5/5/2017

Numerical Implementation of 2D Helmholtz Equation

Dakota Dalton - 1366027

MECE 5397 – Scientific Computing

ABSTRACT

This project entails the representation of the Helmholtz Equation in two dimensions using numerical methods in MATLAB. The Helmholtz equation is a time independent form of the wave equation similar to the Laplace and Poisson equations. The boundary conditions, domain limits, and constant parameters are prescribed in the assignment. The Gauss-Seidel/Liebmann iterative method with and without relaxation is used to approximate the equation. A mathematical description of the problem is given, and then a discretized version of the equation is developed. A pseudocode of the algorithm is presented and hardware specifications are given. Different grid sizes and relaxation weights are used to see how the behavior of the numerical method varies. The simpler Laplace and Poisson equations are tested as well, and the boundary conditions are verified. The value of lambda is also changed to test the solver.

Contents

MATHEMATICAL FORMULATION

The Helmholtz equation is an elliptic partial differential equation of the second order with respect to the spatial variables. It is similar to the Poisson equation but includes an additional output term that is not differentiated. The equation is

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \Lambda u = F(x,y) \tag{1}$$

The parameter Λ is given as 1. The domain is a rectangle, given as

$$\mathrm{a}_x < x < b_x, \quad a_y < y < b_y \tag{2}$$

Where

$$a_x = a_y = -\pi, \; b_x = b_y = \pi \tag{3}$$

The boundary conditions for y are given as

$$u(x, b_y) = f_b(x), \qquad u(x, a_y) = g_b(x) \tag{4}$$

$$f_b(x) = x(b_x - x)^2, \quad g_b(x) = (b_x - x)^2 \cos\left(\frac{\pi x}{b_x}\right) \tag{5}$$

The boundary conditions for x are given as

$$u(a_x, y) = g_b(a_x) + \frac{y - a_x}{b_y - a_y}[f_b(a_x) - g_b(a_x)], \qquad \left.\frac{\partial u}{\partial x}\right|_{x=b_x} = 0 \tag{6}$$

The forcing function F is given as

$$F(x,y) = \sin\left[\pi \frac{x - a_x}{b_x - a_x}\right] \cos\left[\frac{\pi}{2}\left(2\frac{y - a_y}{b_y - a_y} + 1\right)\right] \tag{7}$$

DISCRETIZATION

For the discretization of this PDE, the centered difference formula is used, where

$$\frac{\partial^2 u}{\partial x^2} \simeq \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\Delta x^2}, \; \frac{\partial^2 u}{\partial y^2} \simeq \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\Delta y^2} \tag{8}$$

The spatial terms are set such that $\Delta x^2 = \Delta y^2 = \Delta^2$ where Δ^2 is the area of the squares used to discretize the domain. Substituting the approximate terms into the original equation yields

$$u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + (\Delta^2 \Lambda - 4) u_{i,j} = \Delta^2 F_{i,j} \tag{9}$$

Solving for $u_{i,j}$,

$$u_{i,j} = \frac{u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - \Delta^2 F_{i,j}}{4 - \Delta^2 \Lambda} \tag{10}$$

This is the equation used to implement the Gauss-Seidel method. The insulated/Neumann boundary condition on the right side of the rectangle is accounted for through the use of a ghost node. The ghost node uses the centered difference formula as such:

$$\frac{\partial u}{\partial x} \simeq \frac{u_1 - u_{-1}}{2\Delta} = 0 \tag{11}$$

$$\Rightarrow u_{i+1,j} = u_{i-1,j} \tag{12}$$

Exploiting this fact for the right side of the domain results in

$$u_{b_x,j} = \frac{2u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - \Delta^2 F_{i,j}}{4 - \Delta^2 \Lambda} \tag{13}$$

allowing for the use of the discretized formula at the edge of the domain.

Additionally, the technique known as Successive Overrelaxation (SOR) is employed. Overrelaxation is the process where the previous and current iterations are weighted to speed up convergence. Mathematically, this is represented as

$$u_{i,j}^{new} = \lambda u_{i,j}^{new} + (1 - \lambda) u_{i,j}^{old} \tag{14}$$

where a λ of 1 would be equivalent to no relaxation, $0 < \lambda < 1$ would be underrelaxation, and $1 < \lambda < 2$ would be overrelaxation. The term $SOR\lambda$ will be used to prevent any confusion with the Λ term in the problem statement.

PSEUDOCODE

The following is the core loop for the implementation of the Gauss-Seidel method. It passes through a full column of the y coordinates before moving to the next x value in order to exploit the column-major ordering of Matlab. The loop termination condition is based upon the infinity norm of the matrix, continuing to iterate until the maximum difference between two iterations is less than 1%.

```
Helmholtz
while error > 0.01

      uprev = u

      for i = 2:length(x)- 1
            for j = 2:length(y) - 1

                  u(i,j) = [u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1)
                        -  Δ^2 * F(i,j)] / (4 - Δ^2 * Λ)
            end

            u(i,end) = [2 * u(i,end-1) + u(i+1,end) + u(i-1,end)
```

```
                        - Δ^2 * F(i,end)] / (4 - Δ^2 * Λ)
      end

      error = ||(u-uprev)/u||_∞
end
```

The pseudocode for the SOR method is very similar and includes only a couple of additional lines to implement it.

```
SORΛ = 1.2

while error > 0.01

      uprev = u

      for i = 2:length(x)- 1
            for j = 2:length(y) - 1

                  u(i,j) = [u(i+1,j) + u(i-1,j) + u(i,j+1) + u(i,j-1)
                           -  Δ^2 * F(i,j)] / (4 - Δ^2 * Λ)
                  u(i,j) = SORΛ * u(i,j) + (1-SORΛ) * uprev(i,j)
            end

            u(i,end) = [2 * u(i,end-1) + u(i+1,end) + u(i-1,end)
                        - Δ^2 * F(i,end)] / (4 - Δ^2 * Λ)
            u(i,end) = SORΛ * u(i,j) + (1-SORΛ) * uprev(i,j)

      end

      error = ||(u-uprev)/u||_∞
end
```

MACHINE TECHNICAL SPECIFICATIONS

OS: Windows 10 Home 64-bit

CPU:

- Intel Celeron N2830 2.16 GHz (Dual-core)
- L1 Cache: 2 x 24 KB
- L2 Cache 1024 KB

Memory: 4 GB DDR3 800 MHz

GPU: Intel HD Graphics 2GB VRAM

Disk: WD Blue 500GB 5400 RPM

RESULTS

The first step taken in the approach to this problem was to verify the implementation of the boundary conditions and the forcing function. The three Dirichlet boundary conditions were fairly straightforward to implement, and are shown in Figure 1.

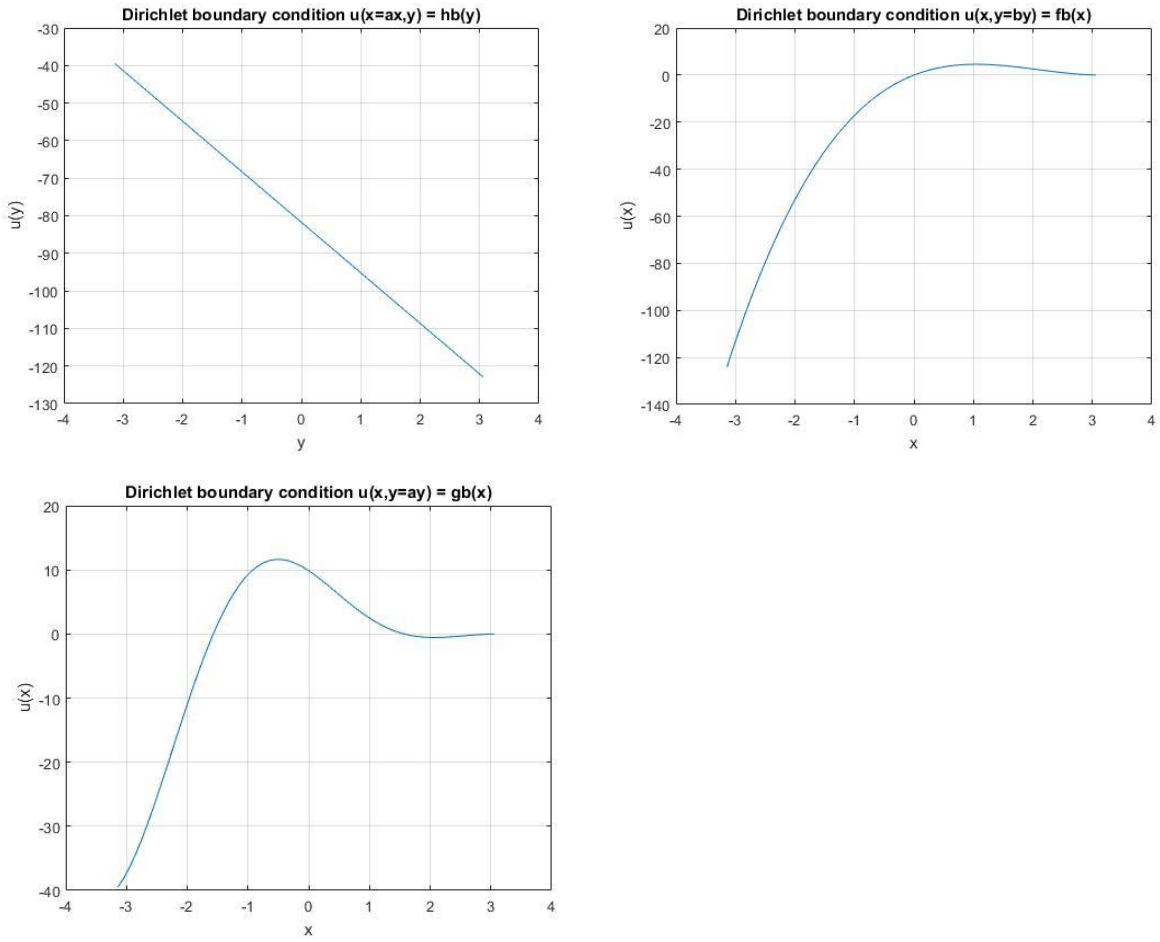


Figure 1. Dirichlet Boundary Conditions

As can be seen, these boundary conditions form three continuous sides of the domain, representing the left, bottom, and top respectively. The forcing function F(x,y) is shown in Figures 2 and 3. The forcing function is symmetric and peaks at -1 in the center of the domain.

