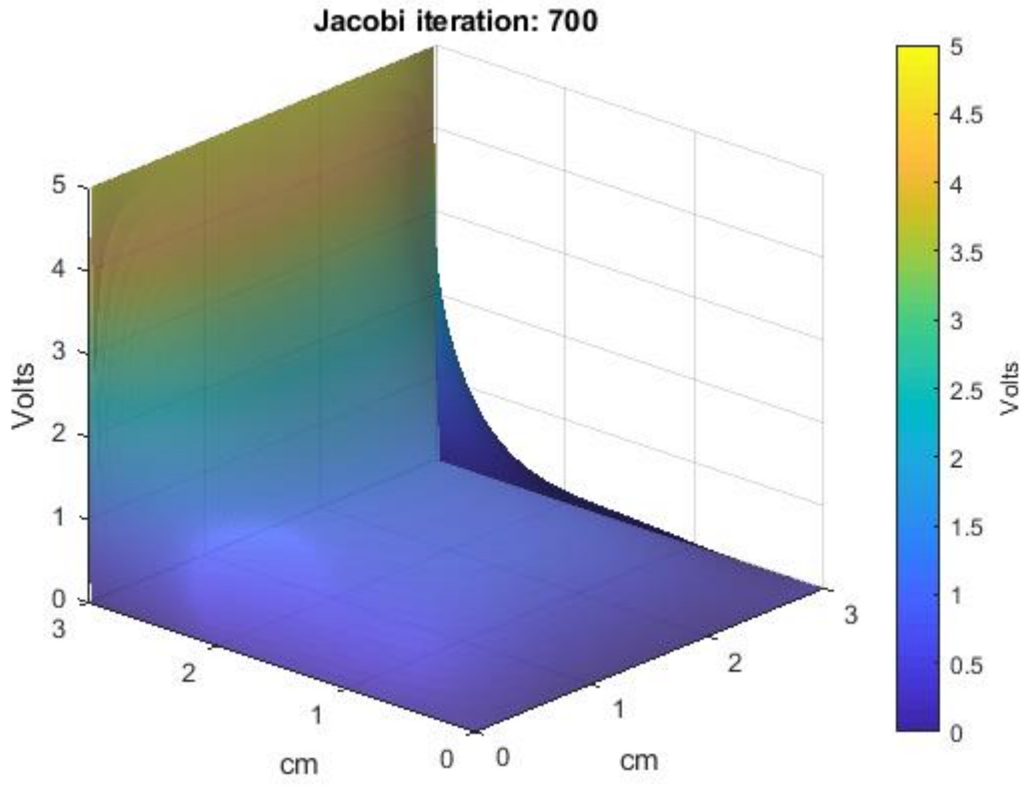
Jacobi iteration: 300
Volts
cm
cm
Volts

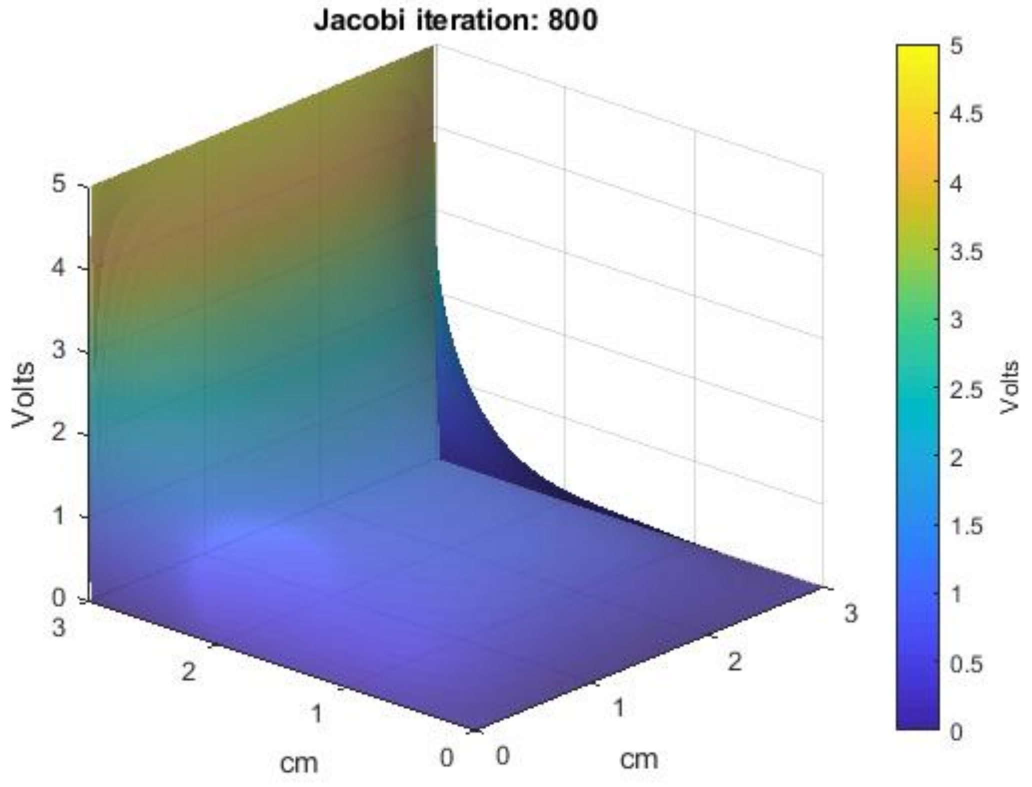


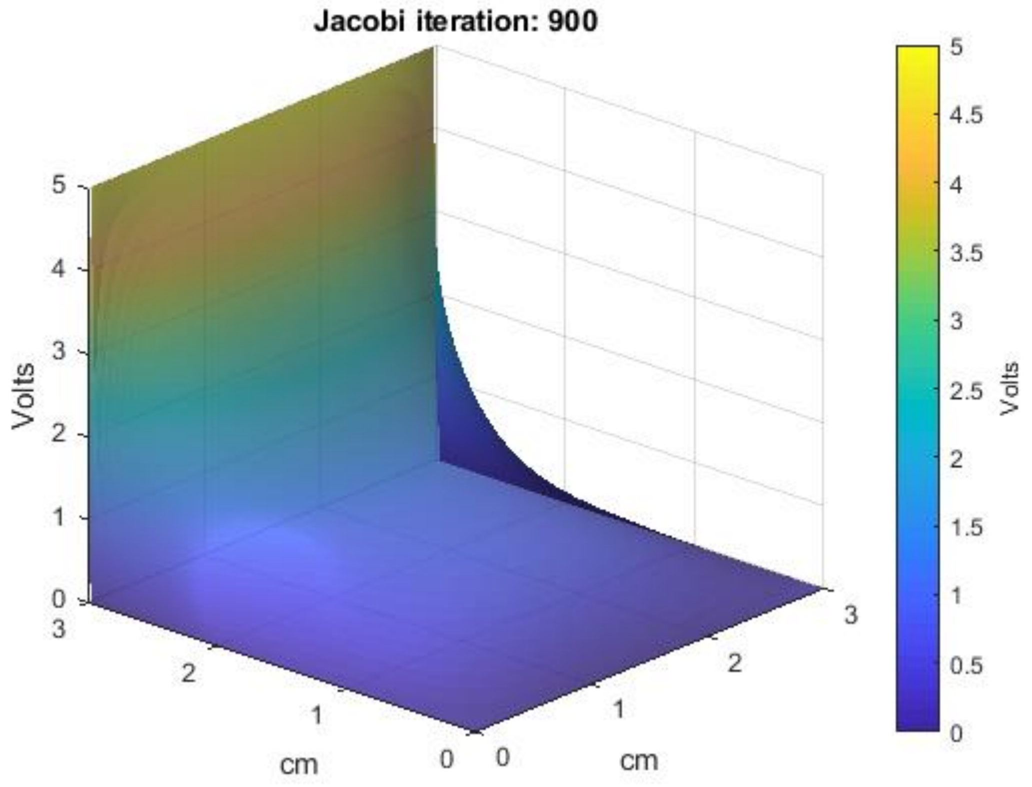
Jacobi iteration: 400
Volts
cm
cm
Volts


Jacobi iteration: 500
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

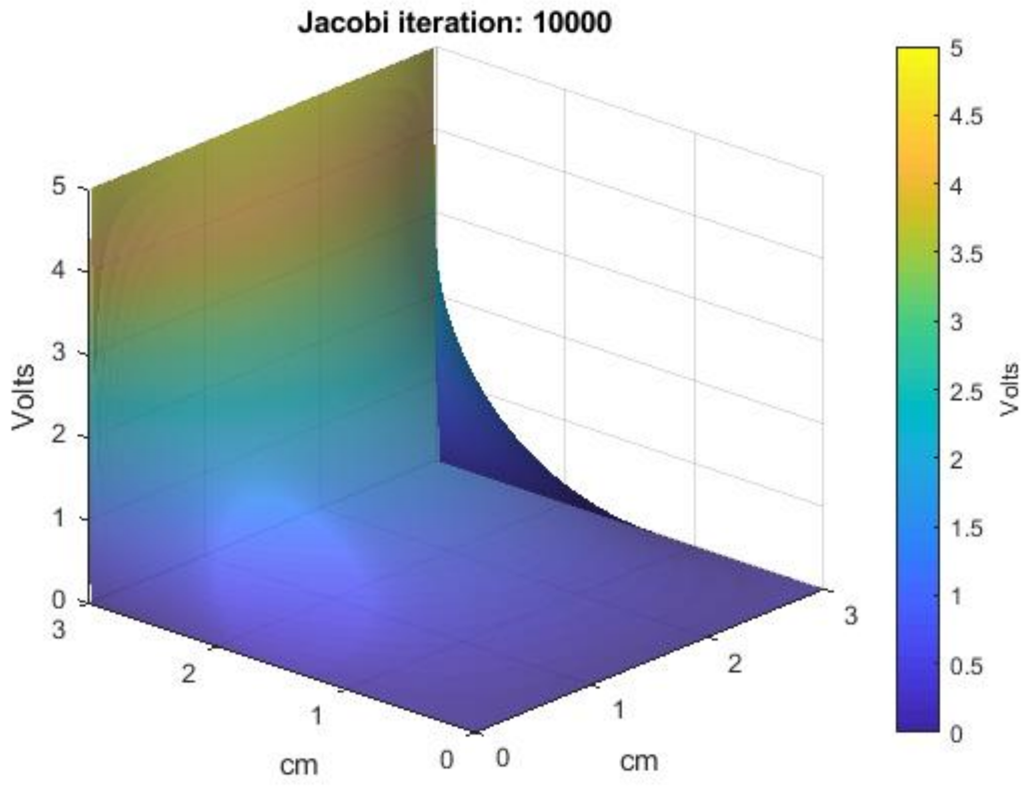

Jacobi iteration: 600
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


Jacobi iteration: 700
Volts
cm
cm
0
1
2
3
4
5
Volts

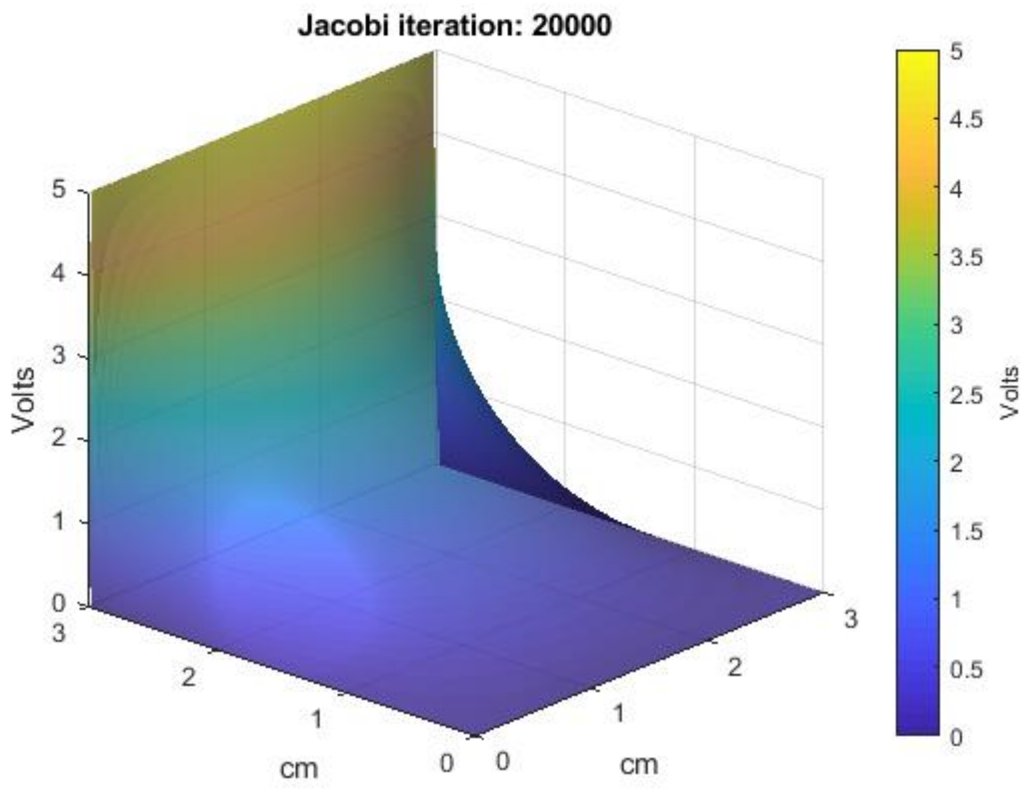

Jacobi iteration: 800
Volts
cm
cm
0
1
2
3
4
5
Volts



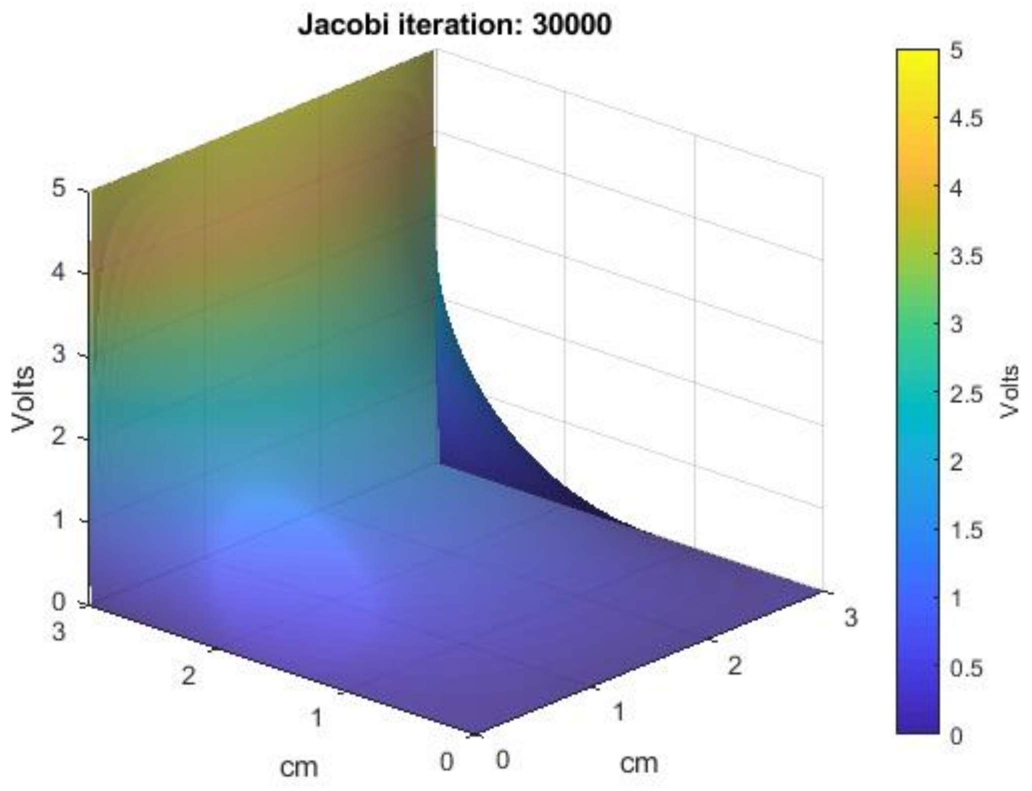
Jacobi iteration: 900
Volts
cm
cm
Volts



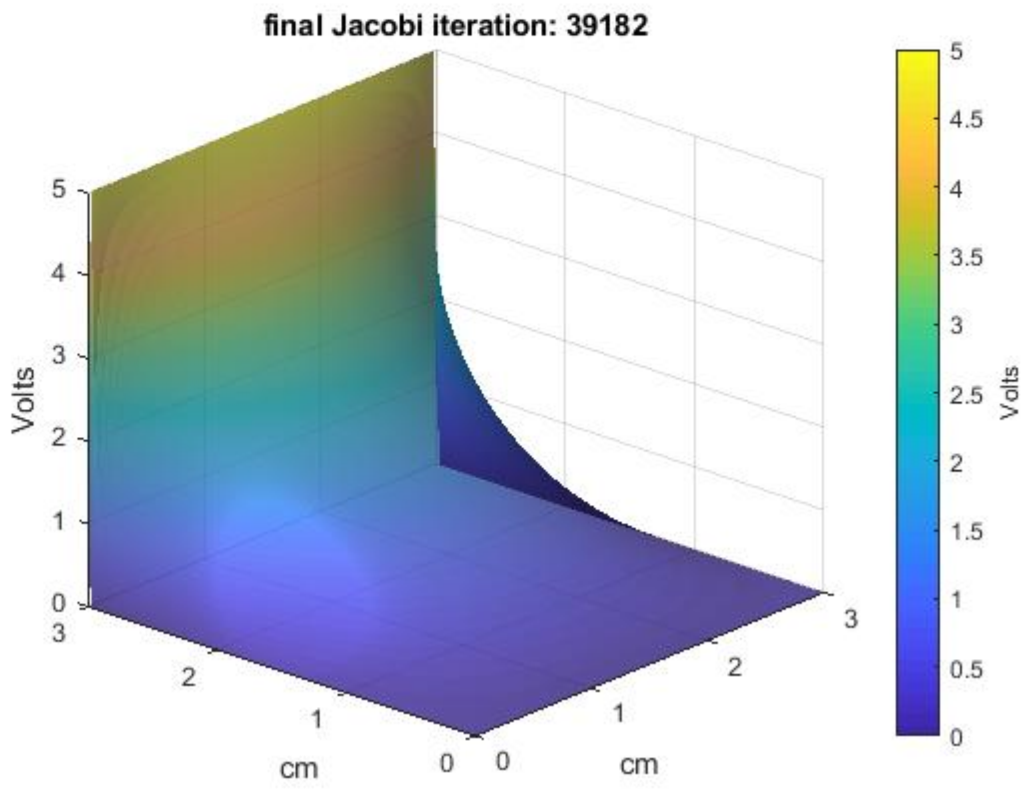
Jacobi iteration: 10000
Volts
cm
cm
Volts



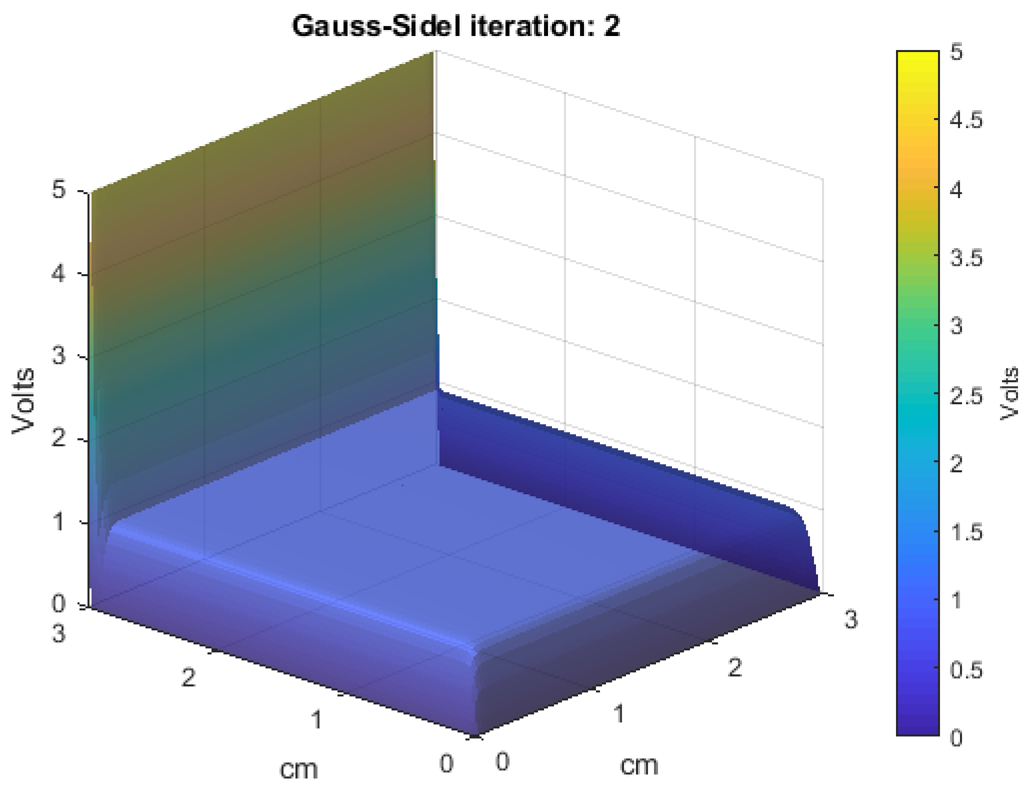
Jacobi iteration: 20000
Volts
cm
cm
Volts

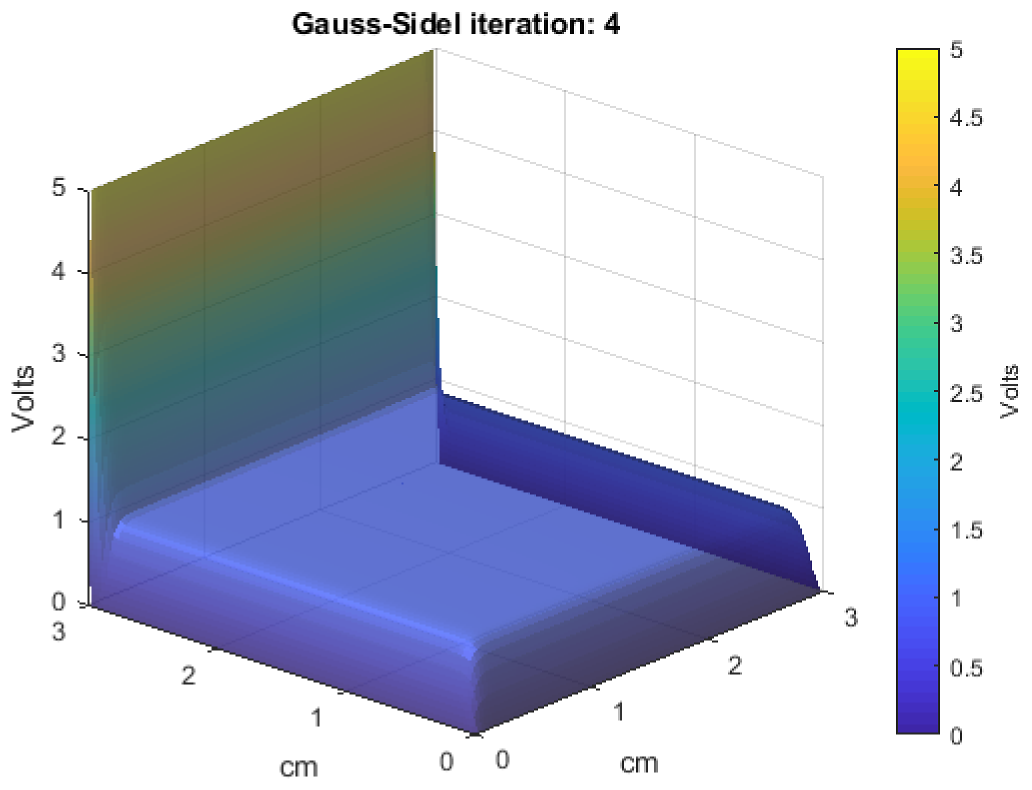


Jacobi iteration: 30000
Volts
cm
cm
Volts

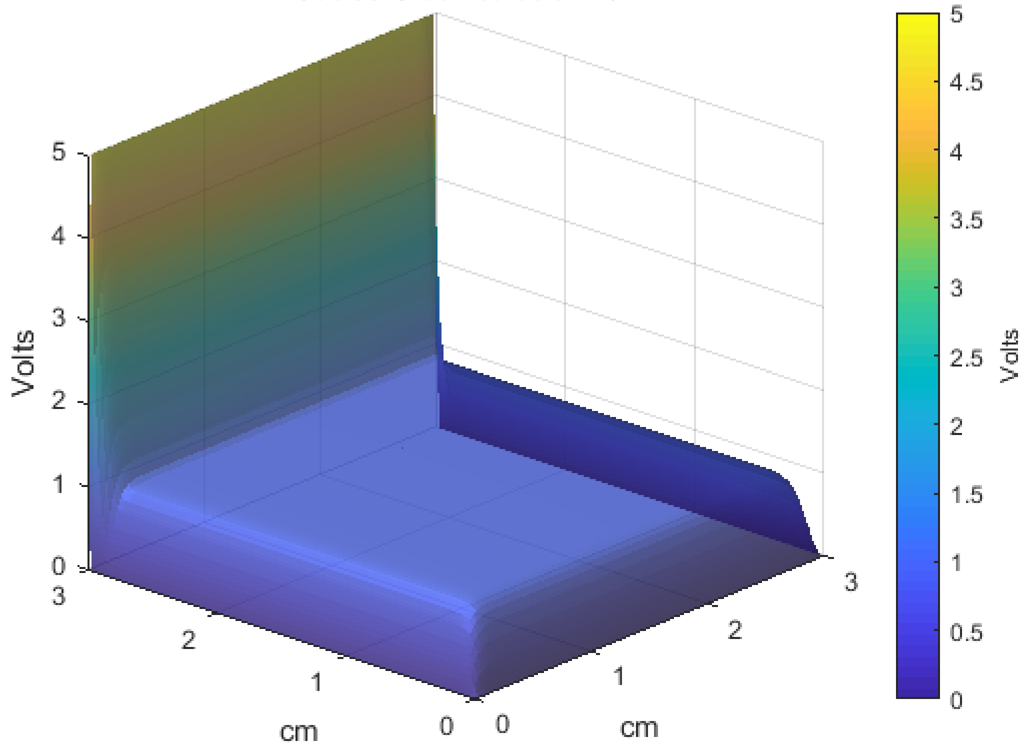

final Jacobi iteration: 39182
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Next is Gauss-Seidel, mesh dimensions are 100x100