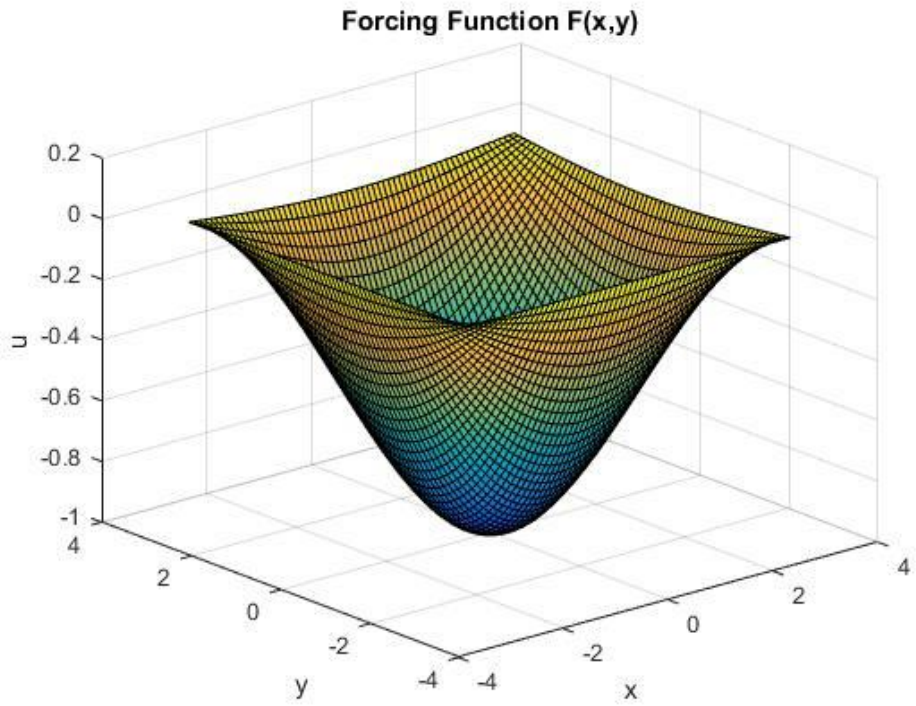


Figure 2. Surface of the forcing function F(x,y)

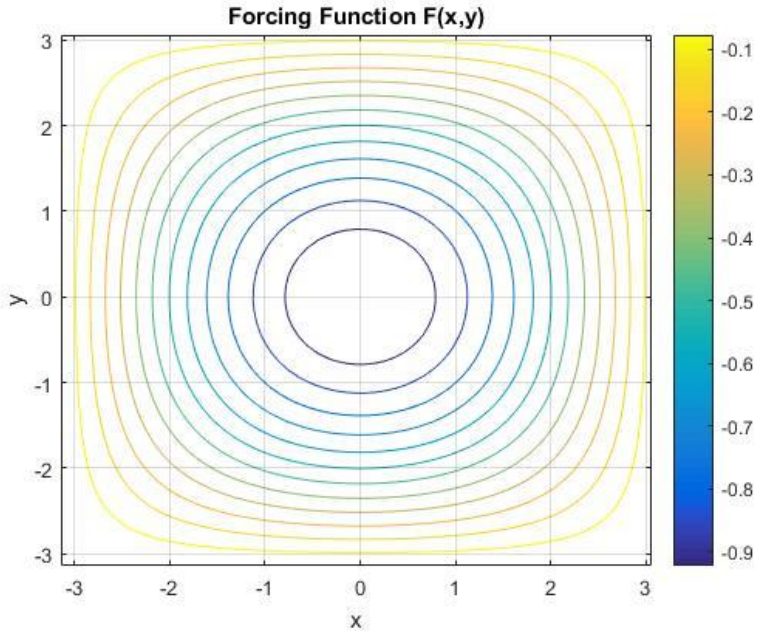


Figure 3. Contour of forcing function F(x,y)

The right side (x=bx) of the domain has a Neumann boundary condition, which showed some unusual behavior, shown in Figures 2 and 3.

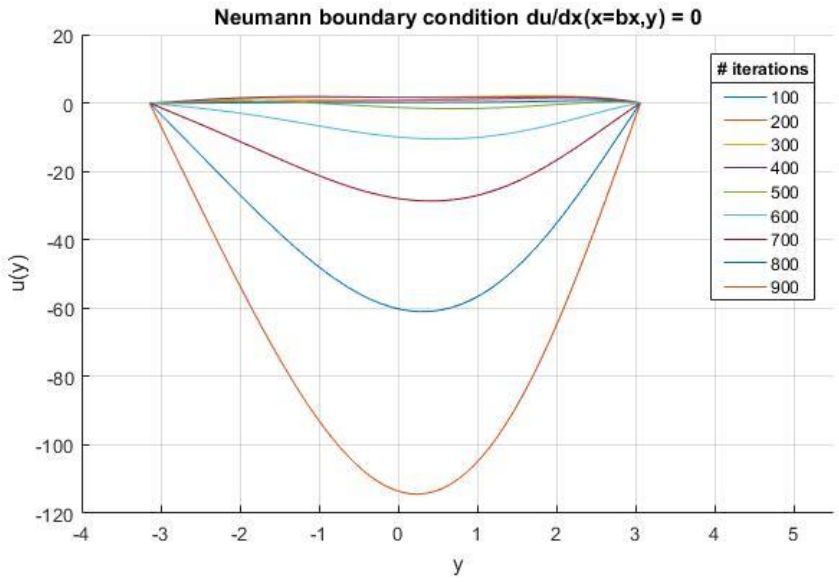


Figure 4. Neumann condition, delta = 0.1

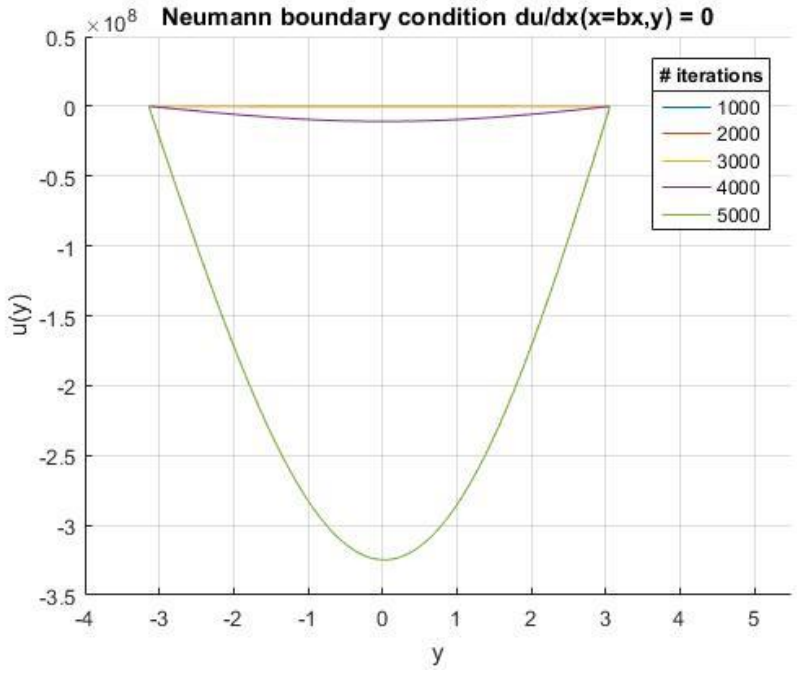


Figure 5. Neumann condition, delta = 0.1

The Neumann boundary condition was accounted for using the centered difference formula and ghost nodes, as shown in the Mathematical Formulation section. Because of this, the values for this boundary condition were part of the iterations of the Gauss-Seidel loop, and changed with each iteration. Figures 2 and 3 show the effect of increasing iterations on the boundary values for $\Delta = 0.1$. At each of these instances the flux remains zero, satisfying the condition, but otherwise continues to diverge to $-\infty$ (note the y-axis values on Figure 3). This behavior was considered unusual so simplifications of the Helmholtz equation,

namely the Laplace and Poisson equation, were tested using the same boundary conditions to compare the behavior.

The Laplace equation in 2 dimensions is represented by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0, \tag{15}$$

while the Poisson equation is represented by

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = F(x, y) \tag{16}$$

These equations occur when setting the forcing function and Λ in the Helmholtz equation to 0.

Figure 6 shows the surface and contour plots of the Laplace equation, and Table 1 shows numerical results for several different grid sizes. The convergence cutoff for these and other results in this report was determined by the difference between the infinity norms of two successive iterations being less than 1%.

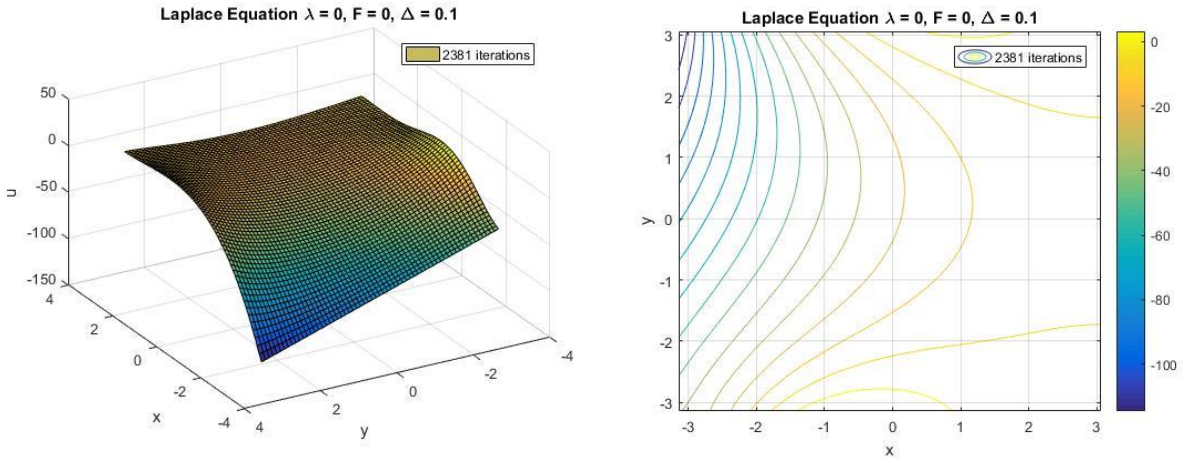


Figure 6. Surface and contour of Laplace function

Table 1. Results for Laplace equation

Laplace		Gauss-Seidel			SOR		
Δ	# nodes	max	mean	min	max	mean	min
0.25	632	11.5384	-25.3143	-123.579	11.5384	-24.9498	-123.579
0.1	3948	11.6244	-25.044	-122.906	11.6244	-24.9609	-122.906
0.025	63165	11.6369	-24.8688	-123.915	11.6369	-24.496	-123.915

The figures show a smooth and continuous response and maintain the given boundary conditions, and the numerical results show little change between differing grid sizes and relaxation methods. Appendix A

contains additional plots and figures for the different grid sizes and relaxations for both the Laplace and Poisson equations. It is apparent that the numerical solver works well for the Laplace equation and displays grid independence. Increasing in complexity is the Poisson equation, results of the numerical implementation of which are shown in Figure 7 and Table 2. The results are nearly the same as with the Laplace equation, again showing little variation with changes in grid size and the implementation of relaxation.

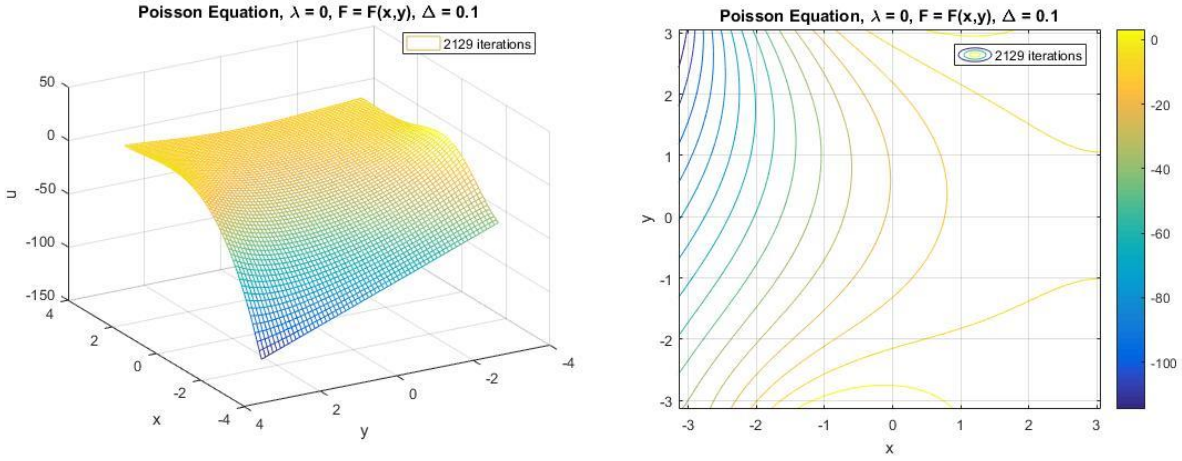


Figure 7. Surface and contour plots of the Poisson equation

Table 2. Results for Poisson equation

Poisson		Gauss-Seidel			SOR		
Δ	# nodes	max	mean	min	max	mean	min
0.25	632	11.5384	-24.2268	-123.578	11.5384	-24.9498	-123.579
0.1	3948	11.6244	-23.8074	-122.906	11.6244	-24.9609	-122.906
0.025	63165	11.6369	-23.6992	-123.915	11.6369	-24.496	-123.915

These simpler test cases were implemented to verify that the Gauss-Seidel solver could produce coherent results with and without relaxation before tackling the full Helmholtz equation.

Figures 8-13 show meshes and contours of the Helmholtz equation with and without relaxation for different grid sizes. Table 3 summarizes these results for the Helmholtz equation. The behavior of the Neumann boundary conditions seen in Figures 4 and 5 is demonstrated more clearly in the following plots as finer grid sizes require more iterations. The grid with elements of side length 0.1 requires 964 iterations to converge while the grid with elements of side length 0.0125 requires 84503 iterations to converge, The magnitude of the peak of the latter is over twice as large as the former due to the greater number of iterations. The results with the SOR method show that it is the iteration count and not the grid size that drives the

Neumann condition downward, as it is less negative than the same grid size without relaxation and differs only in the number of iterations used.