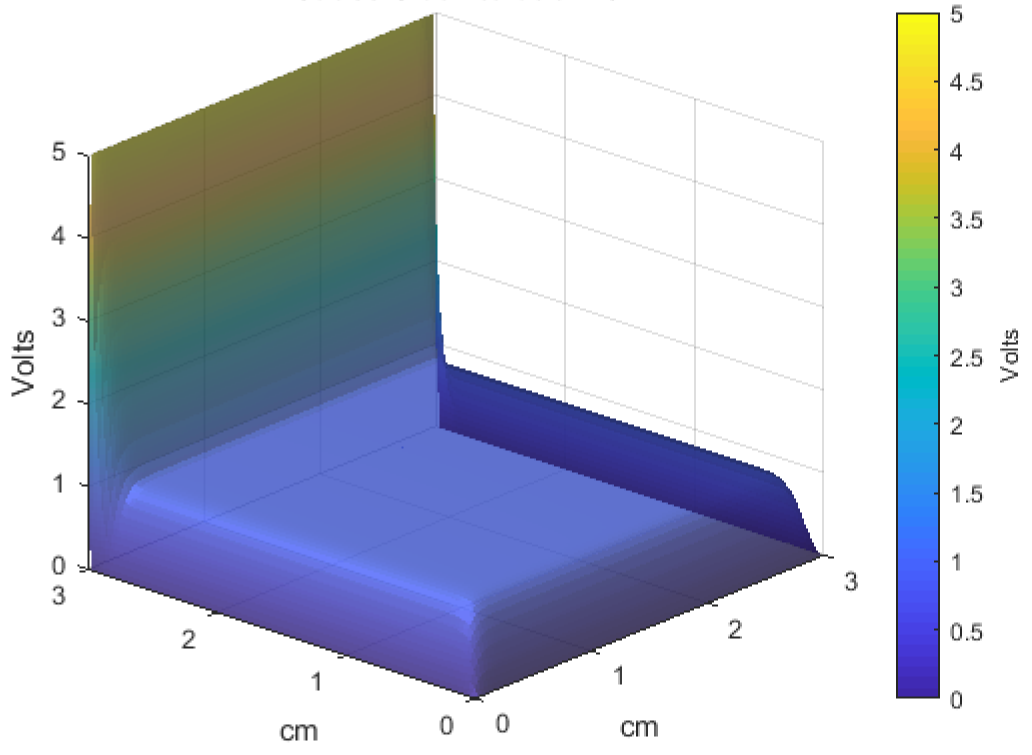


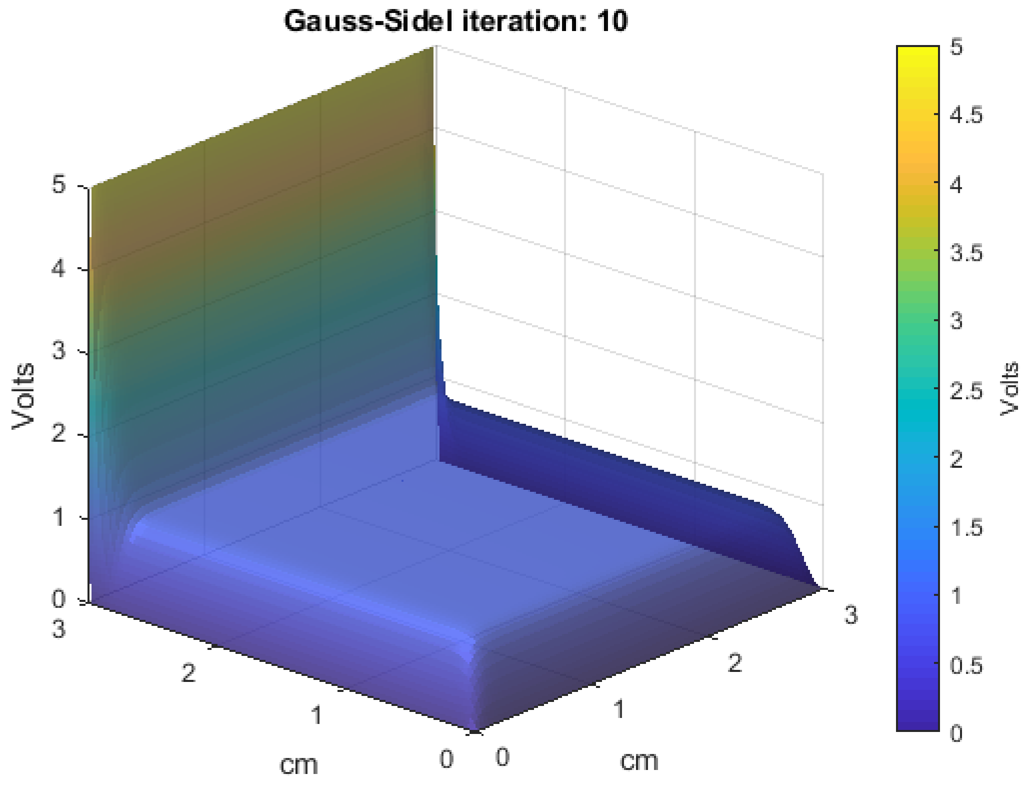


Gauss-Sidel iteration: 6

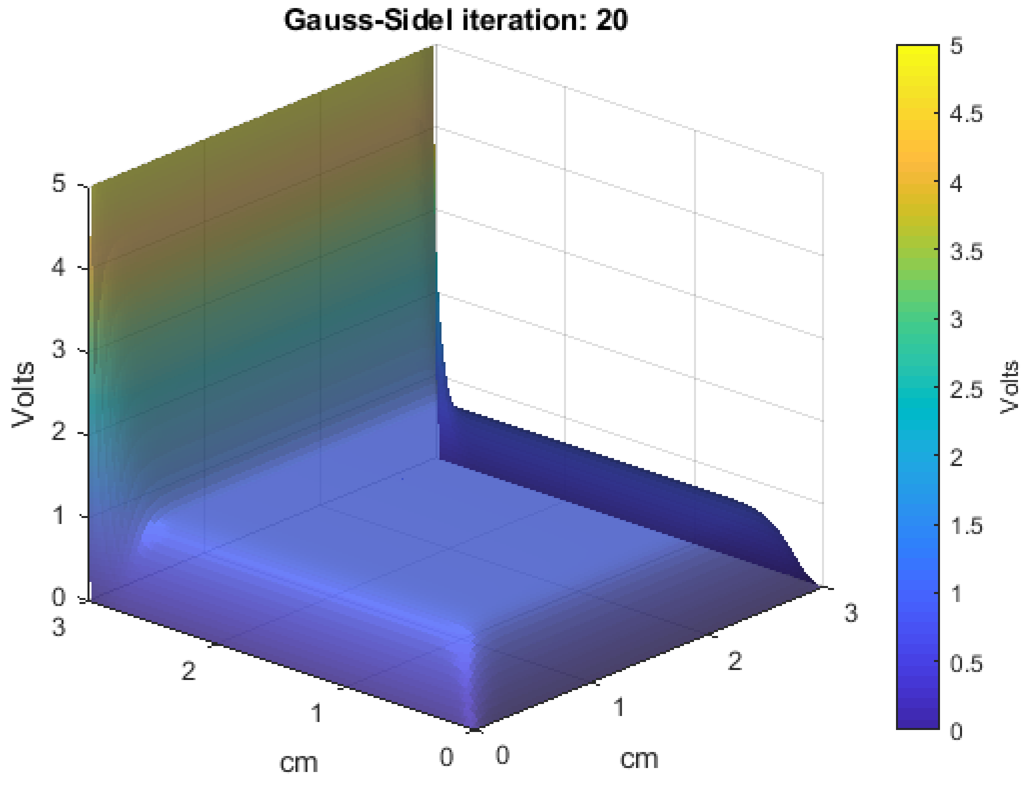


Gauss-Sidel iteration: 8



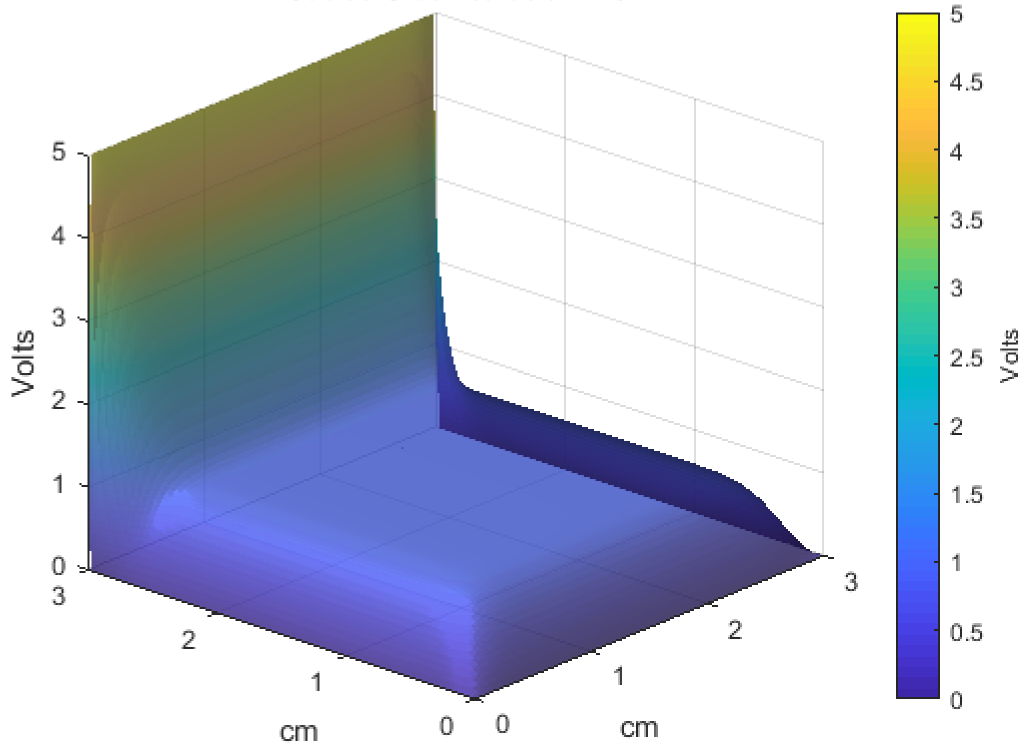


Gauss-Sidel iteration: 10
Volts
cm
cm
Volts

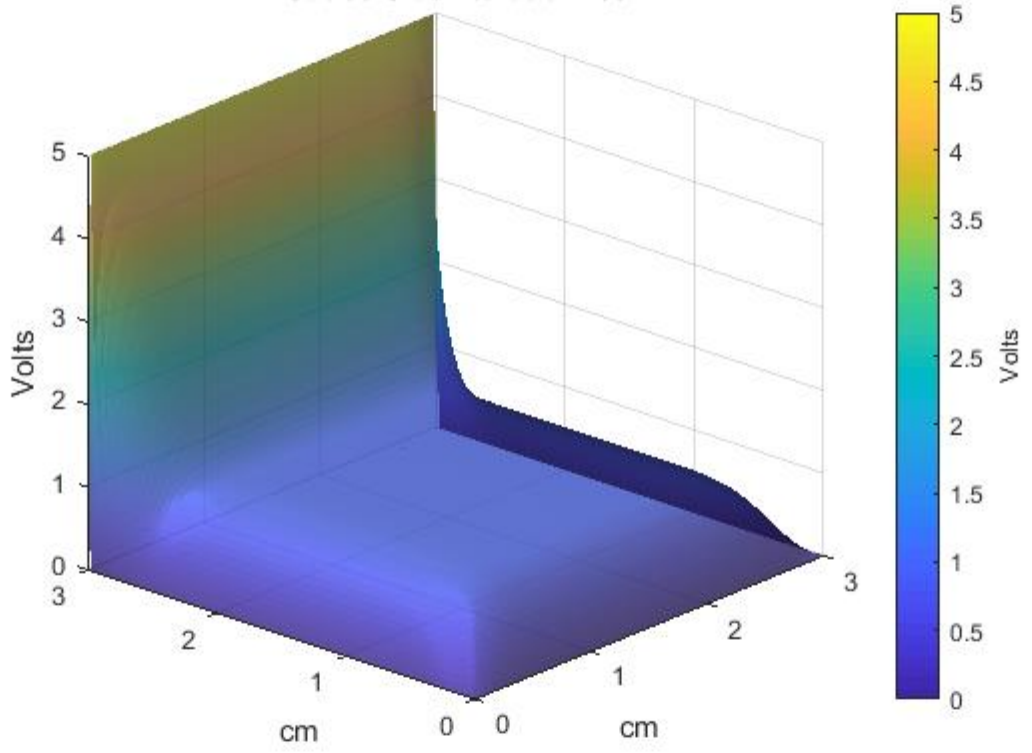


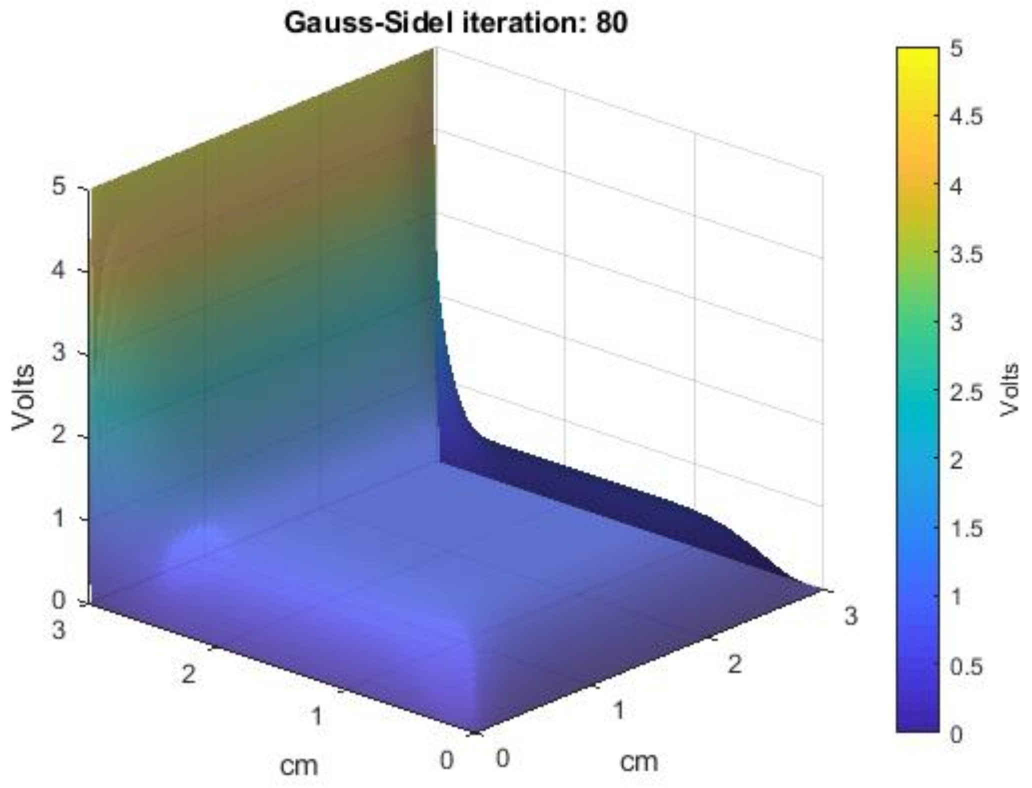
Gauss-Sidel iteration: 20
Volts
cm
cm
Volts

Gauss-Sidel iteration: 40

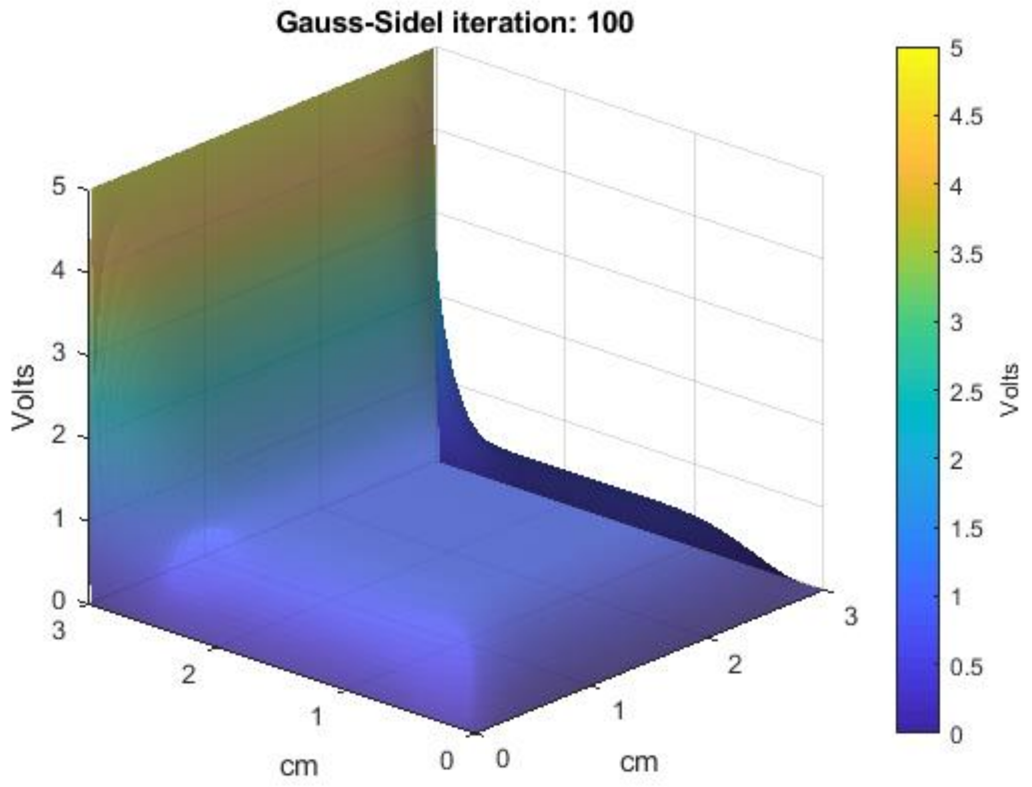


Gauss-Sidel iteration: 60

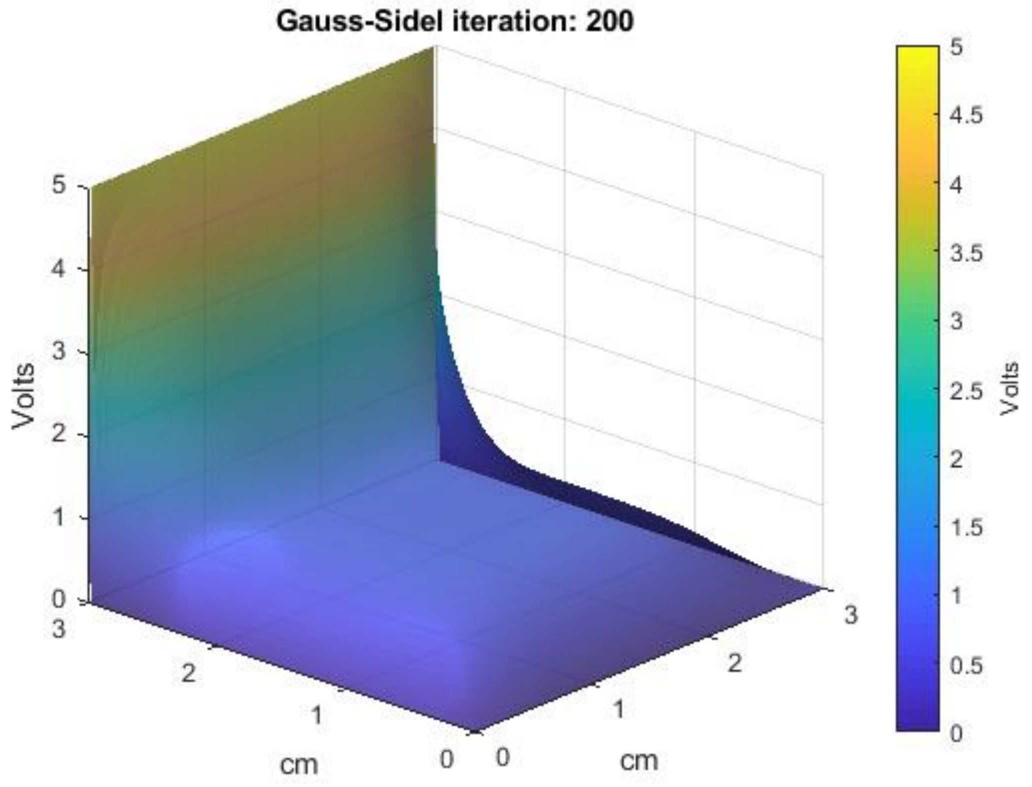




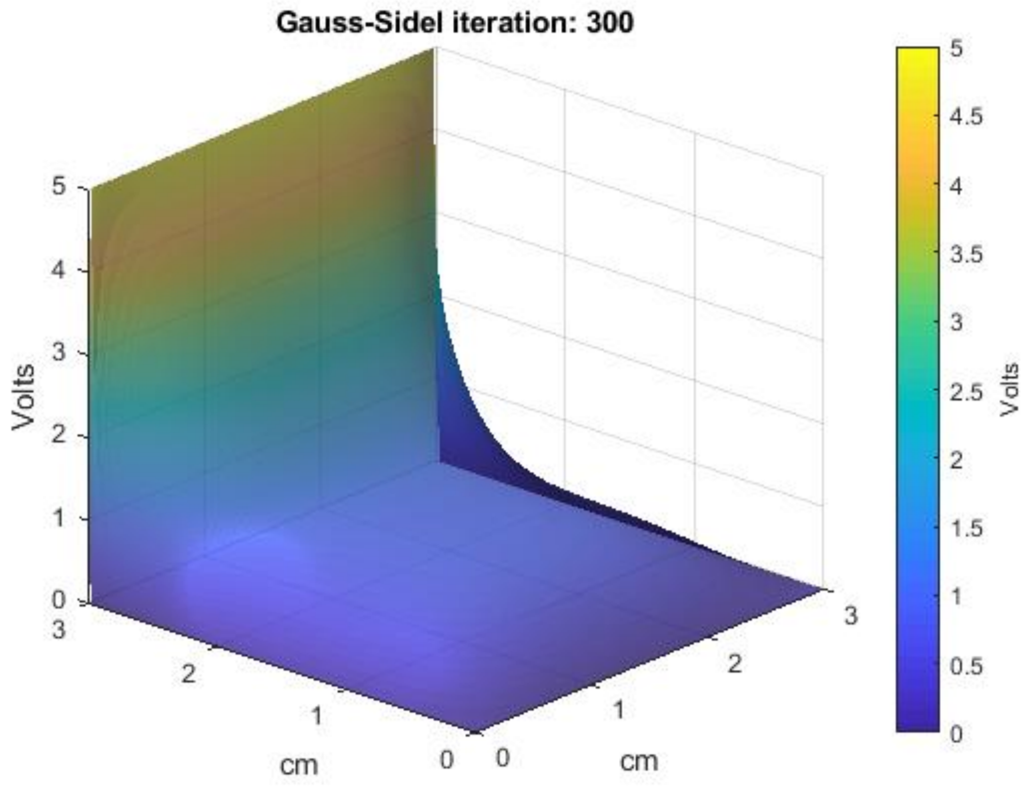
Gauss-Sidel iteration: 80
Volts
cm
cm
Volts