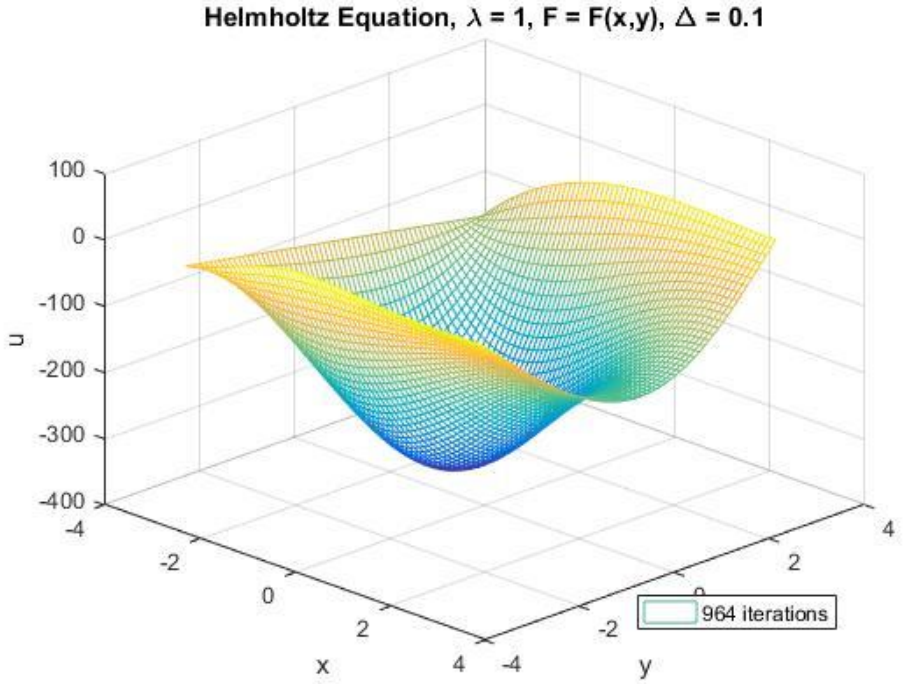


Figure 8. Mesh of Helmholtz equation

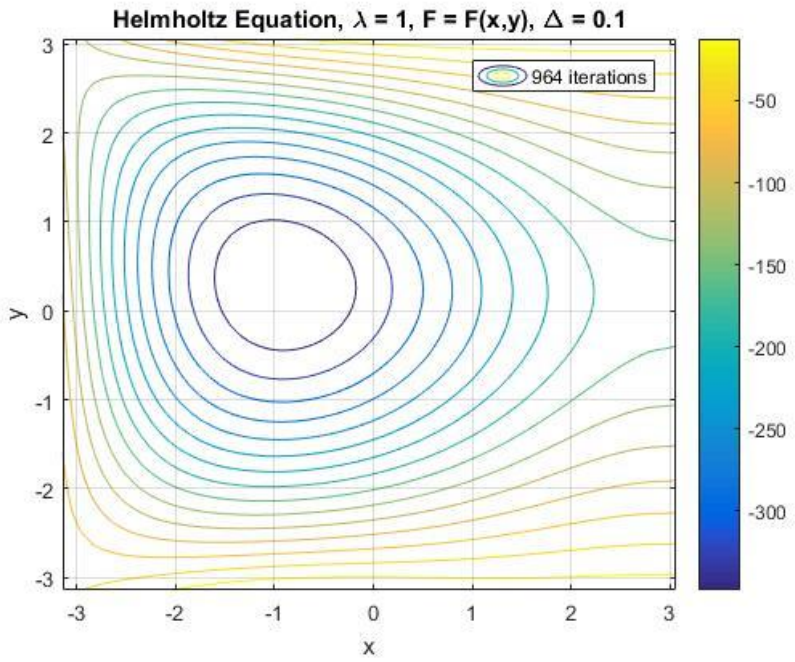


Figure 9. Contour of Helmholtz equation

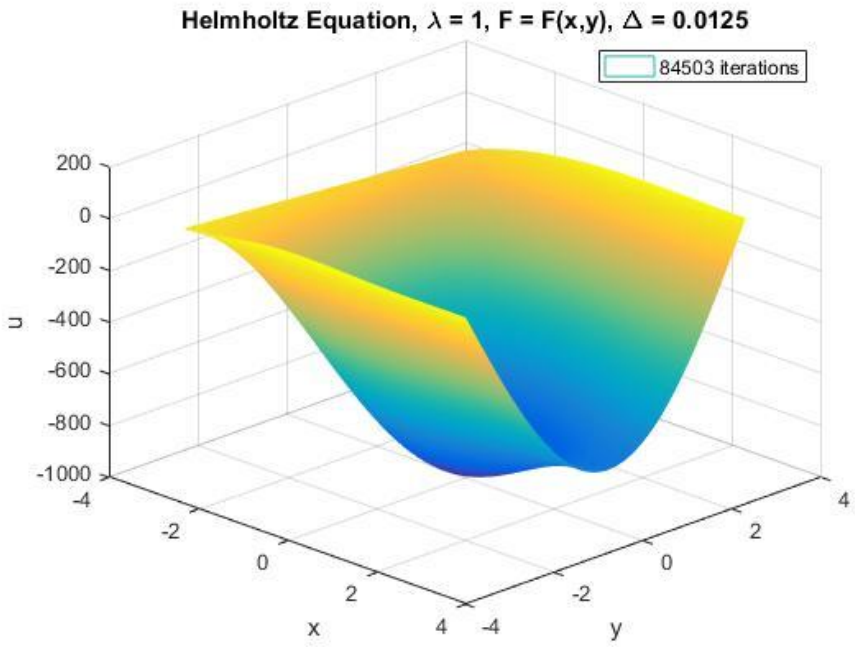


Figure 10. Mesh of Helmholtz equation

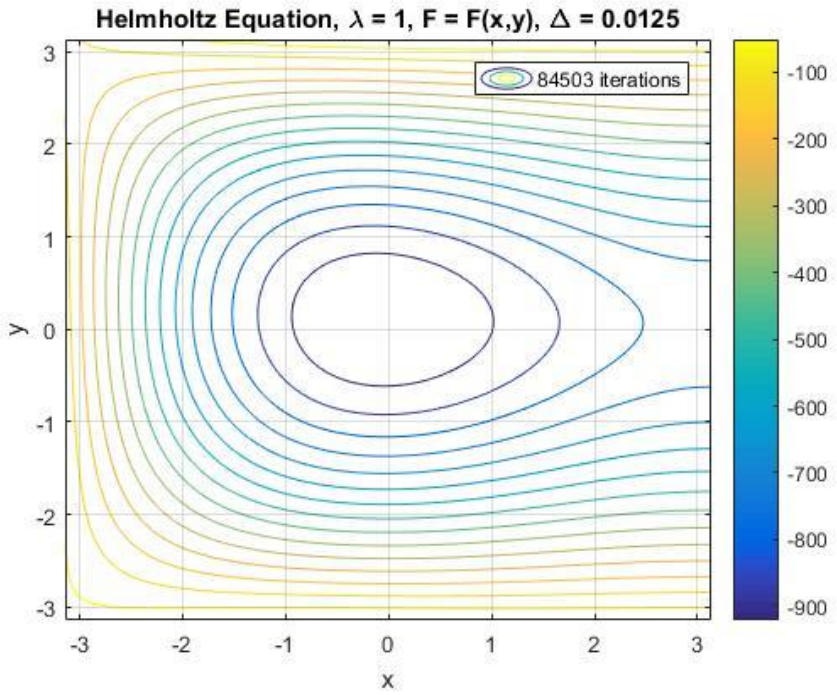


Figure 11. Contour of Helmholtz equation

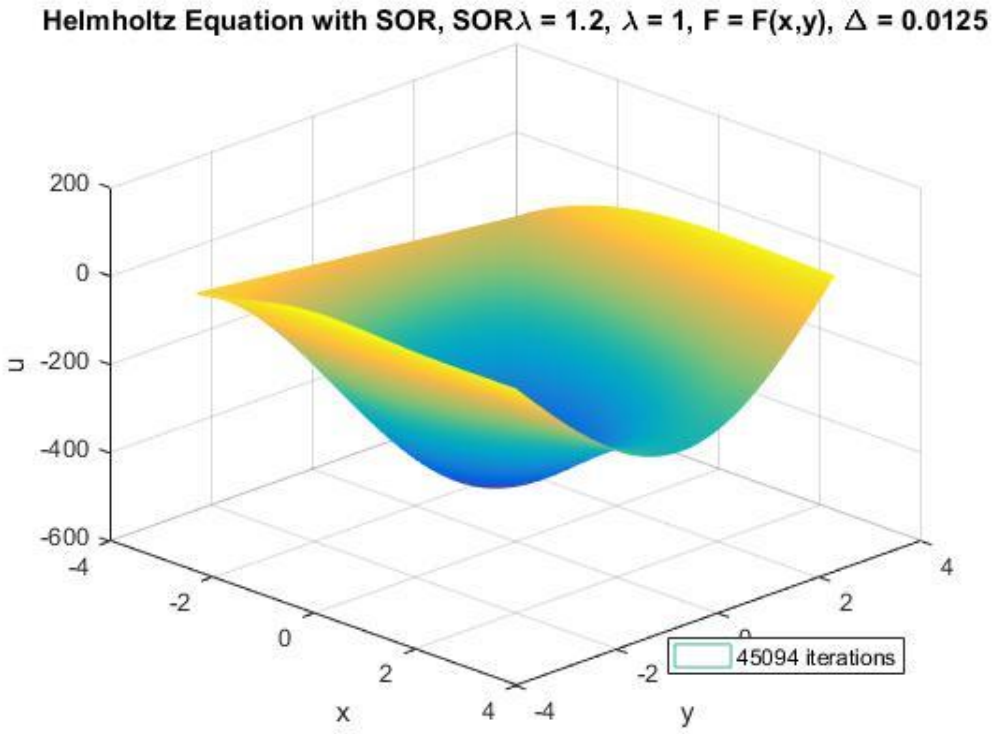


Figure 12. Mesh of Helmholtz equation with SOR

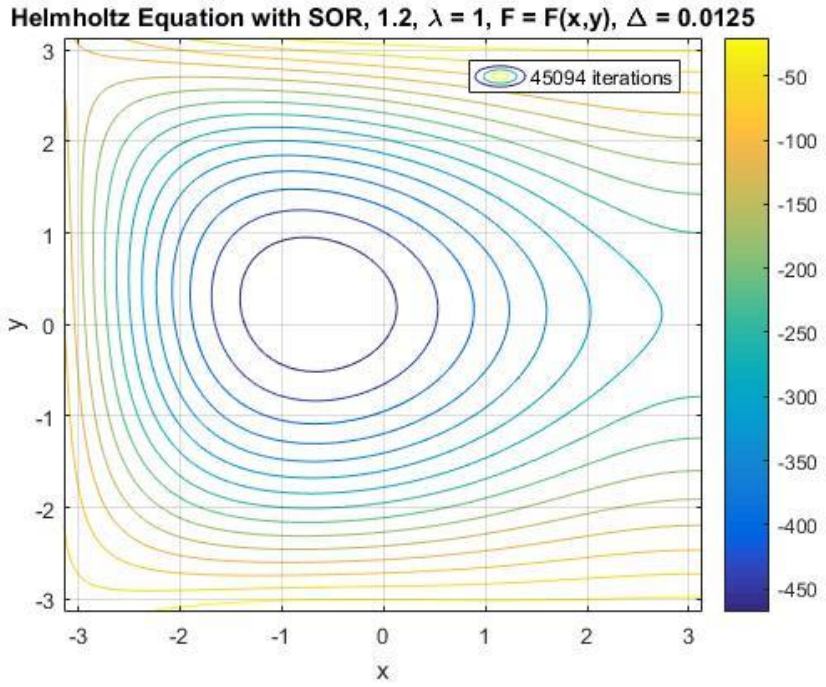


Figure 13. Contour of Helmholtz equation with SOR

The summary of results in Table 3 show that the maximum values remain essentially unchanged without regard to grid size, relaxation, or iteration count. The solver fails to converge even for a very fine mesh of over 250,000 nodes. This behavior was not observed for either the Laplace or Poisson equations, and so it seems to be a consequence of the term Λu affecting the Gauss-Seidel solution method.

Table 3. Results for Helmholtz Equation

Helmholtz		Gauss-Seidel			SOR		
Δ	# nodes	max	mean	min	max	mean	min
0.15	1755	11.6076	-383.614	-761.42	11.6076	-2188.6	-5195.4
0.1	3948	11.6244	-170.721	-372.087	11.6244	-224.167	-462.064
0.05	15791	11.6369	-259.296	-534.755	11.6369	-276.696	-560.354
0.025	63165	11.6369	-514.55	-1004.4	11.6369	-526.96	-1023.5
0.0125	252662	11.6378	-503.868	-982.956	11.6378	-242.328	-500.082

To further test the effect of the Λu term, the sign was flipped and the case with $\Lambda = -1$ was implemented. Figures 14-17 and Table 4 show the effect of this change. By reversing the sign, the dip vanishes and the numerical implementation of the Helmholtz equation demonstrates grid independence.

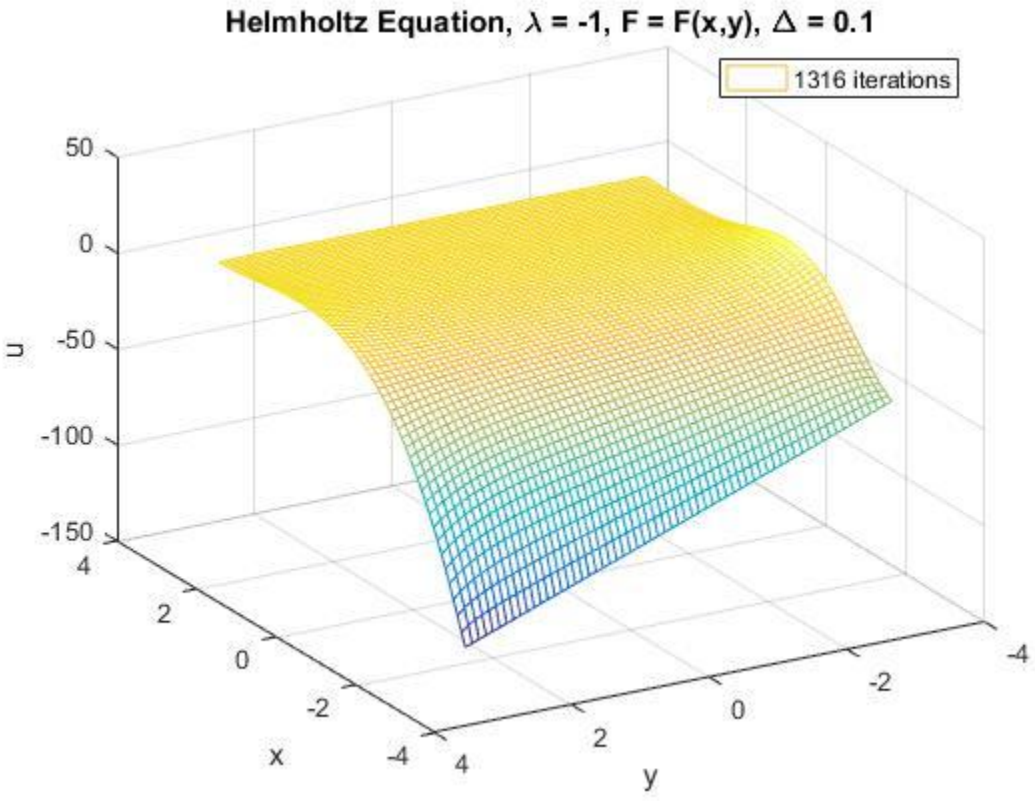


Figure 14. Mesh of Helmholtz equation with negative lambda

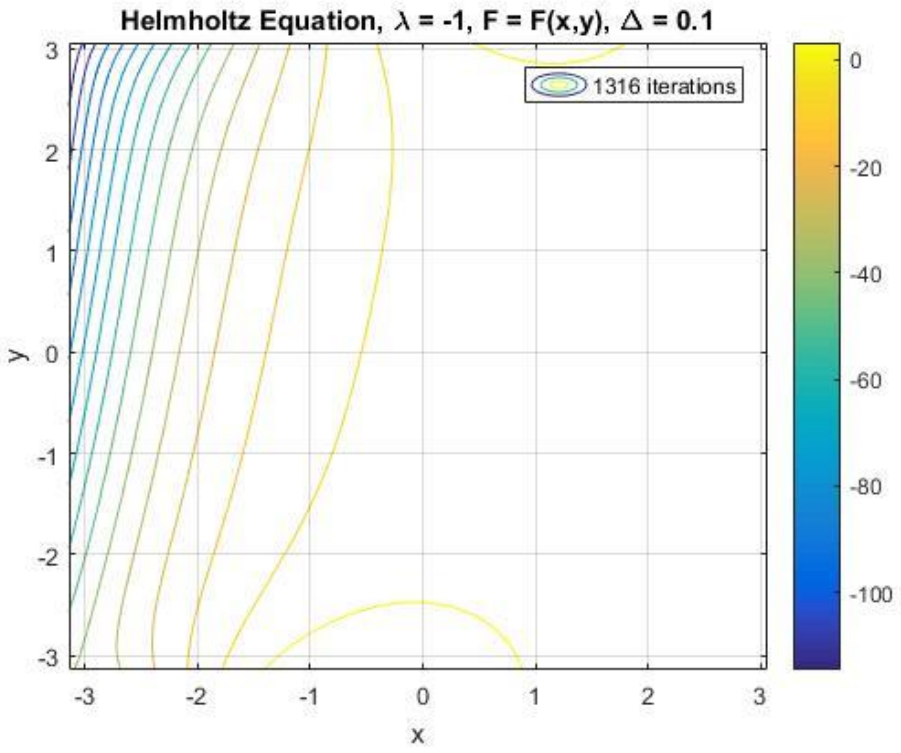
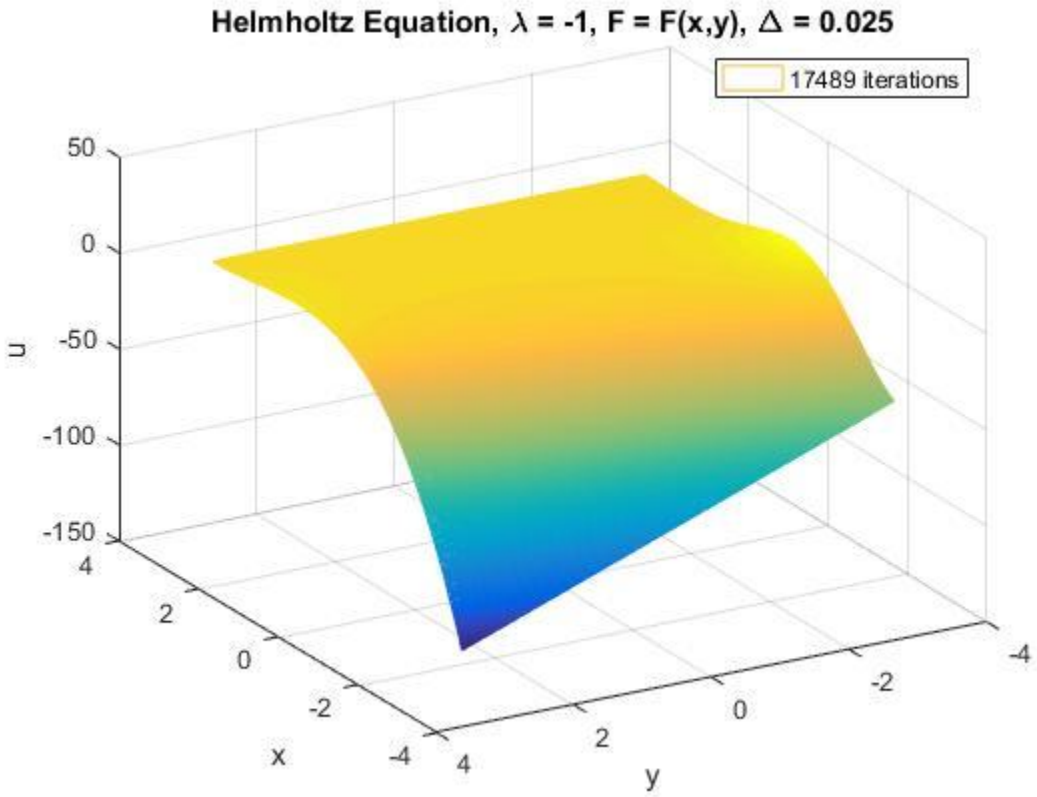


Figure 15. Contour of Helmholtz equation with negative lambda



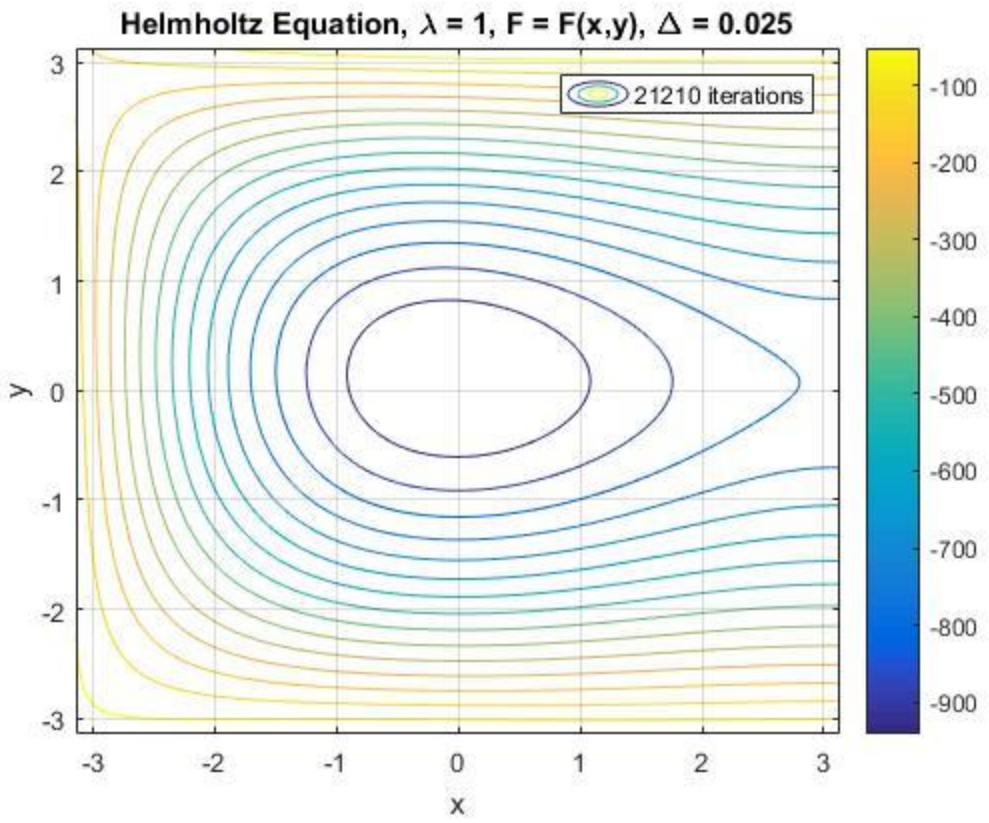


Table 4 shows that even when quartering the step size (resulting in a 16-fold increase in the number of nodes), the difference in the values is on the order of a couple of percentage points or less.