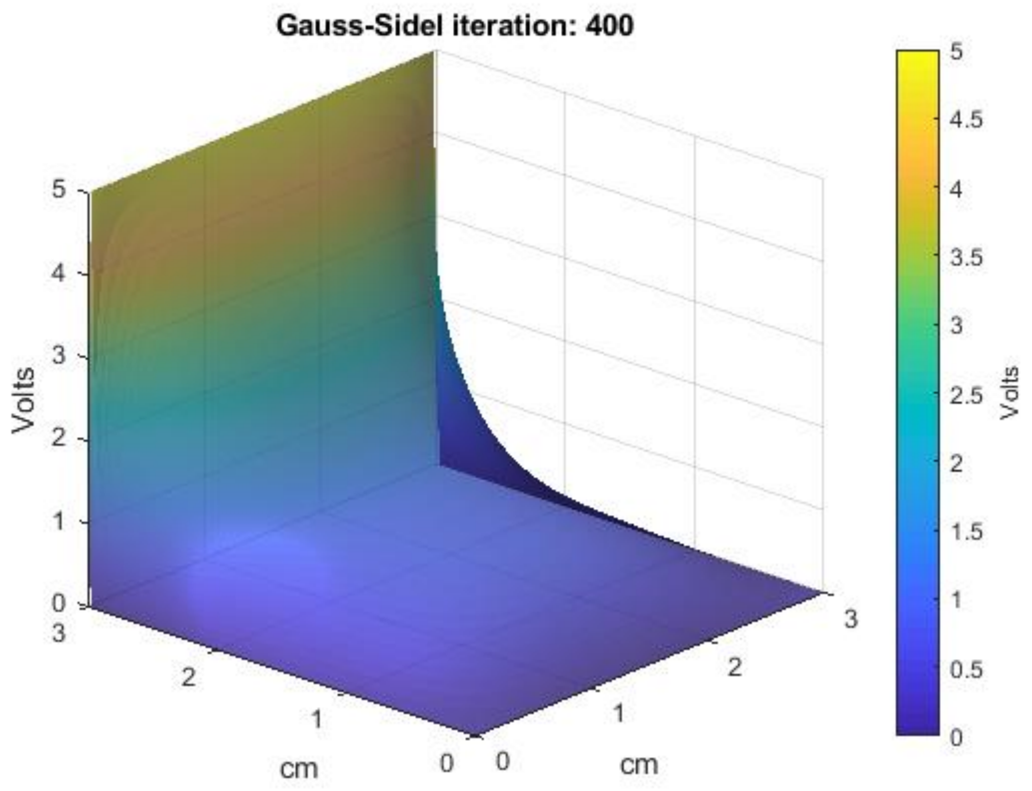
Gauss-Sidel iteration: 100
Volts
cm
cm
Volts



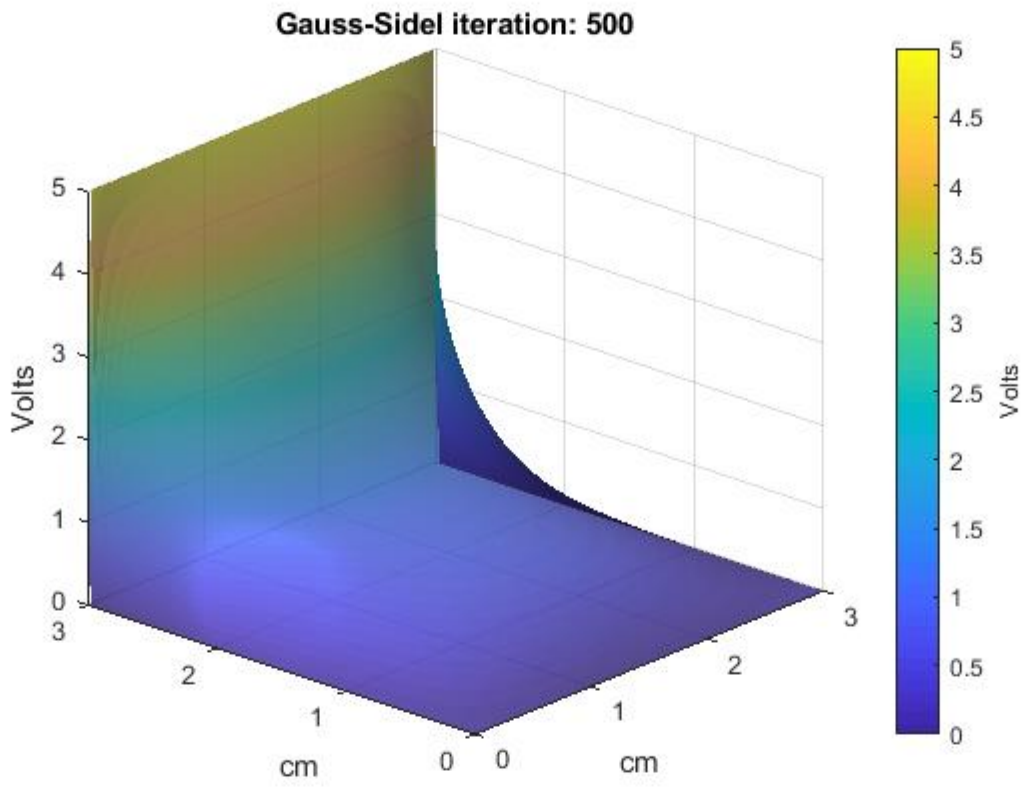
Gauss-Sidel iteration: 200
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



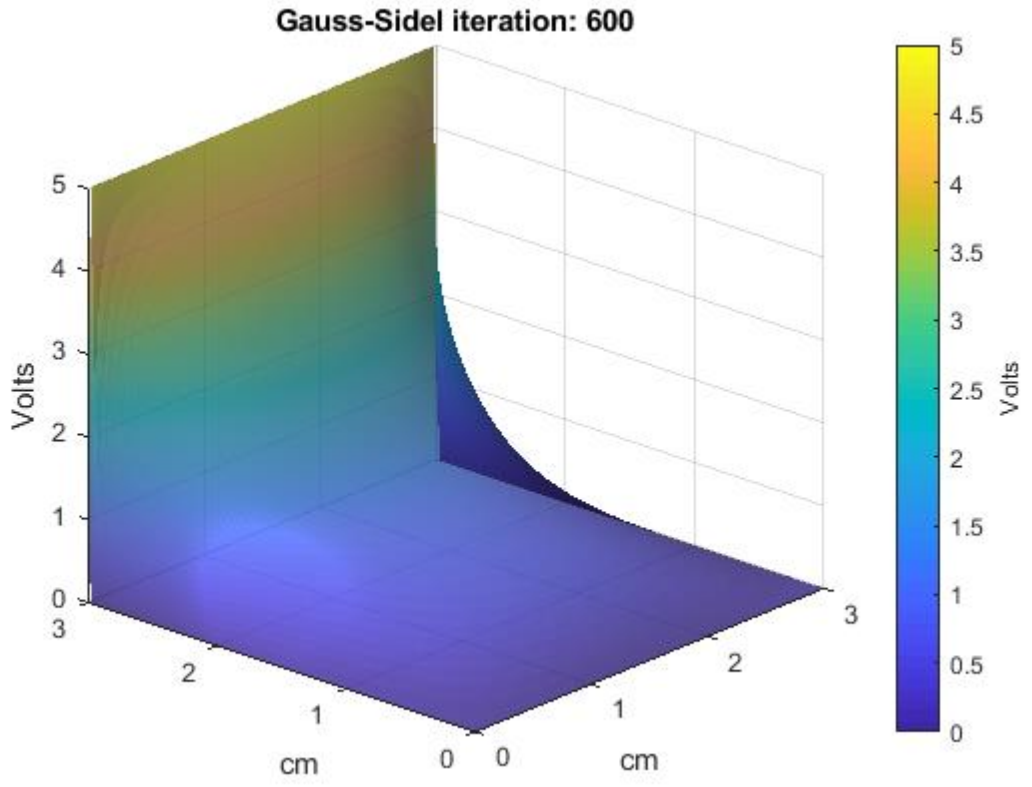
Gauss-Sidel iteration: 300
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



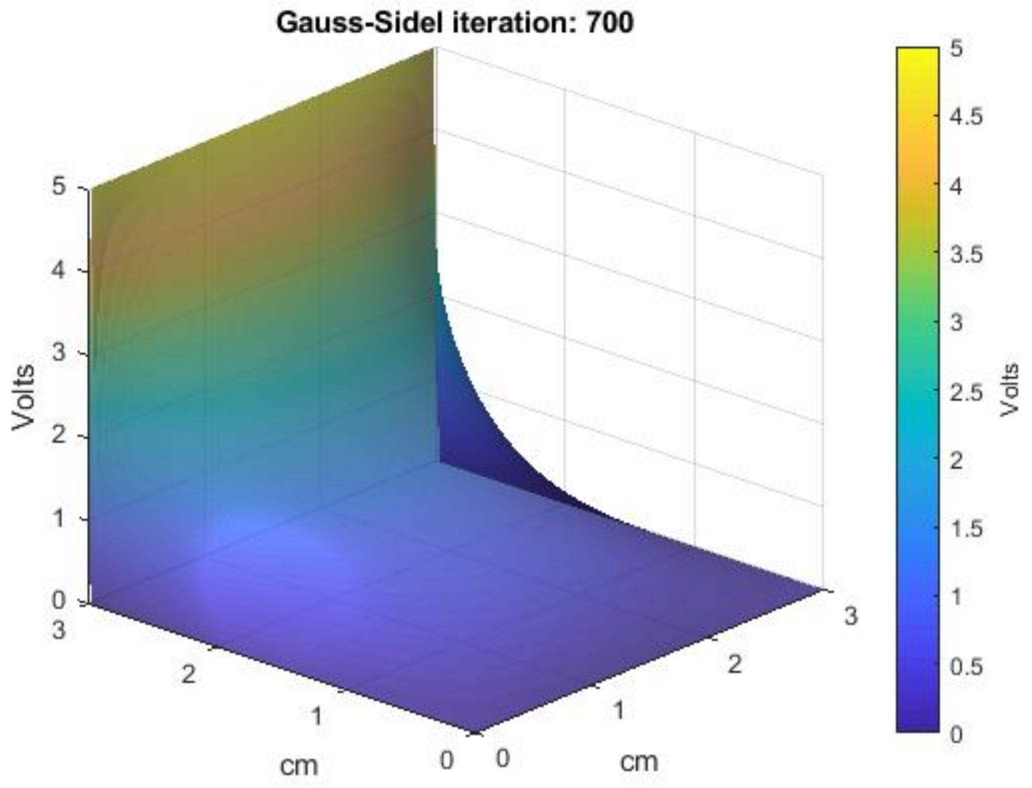
Gauss-Sidel iteration: 400
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5



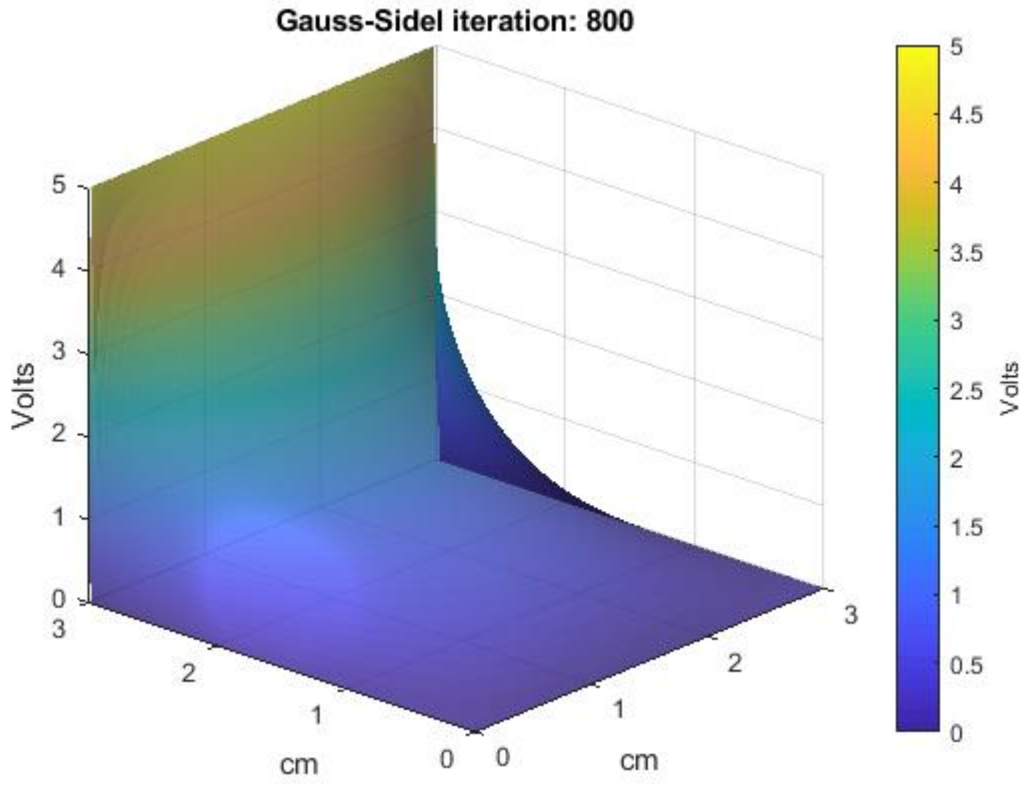
Gauss-Sidel iteration: 500
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5