Table 4. Helmholtz equation with negative lambda

Helmholtz (Λ = -1)		Gauss-Seidel			SOR		
Δ	# nodes	max	mean	min	max	mean	min
0.1	3948	11.6244	-12.5682	-122.906	11.6244	-12.8613	-122.906
0.025	63165	11.6369	-12.1521	-123.915	11.6369	-12.4471	-123.915

With these studies it appears that by introducing a positive value of Λ, the Gauss-Seidel iteration method goes to negative infinity as the number of iterations increases due to the Neumann condition. The zero flux condition continues to be met at the boundary but the magnitude is driven downward with each iteration.

Figure 16 shows the difference in the infinity norm of the u matrix between successive iterations for the Gauss-Seidel method with and without SOR. The relaxation consistently reduces the number iterations needed before the error threshold is reached, which is the intended function of the overrelaxation.

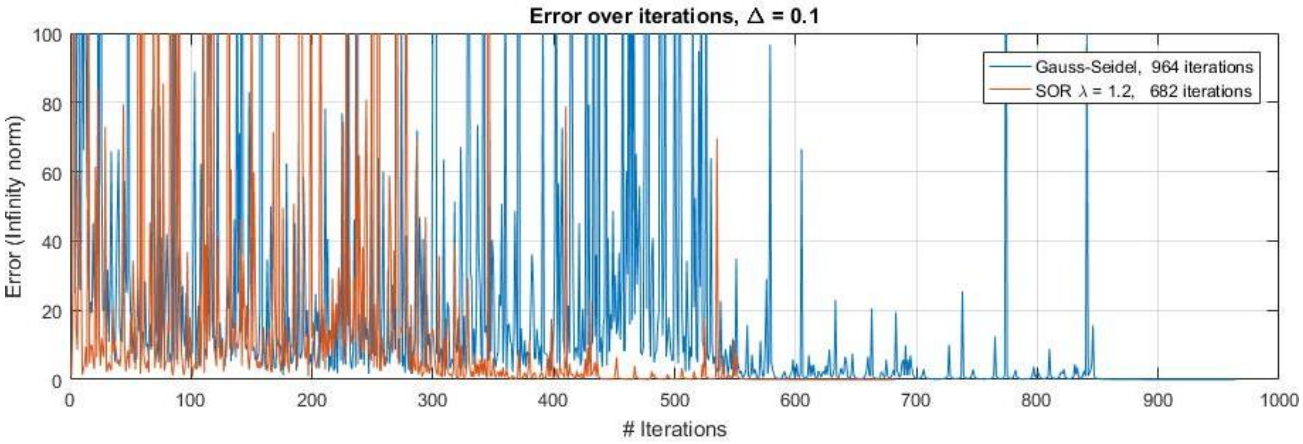


Figure 16. Error as the solver iterates

Table 5 shows a comparison of the number of iterations needed between relaxed and unrelaxed methods. With an SOR $\lambda = 1.2$, there's a consistent improvement of nearly a factor of 2 in the number of iterations needed to meet the error threshold.

Table 5. Comparison of iterations with and without SOR

Helmholtz		Gauss-Seidel	SOR (λ = 1.2)
Δ	# nodes	# iterations	# iterations
0.1	3948	964	682
0.05	15791	4421	2923
0.025	63165	21210	13903
0.0125	252662	84503	45094

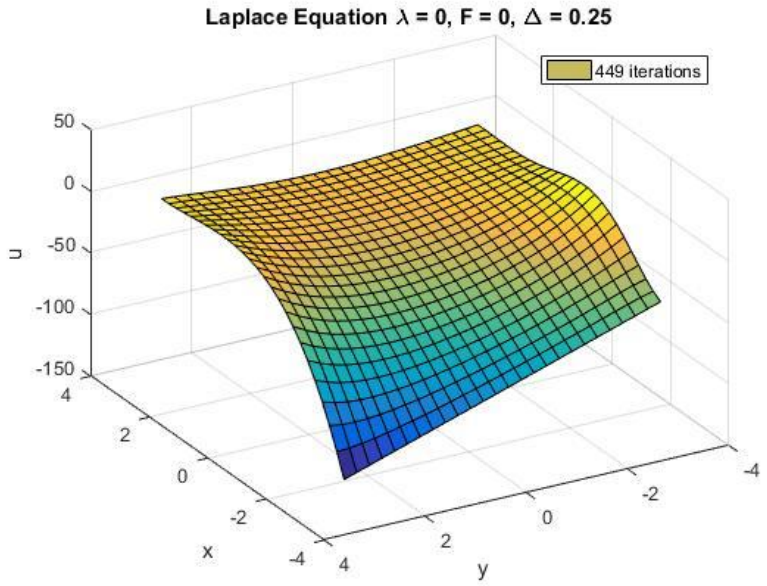
Table 6 shows a step through of different relaxation parameters for an element size of 0.025. The number of iterations needed decreases continually until reaching 2, where it suddenly increases in size. There are limitations on the extent to which relaxation can be used before it destabilizes the result.

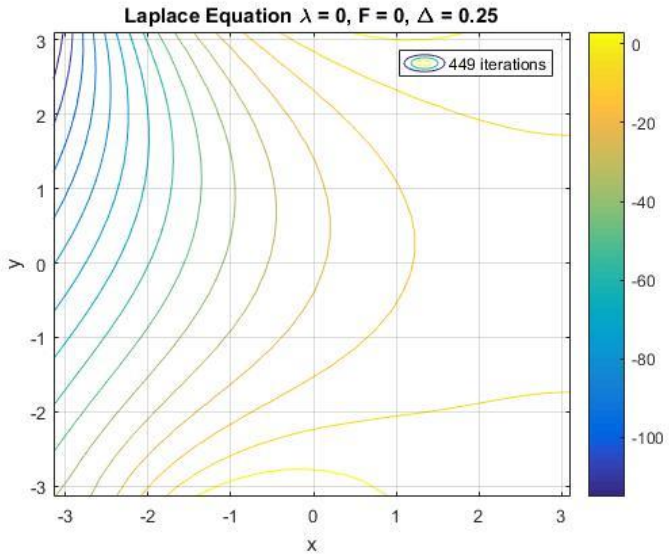
Table 6. Sweep through SOR weights

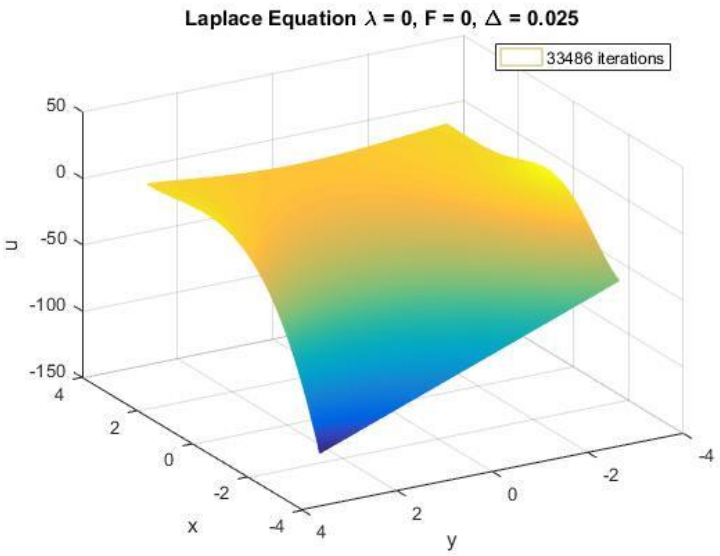
Δ = 0.025	
SOR λ	# iterations
1	17183
1.1	14078
1.2	13903
1.3	11255
1.4	8985
1.5	7019
1.6	5299
1.7	3783
1.8	2441