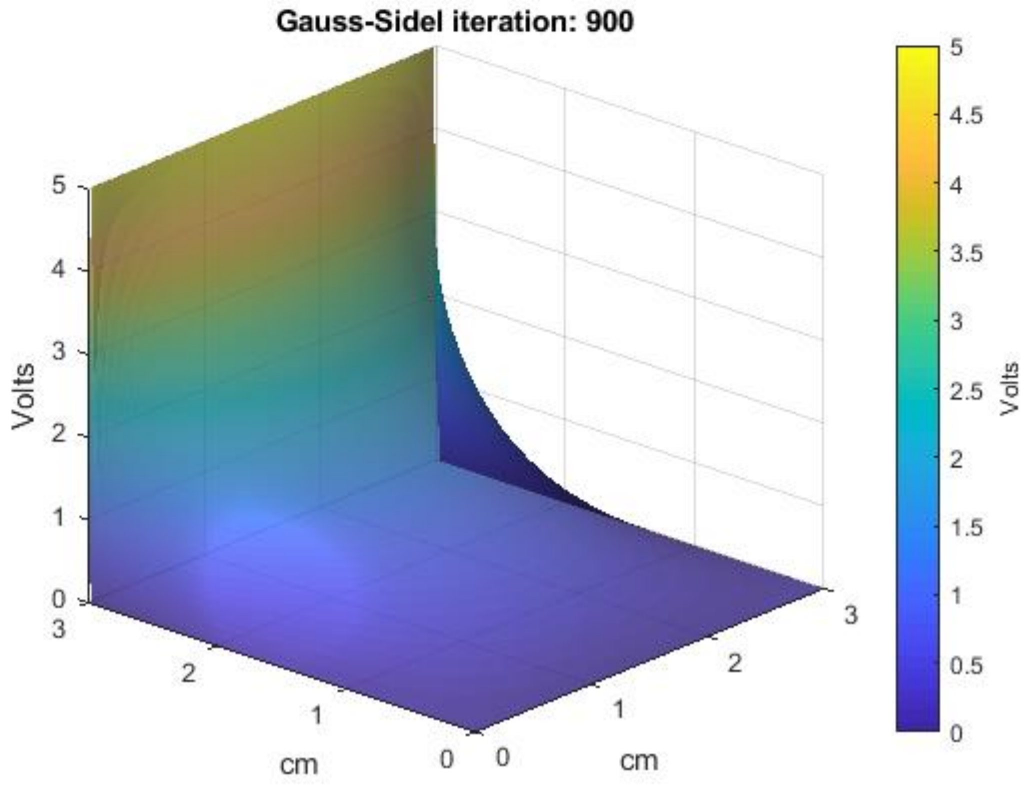


Gauss-Sidel iteration: 600
Volts
cm
cm
Volts

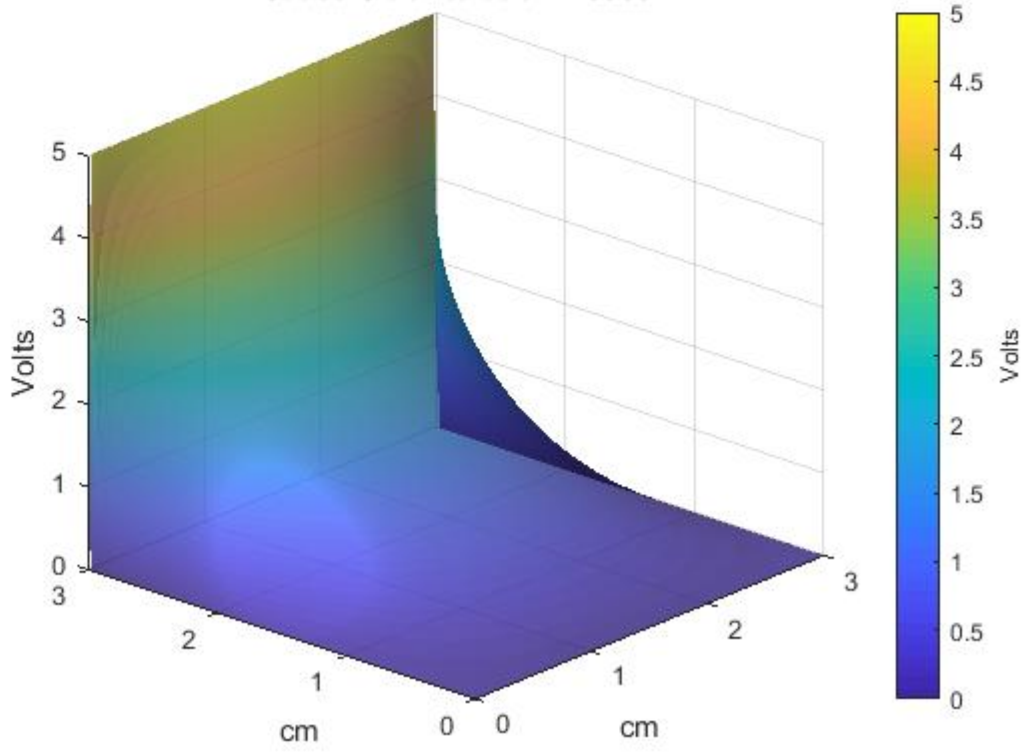


Gauss-Sidel iteration: 700
Volts
cm
cm
Volts

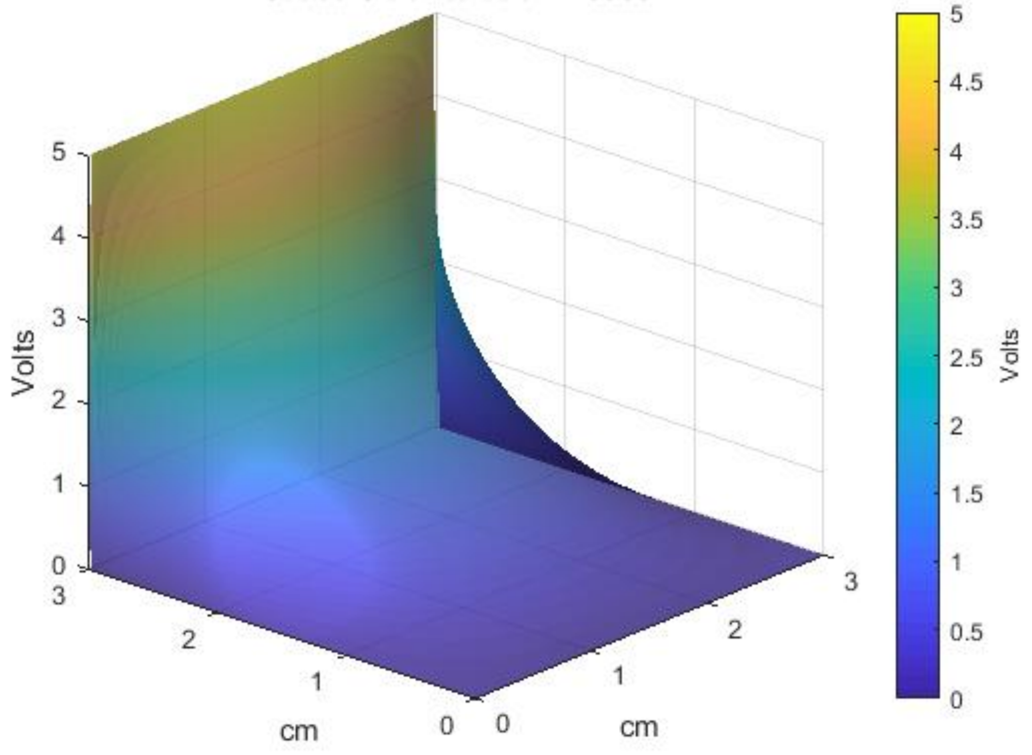

Gauss-Sidel iteration: 800
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

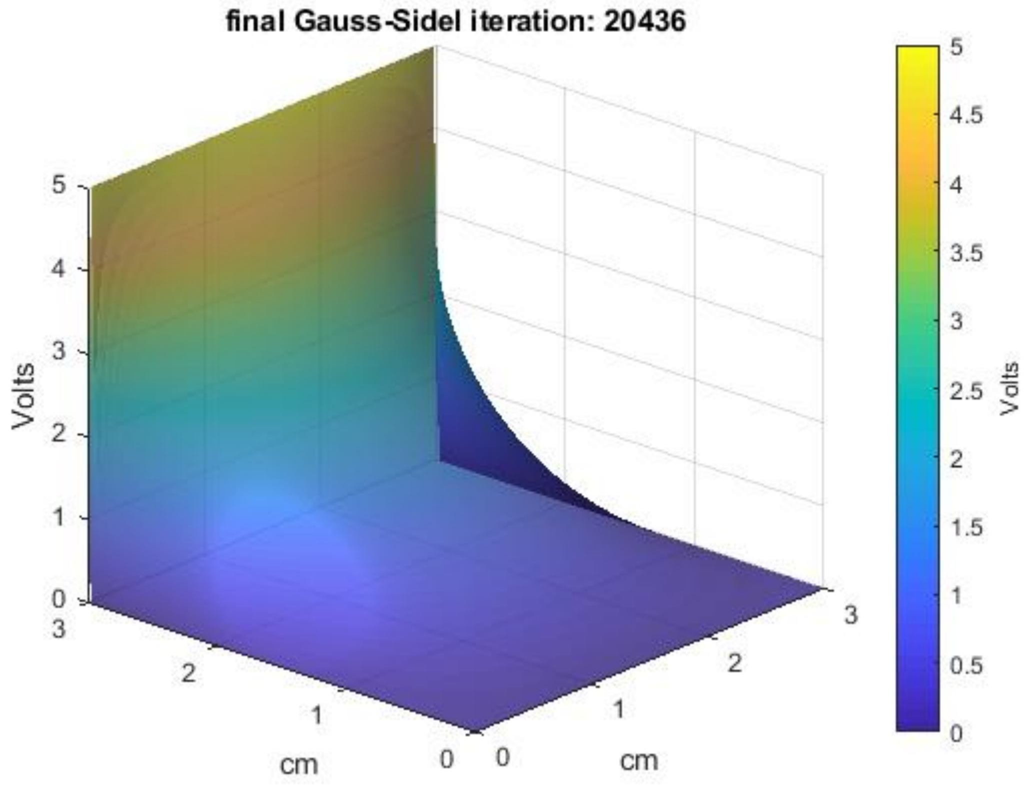

Gauss-Sidel iteration: 900
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Gauss-Sidel iteration: 10000



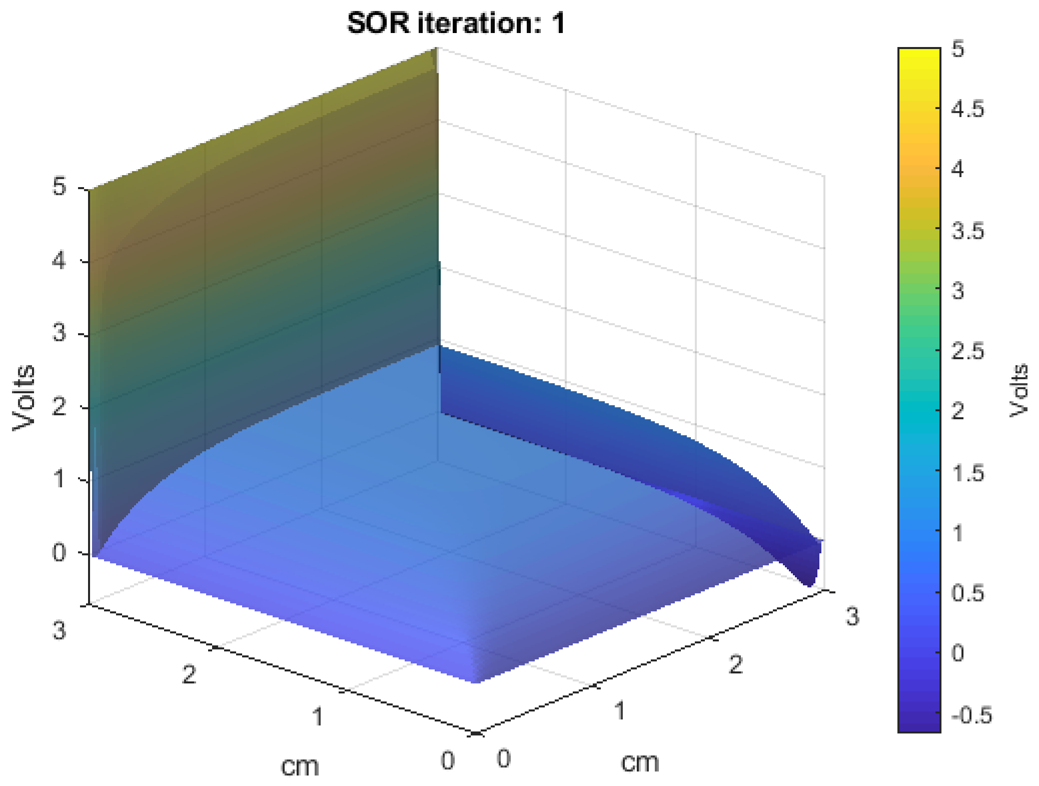
Gauss-Sidel iteration: 20000

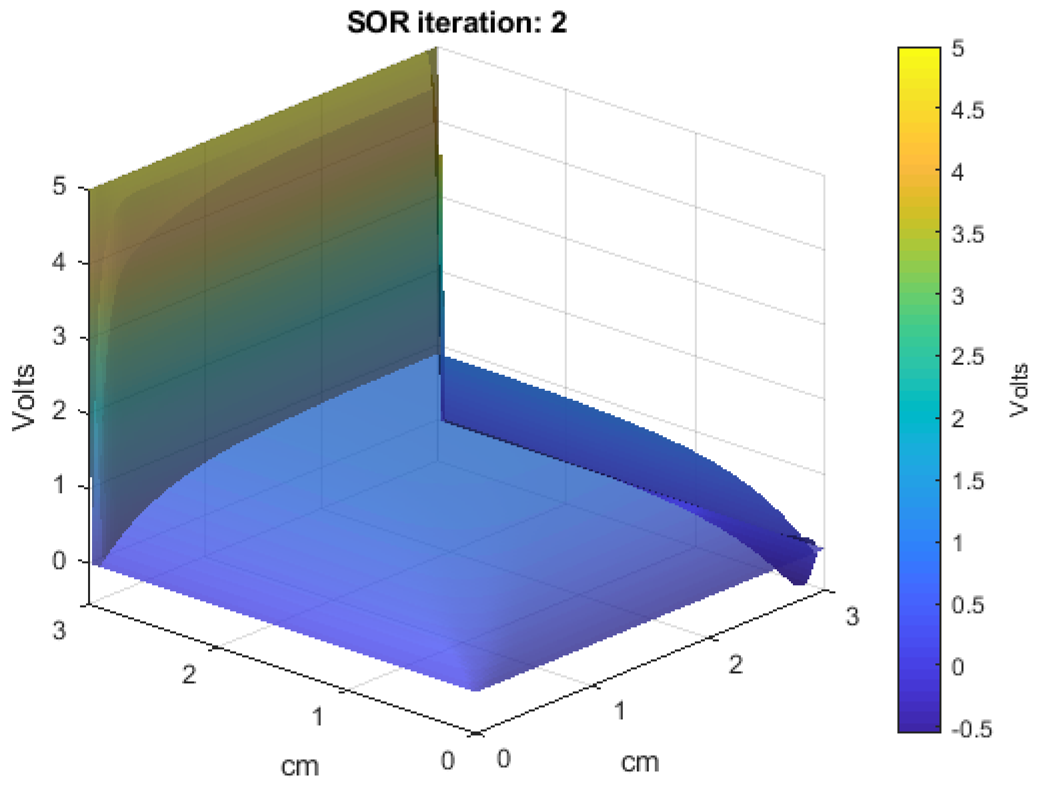




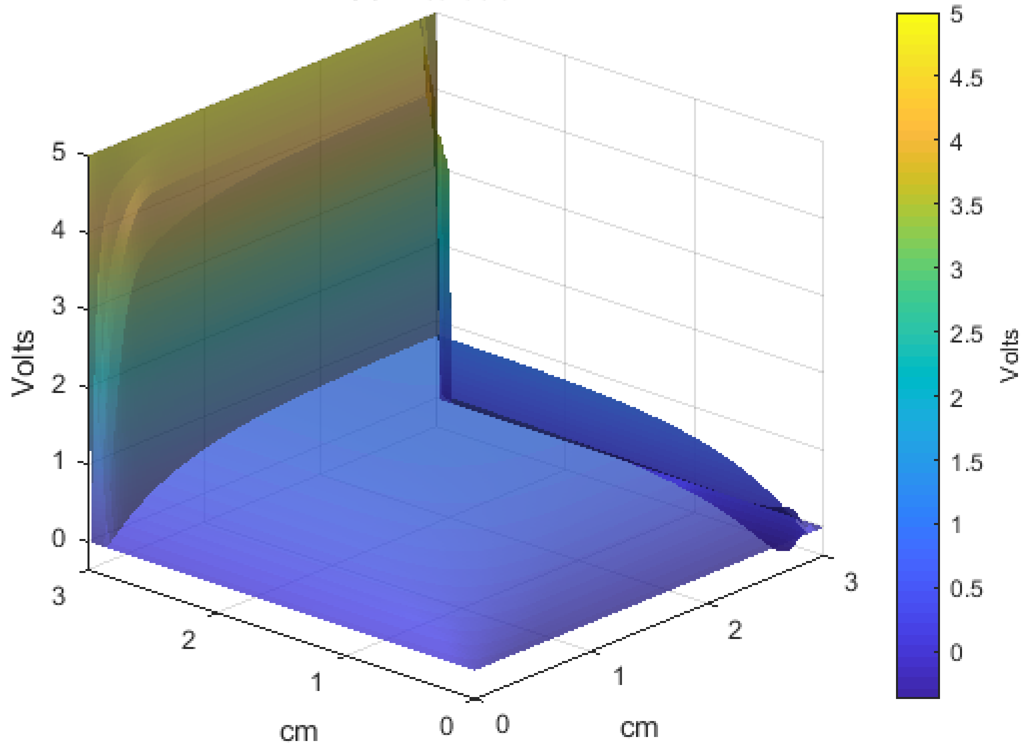
final Gauss-Sidel iteration: 20436
Volts
0
1
2
3
4
5
cm
cm
3
2
1
0
0
1
2
3
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

Next is SOR (mesh dimensions are 100x100)

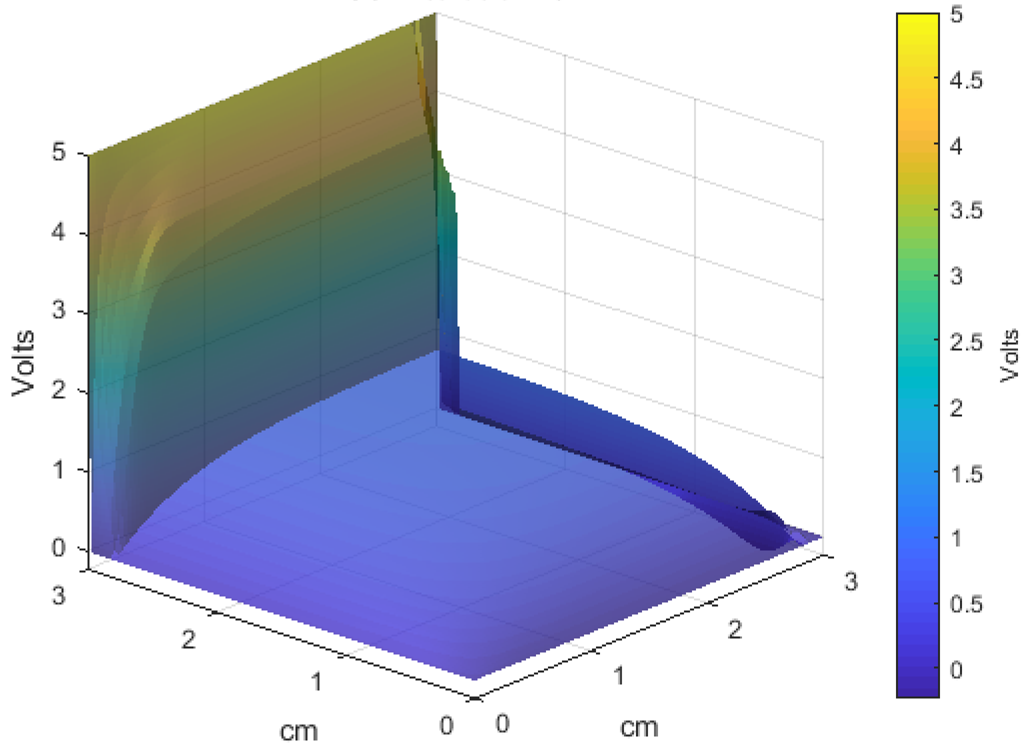




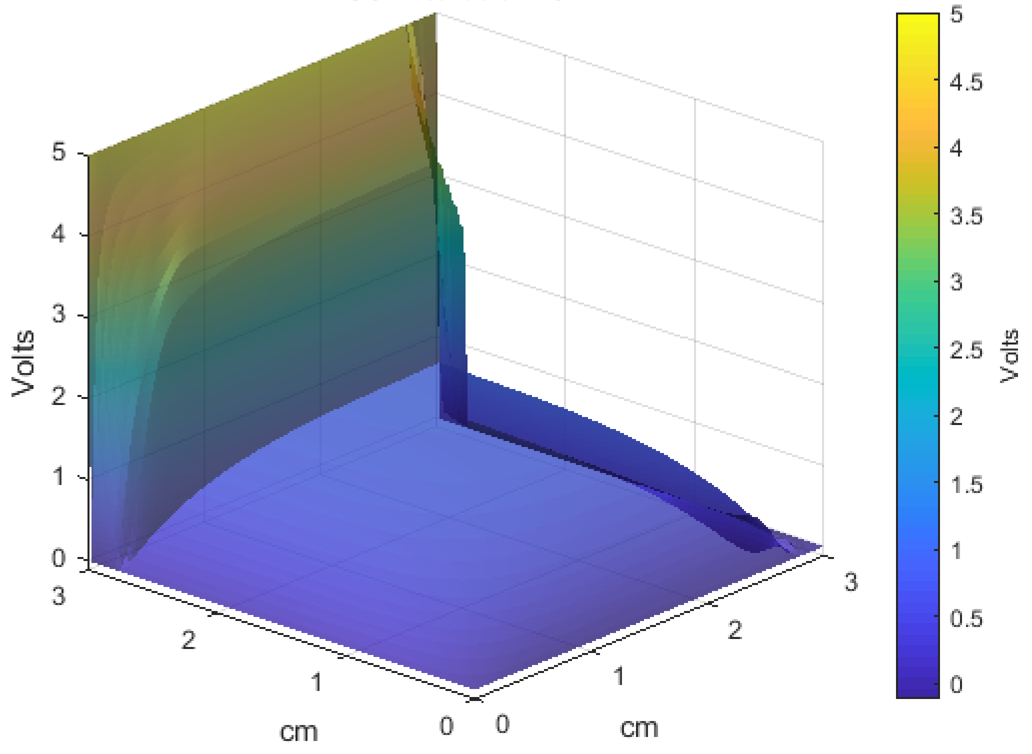
SOR iteration: 4



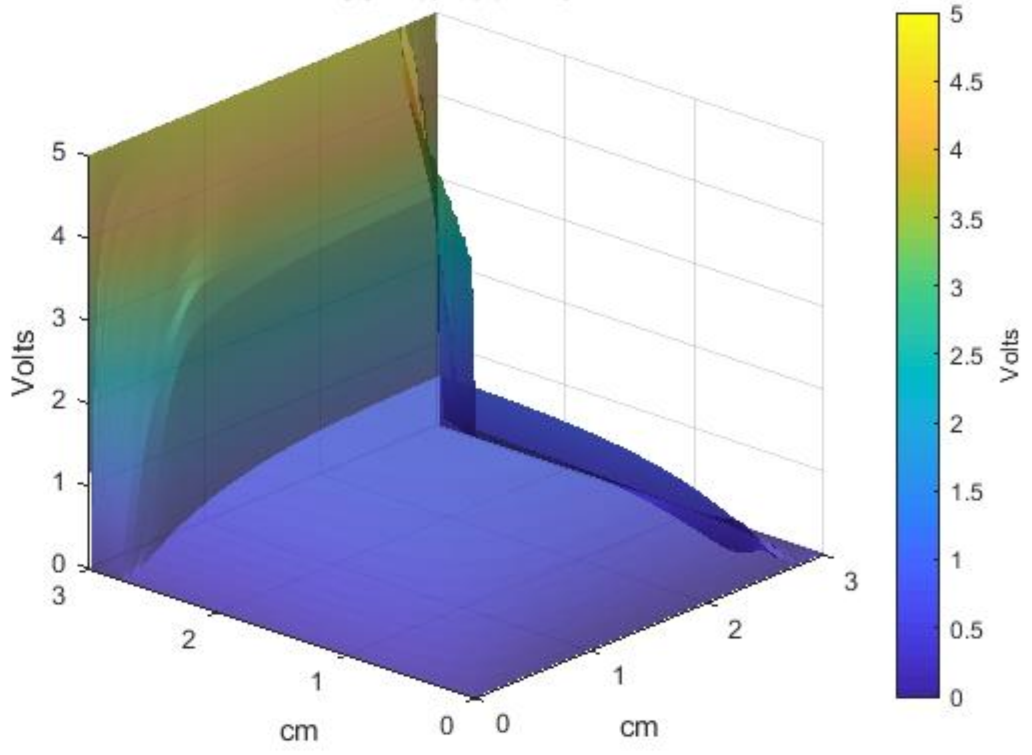
SOR iteration: 6

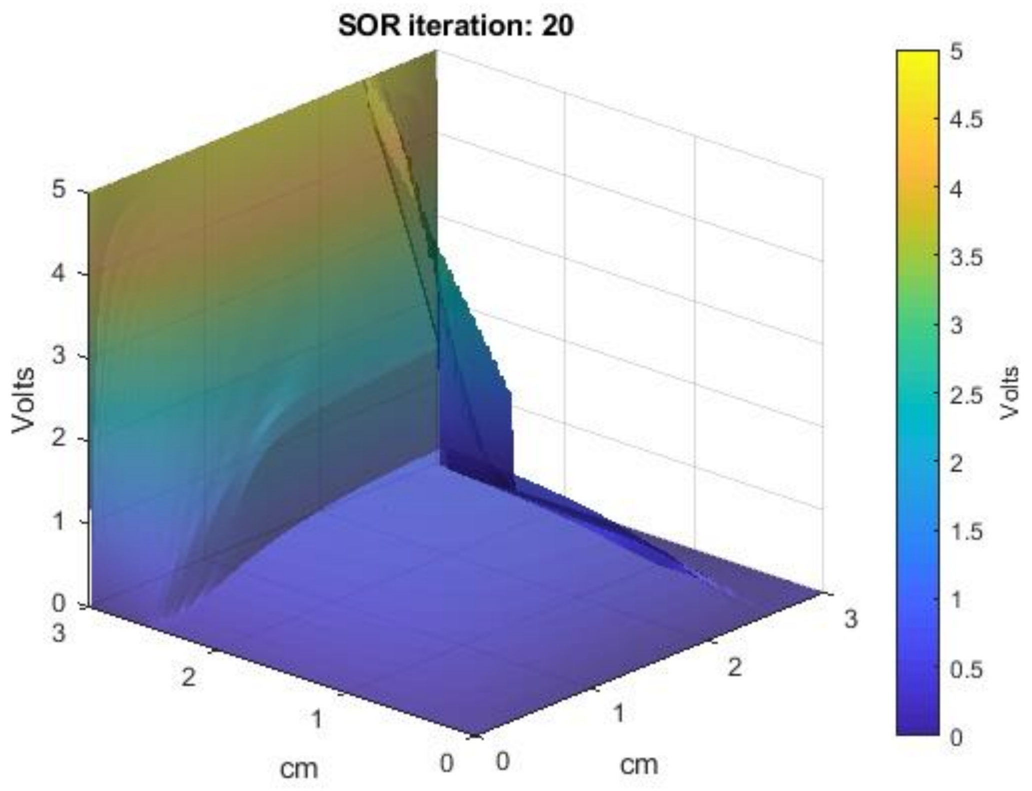


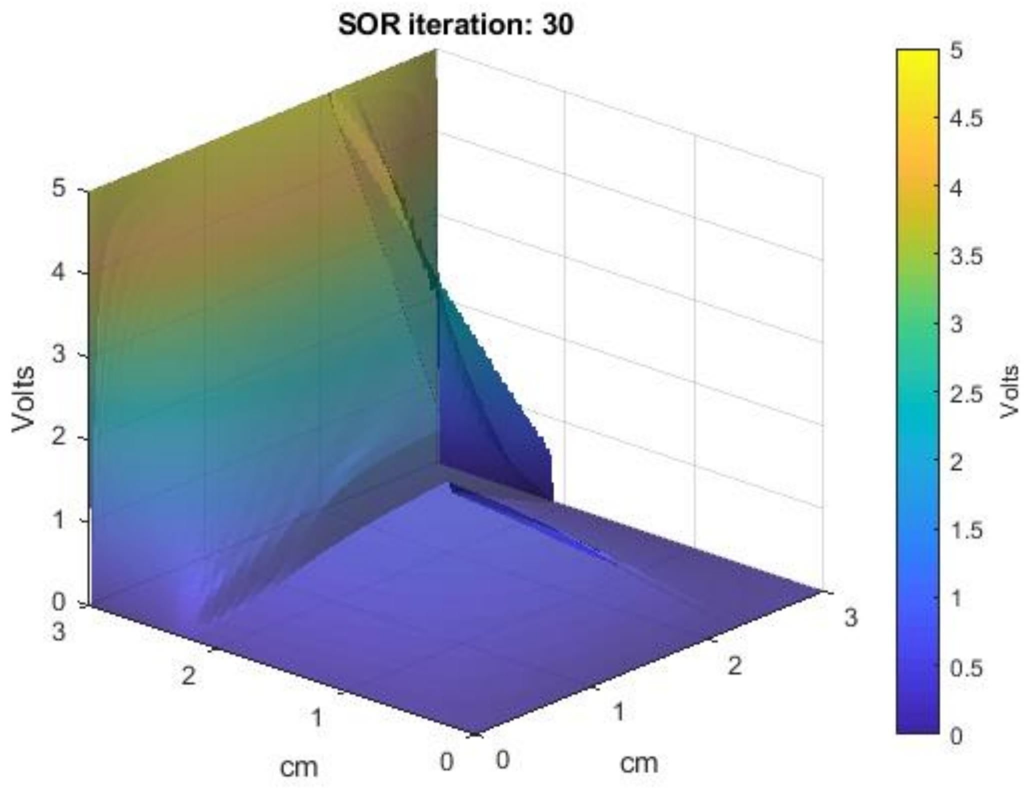
SOR iteration: 8



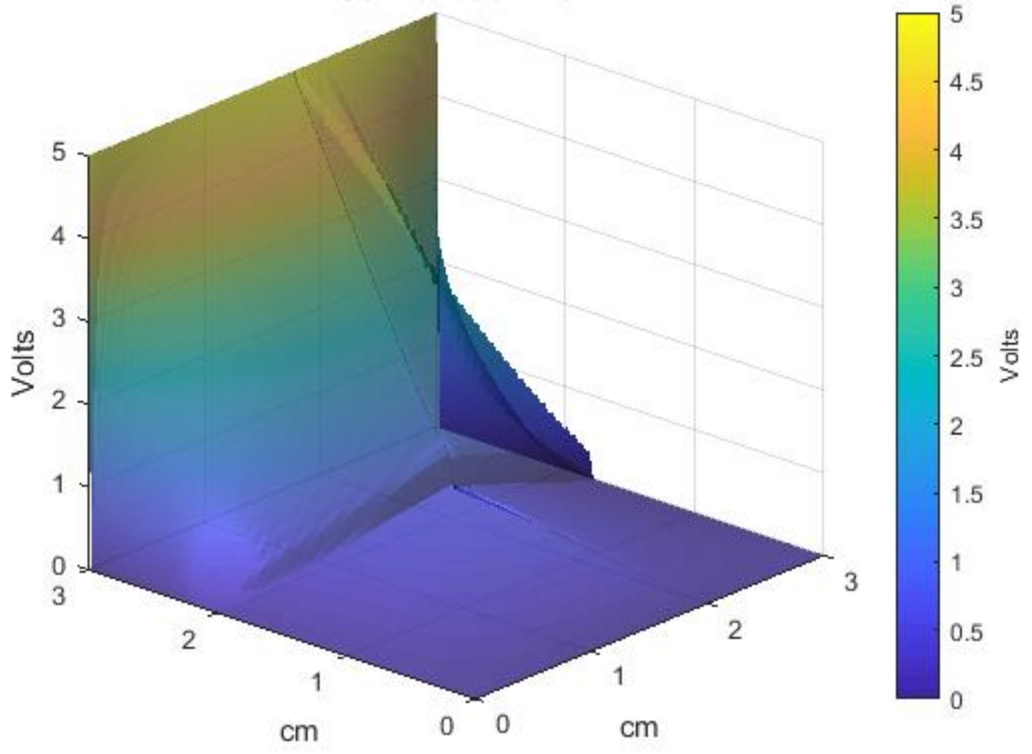
SOR iteration: 10



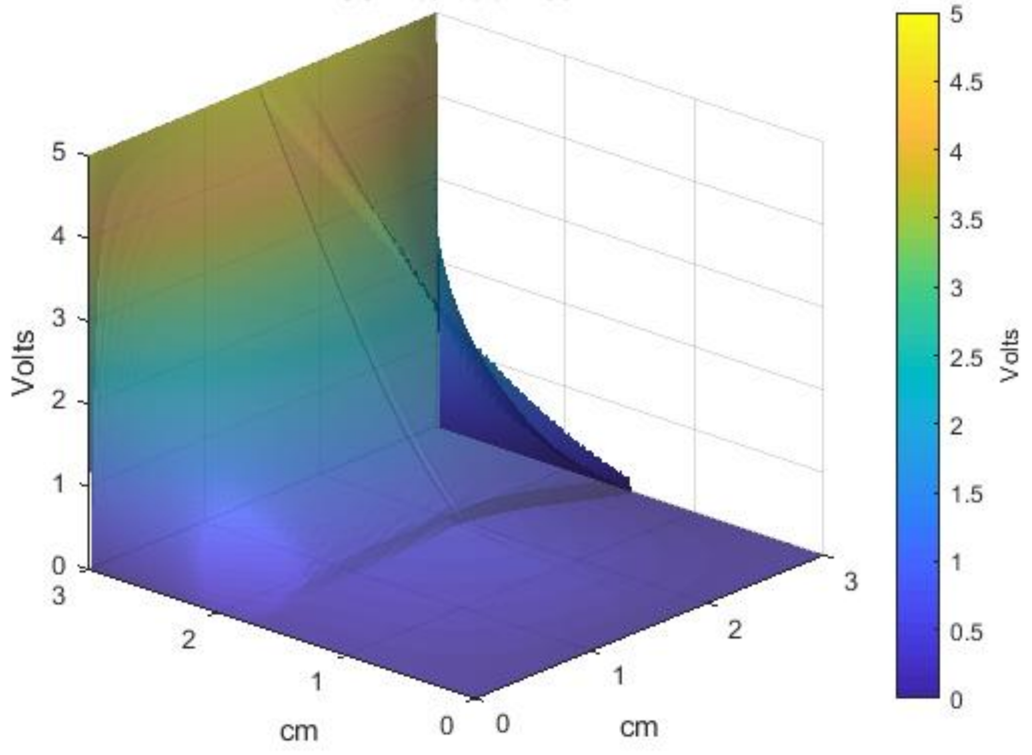

SOR iteration: 20
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