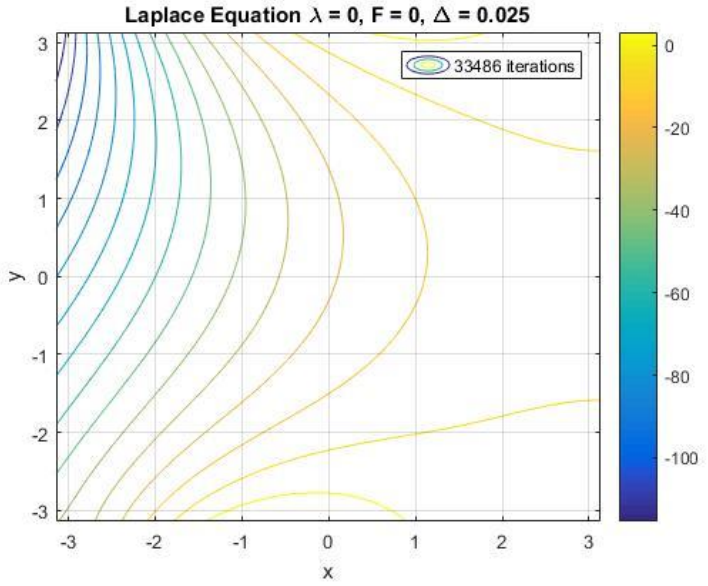
1.9	1258
2	24280

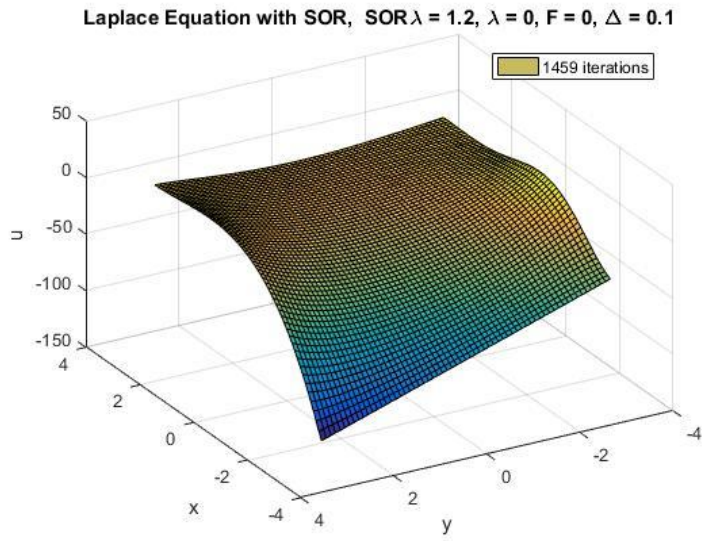
APPENDIX A: ADDITIONAL PLOTS AND FIGURES

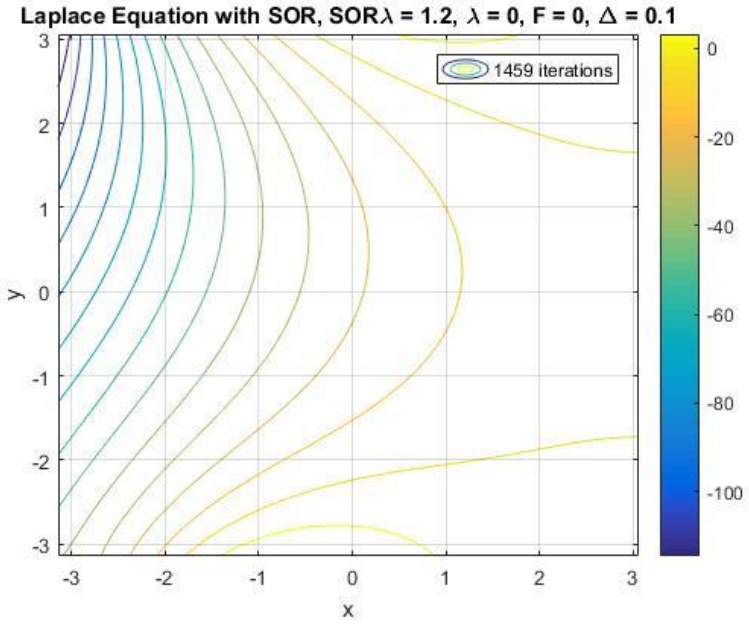


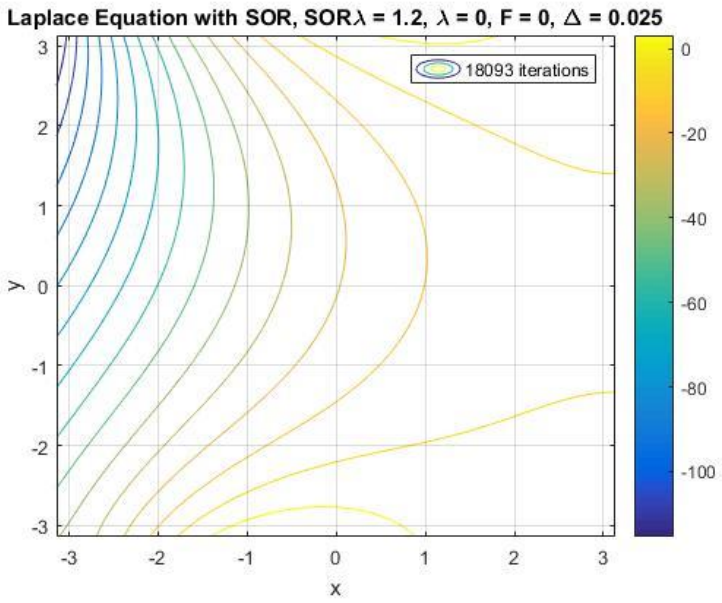


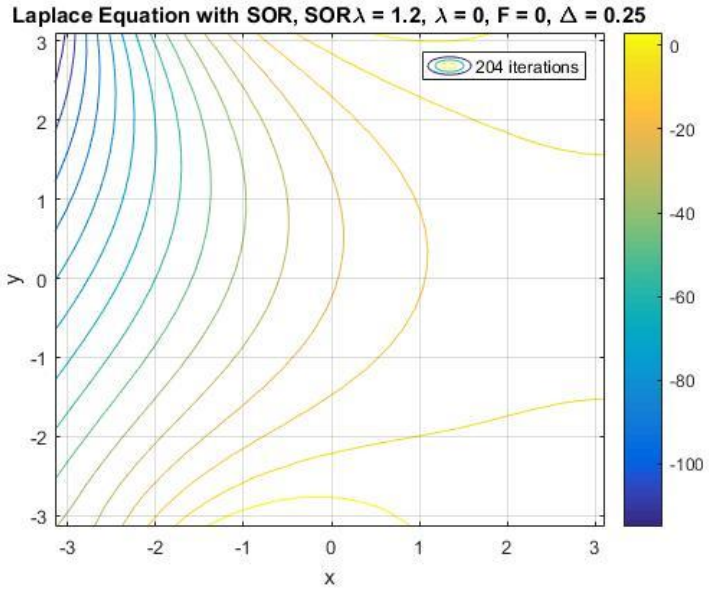

Laplace Equation λ = 0, F = 0, Δ = 0.025
33486 iterations
u
50
0
-50
-100
-150
x
4
2
0
-2
-4
y
4
2
0
-2
-4

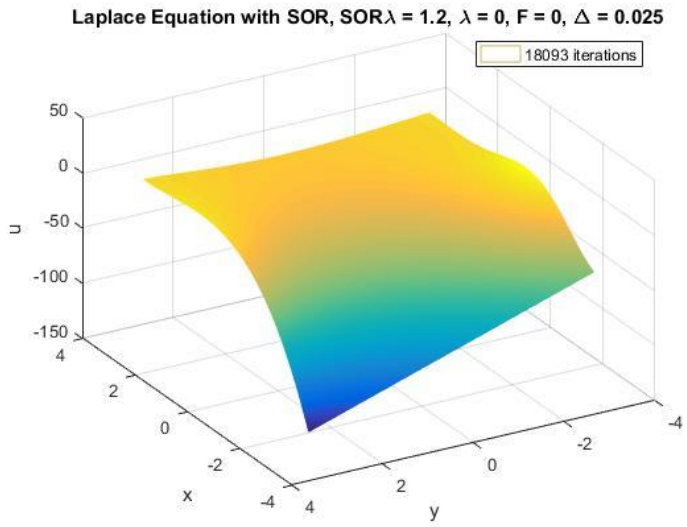

Laplace Equation λ = 0, F = 0, Δ = 0.025
33486 iterations
y
3
2
1
0
-1
-2
-3
x
-3
-2
-1
0
1
2
3
0
-20
-40
-60
-80
-100

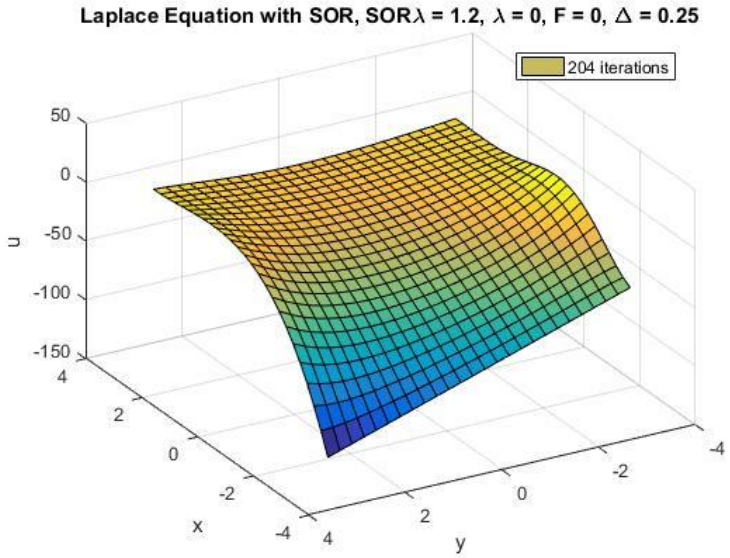

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.1
1459 iterations
u
50
0
-50
-100
-150
x
y