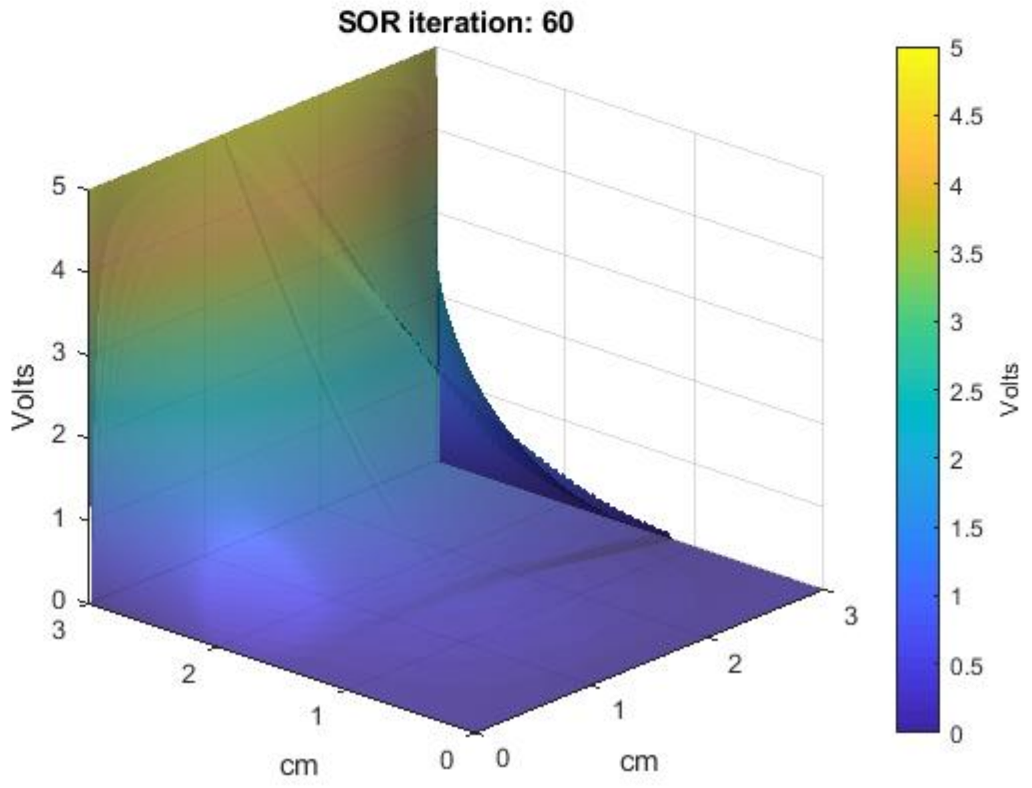

SOR iteration: 30
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

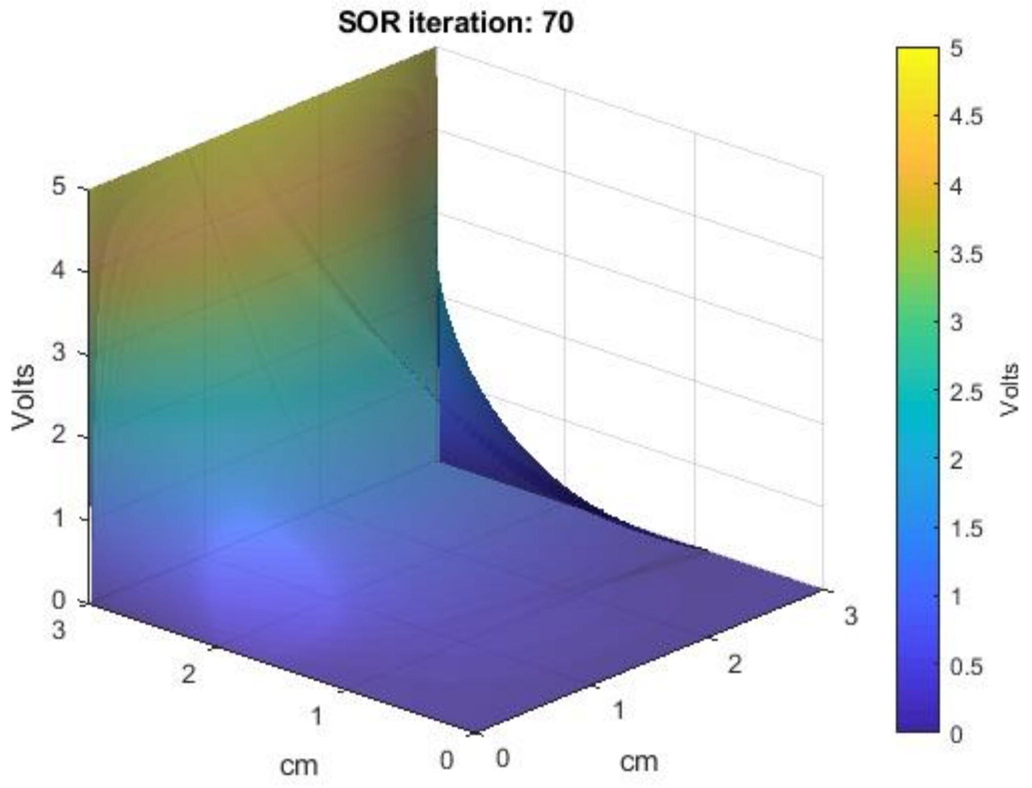
SOR iteration: 40

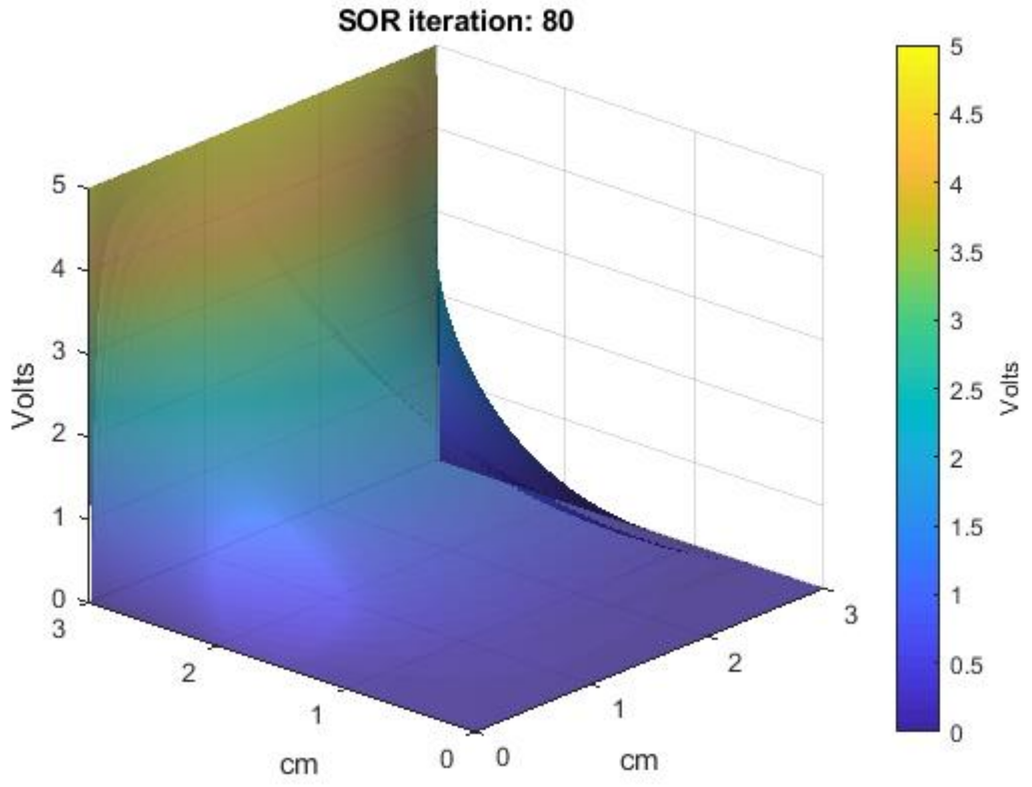


SOR iteration: 50

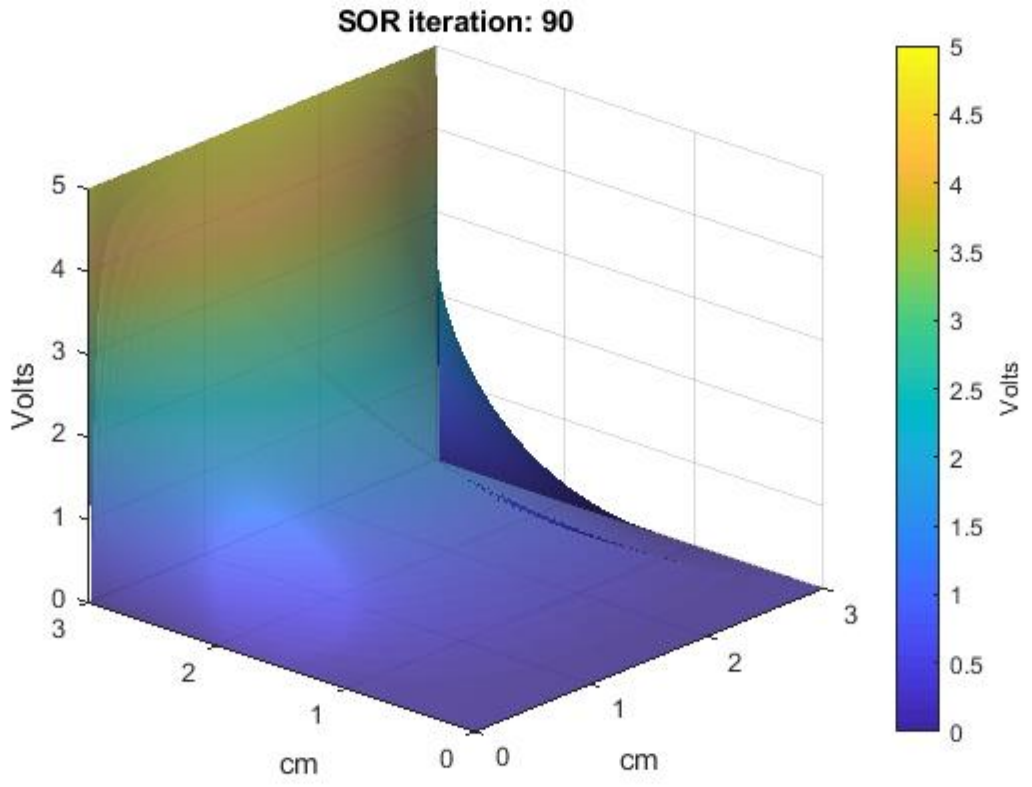



SOR iteration: 60
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

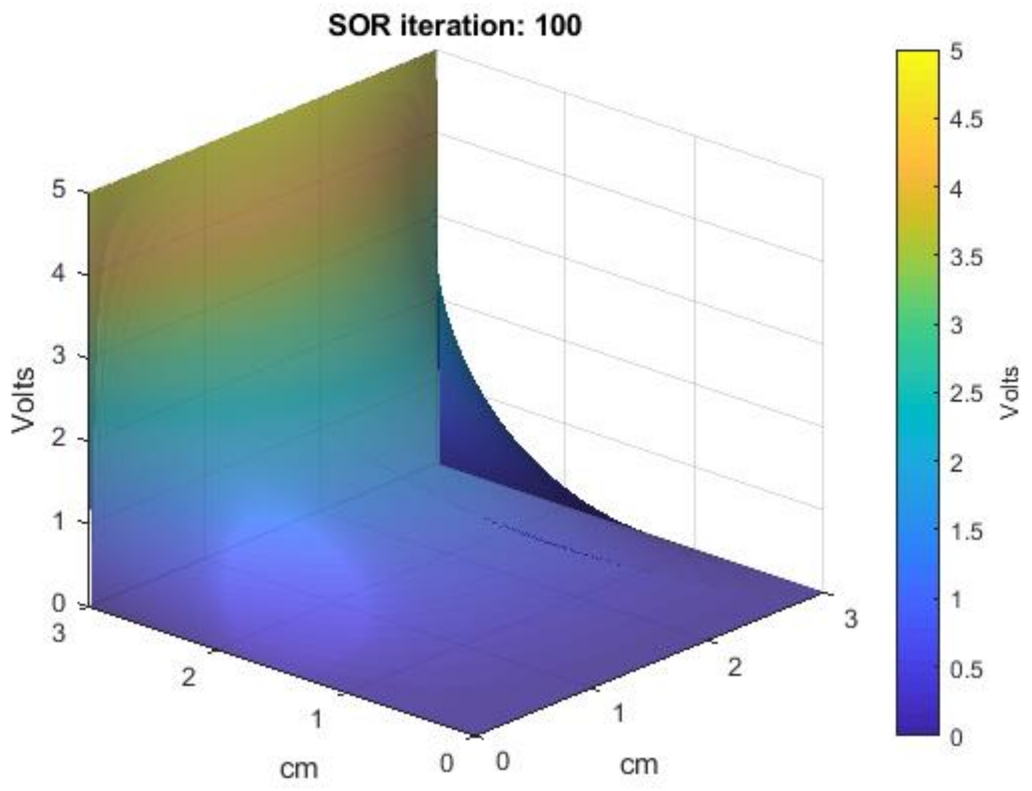

SOR iteration: 70
Volts
cm
cm
Volts
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

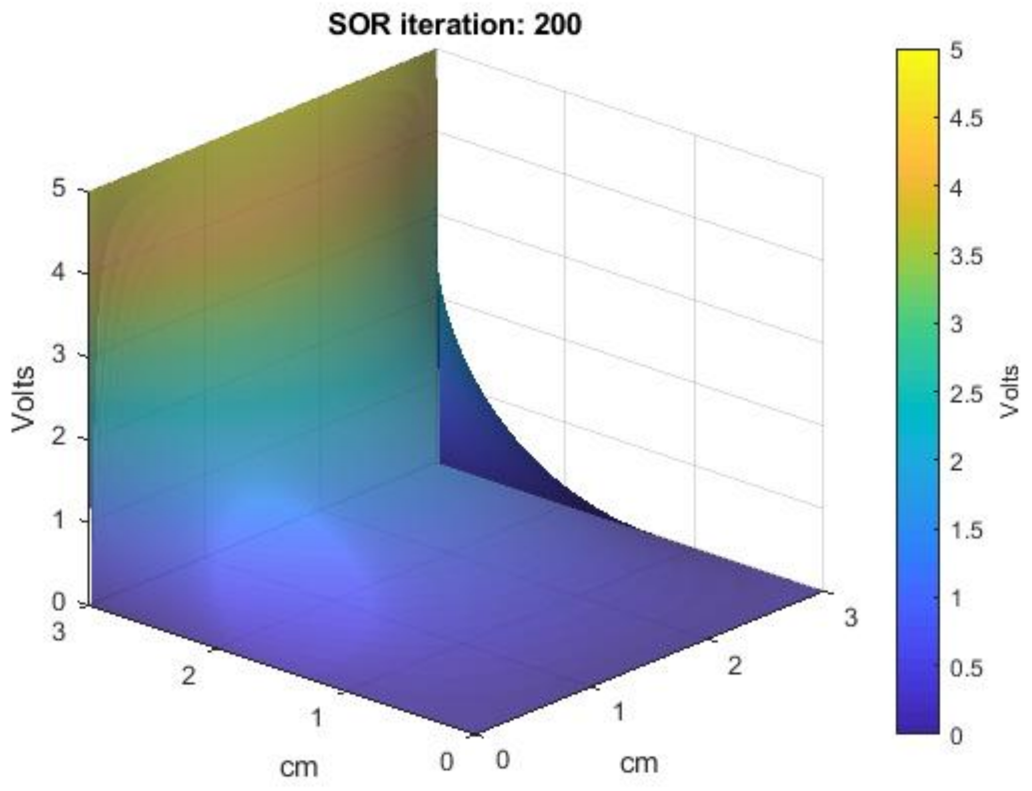


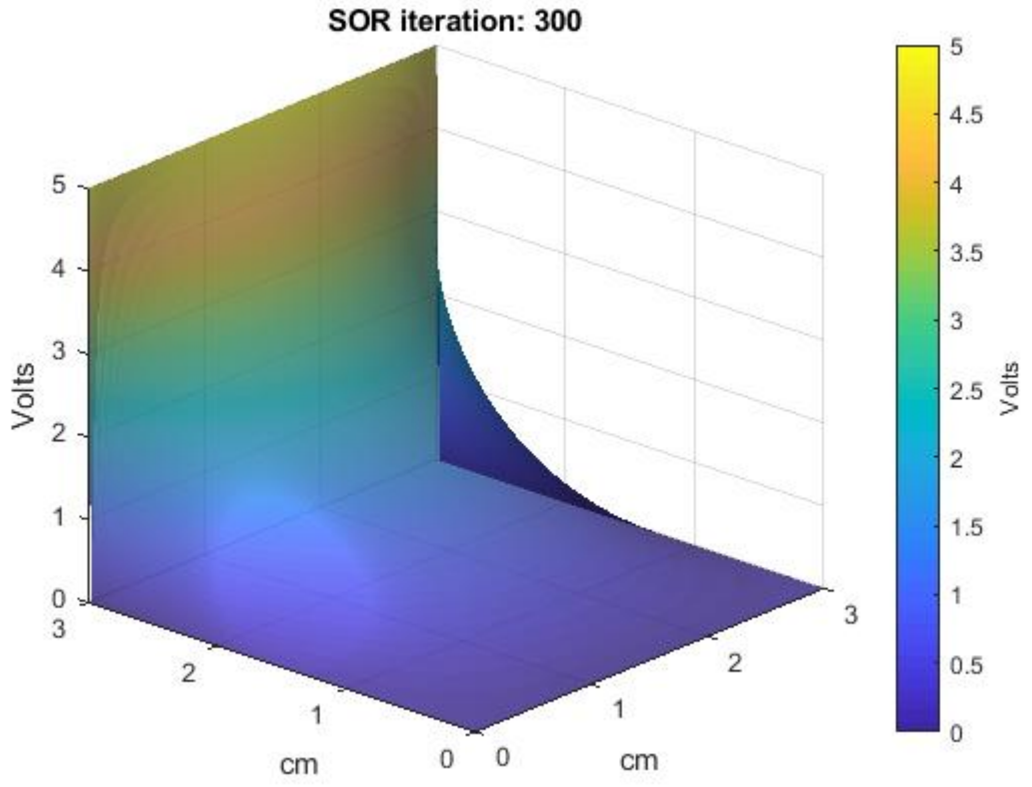
SOR iteration: 80
Volts
cm
cm
Volts

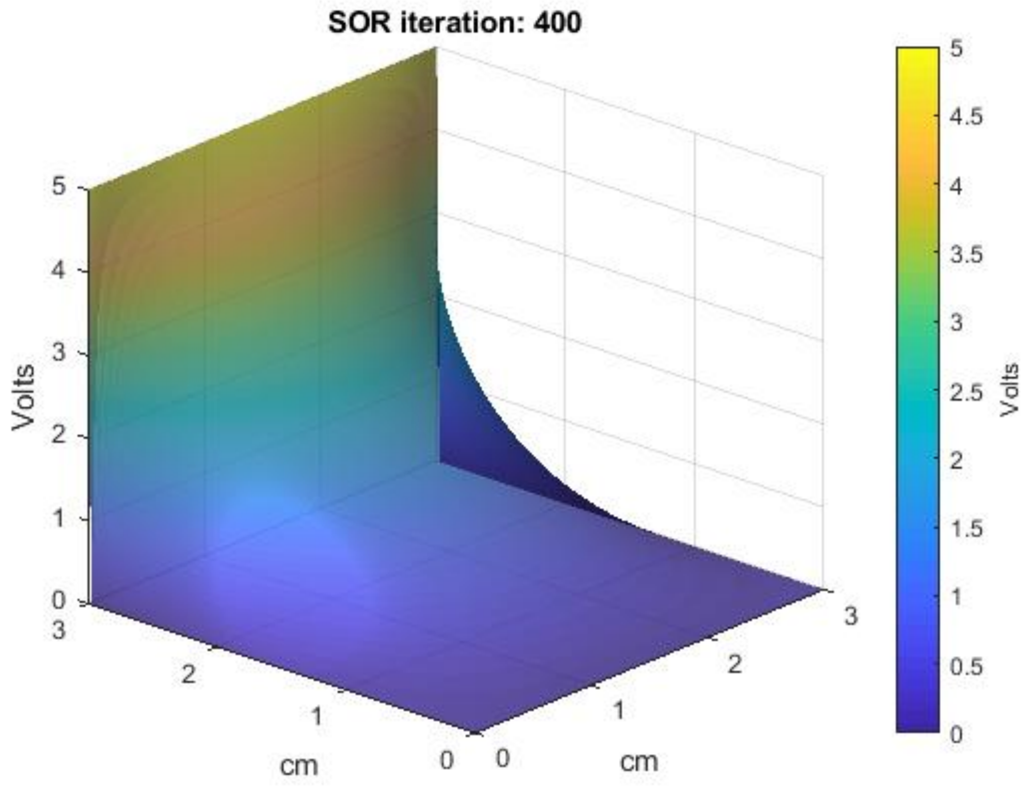


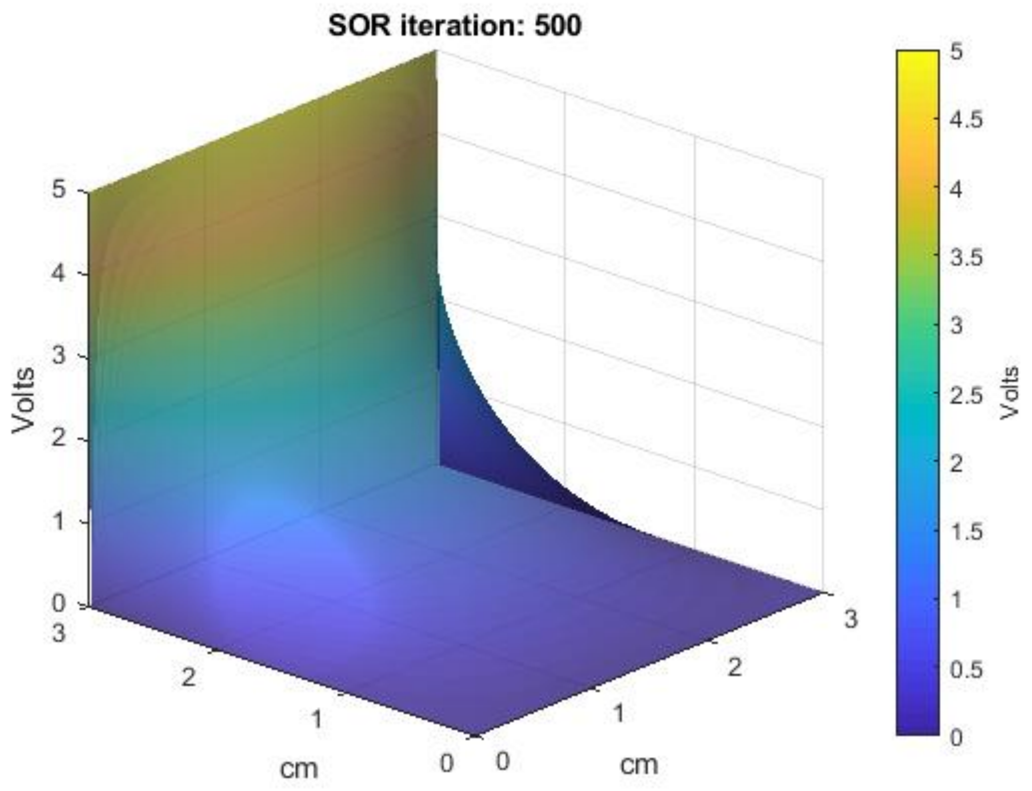
SOR iteration: 90
Volts
cm
cm
Volts


SOR iteration: 100
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

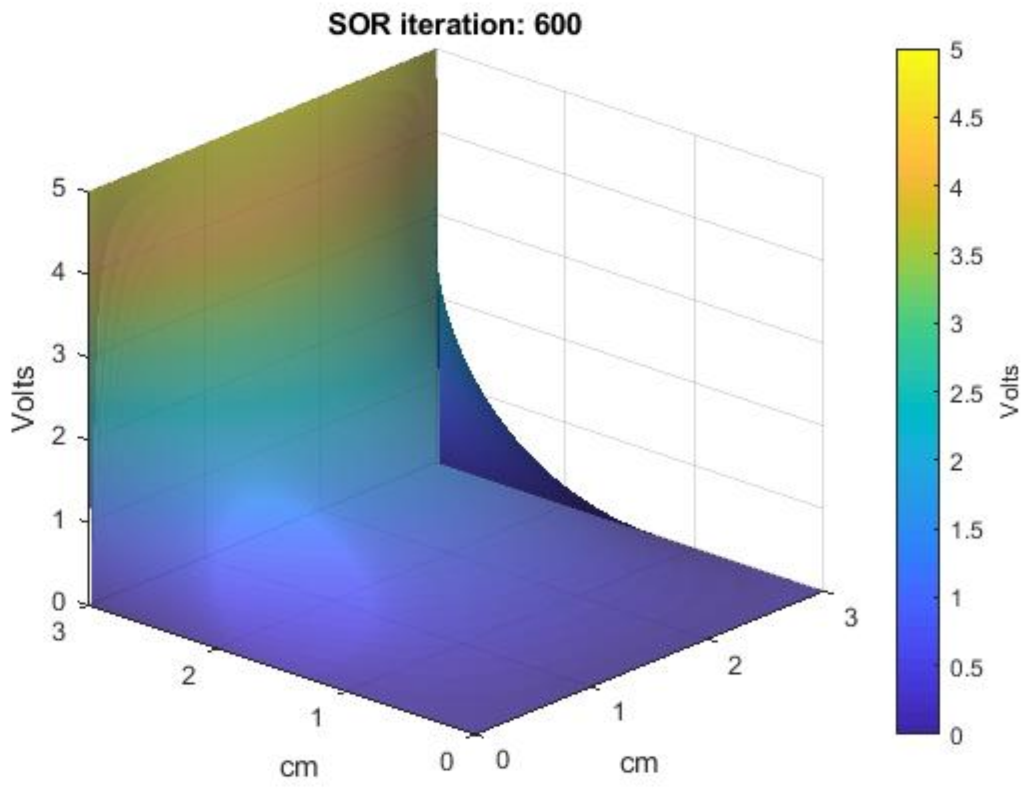

SOR iteration: 200
Volts
cm
cm
Volts
0
1
2
3
4
5
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


SOR iteration: 300
Volts
cm
cm
Volts
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

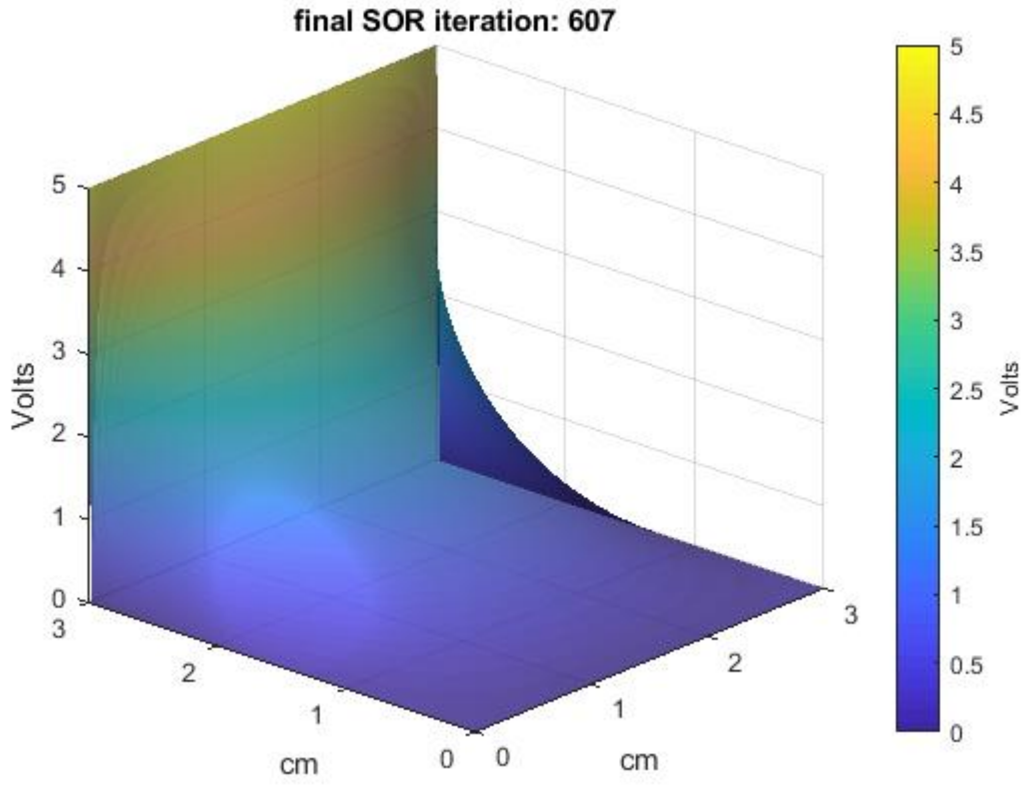

SOR iteration: 400
Volts
cm
cm
Volts
0
1
2
3
4
5
0
1
2
3
0
1
2
3
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



SOR iteration: 500
Volts
cm
cm
Volts



SOR iteration: 600
Volts
cm
cm
Volts



We see the SOR converges in far less iterations than the Gauss-Seidel and Jacobi schemes. The SOR does makes some unusual progressions getting to the final answer, it doesn't simply "smooth out" like the other two schemes. I'd guess that this is probably because the SOR is "overcorrecting" between guesses (textbook p.252). I have calculated some points by hand to ensure the scheme is put in correctly and sometimes I do expect negative values (seen in iterations such as 2-10). But at the end of the day, it's the final result that matters and the SOR does deliver the same results at a much faster convergence rate than the other two schemes.

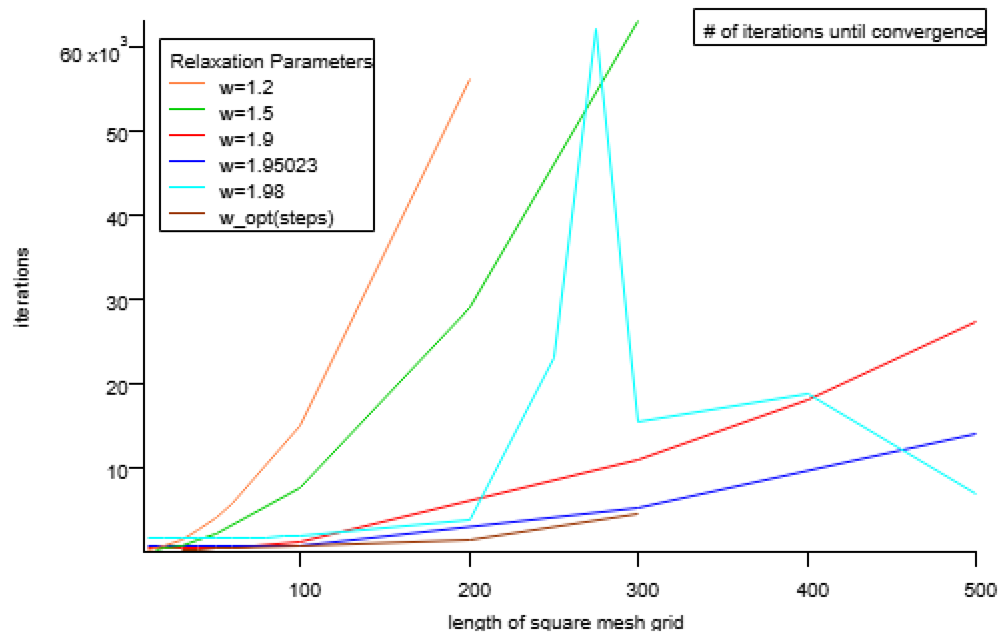
Section 6: which is the optimal S.O.R. relaxation parameter for this Problem?

The text gives a formula for the optimum value for the relaxation parameter, given the mesh is a 2-d square (eq. 8.27, p 254).

$$\omega_{opt} = \frac{2}{1 + sin(\frac{\pi}{N})}$$

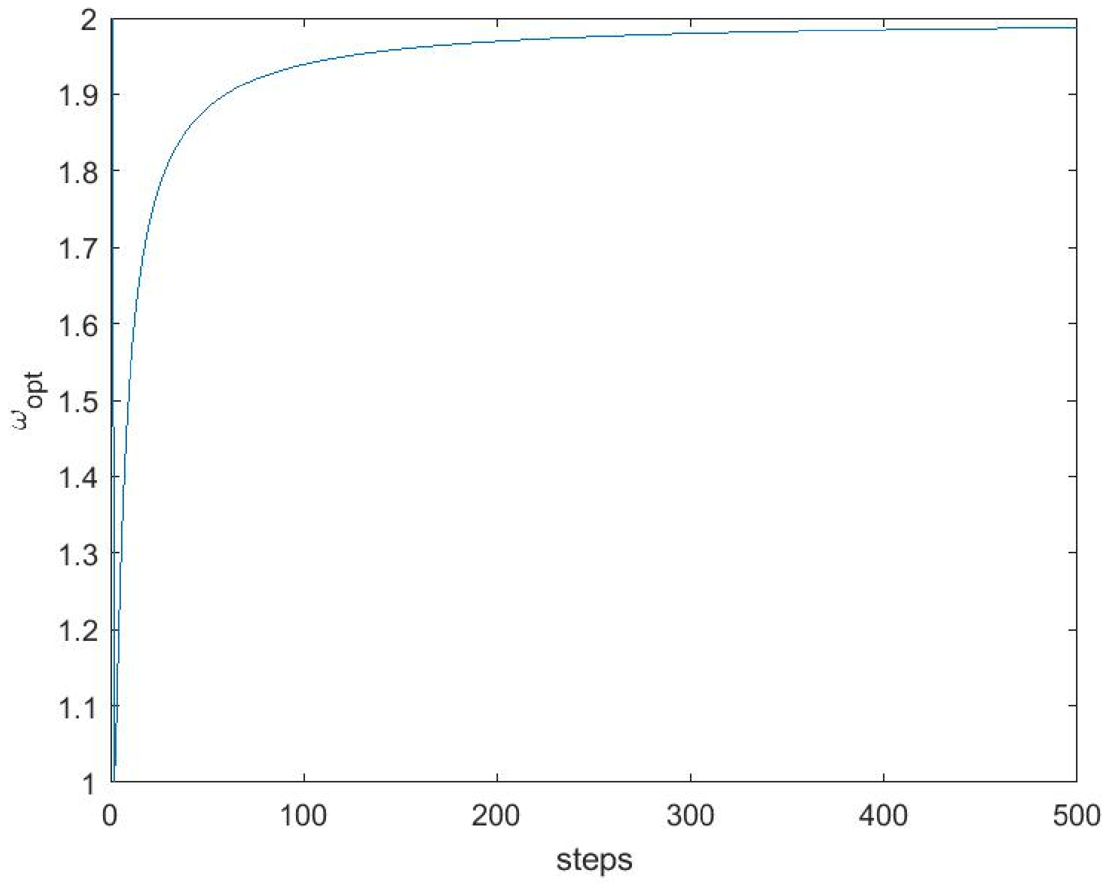
Eq 8.27 optimum value for SOR parameter, if the mesh is a square grid of length N points

I also tested the formula by plotting how fast the program converged for different values of ω, and various mesh sizes. The results are shown in the following graph



The wildest plot shown is for when ω equals 1.98. the spike in the 1.98 curve around 275 points is unbelievable, but I have double checked re-running the program and recording the number of iterations until convergence is achieved (fractional change must be less than 10^{-12} for all cases), and the same iteration results occur repeatedly and consistently.

Another interesting curve is the brown line where the parameter is a function dependent on the mesh size (given by eq. 8.27 seen above). This function indeed gives the best convergence rate but only up to mesh length of 300. Values over 300 are not plotted because the program does no converge when using eq. 8.27. I've plotted eq. 8.27 as a function of mesh length to show how the optimal parameter quickly approaches 2 for higher mesh sizes as seen below



This graph might shed light why the optimum parameter (according to text eq. 8.27) does not converge for steps greater than 300. My theory is that for these cases, ω is very close to 2 in which the program loses stability. If one does want a plot with more than 300x300 mesh points, my experience has found they can stick with ω=1.9052 and the scheme will produce a result with a reasonable convergence rate. If they want more than 450x450 points, they might consider trying ω=1.98. One thing is clear, the optimum value of the relaxation parameter changes for different mesh sizes.