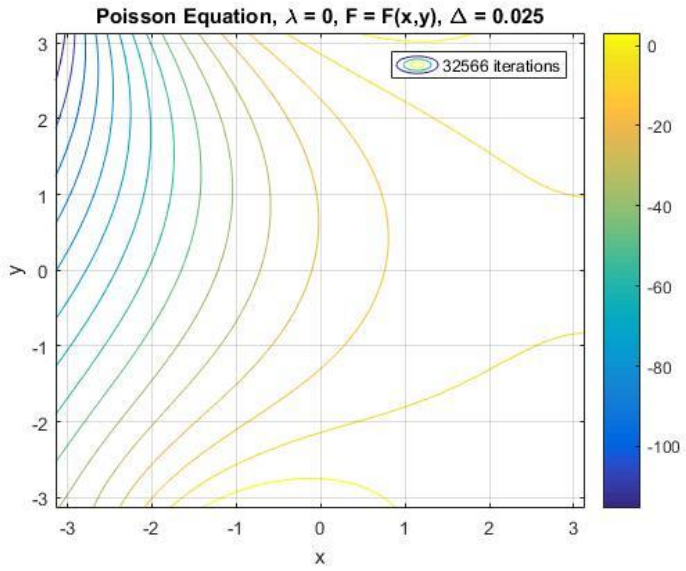

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.1
1459 iterations
x
y

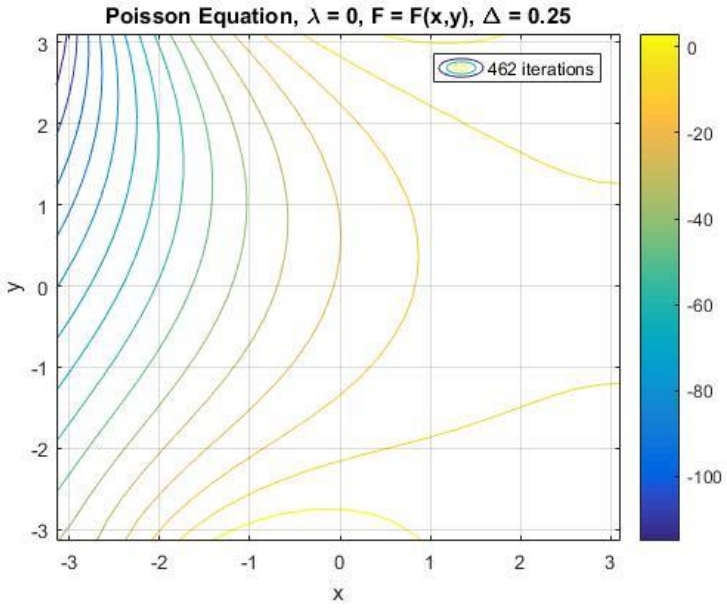

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.025
18093 iterations
x
y

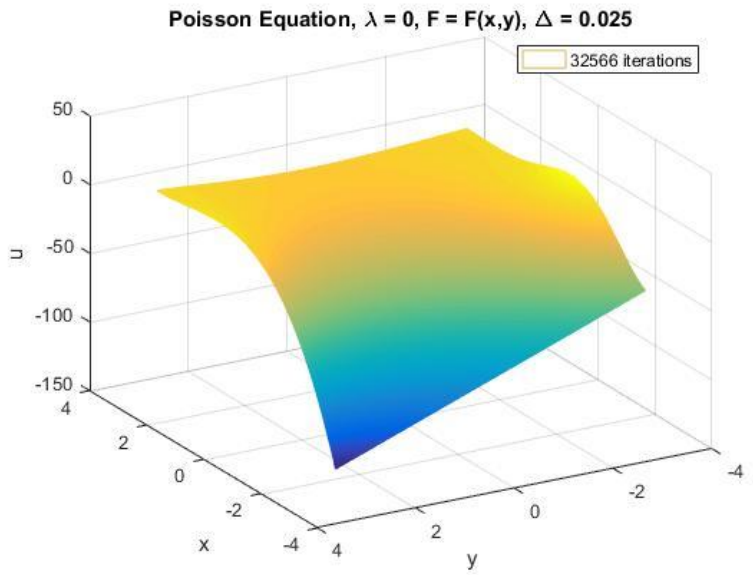

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.25
204 iterations
x
y

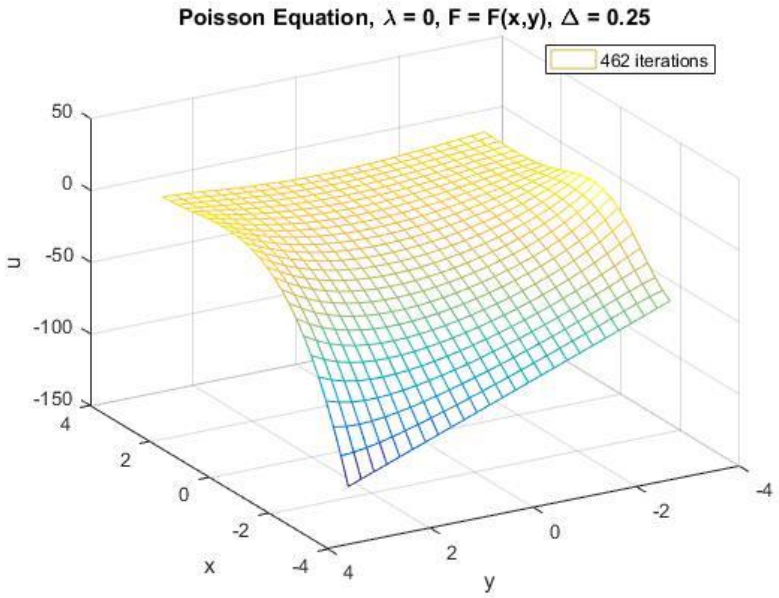

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.025
18093 iterations
50
0
-50
-100
-150
u
4
2
0
-2
-4
x
4
2
0
-2
-4
y

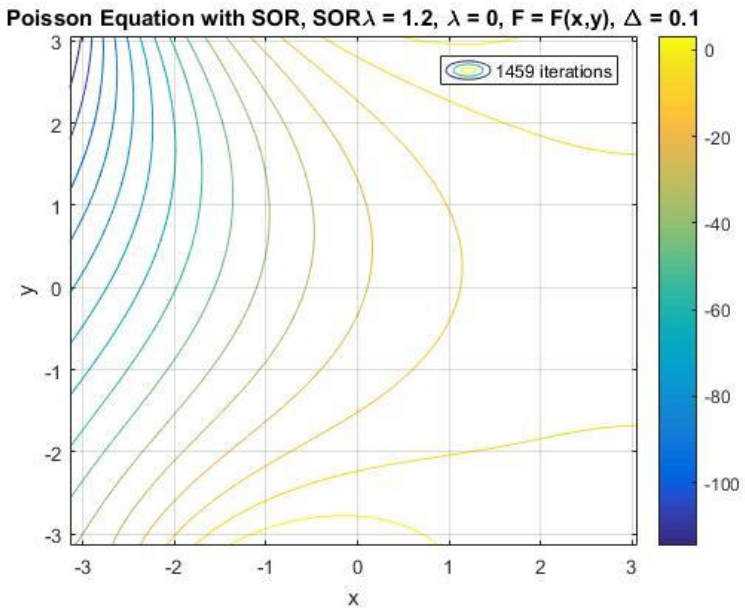

Laplace Equation with SOR, SORλ = 1.2, λ = 0, F = 0, Δ = 0.25
204 iterations
50
0
-50
-100
-150
u
4
2
0
-2
-4
x
4
2
0
-2
-4
y

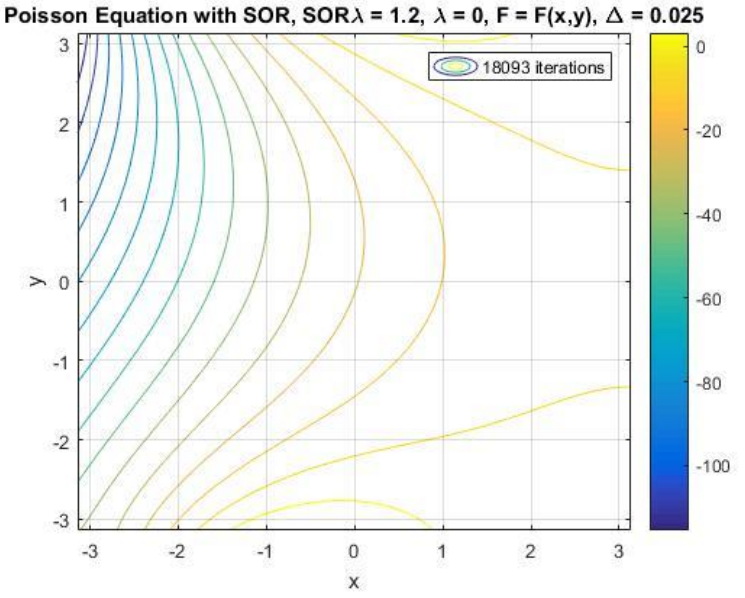

Poisson Equation, λ = 0, F = F(x,y), Δ = 0.025
32566 iterations
x
y

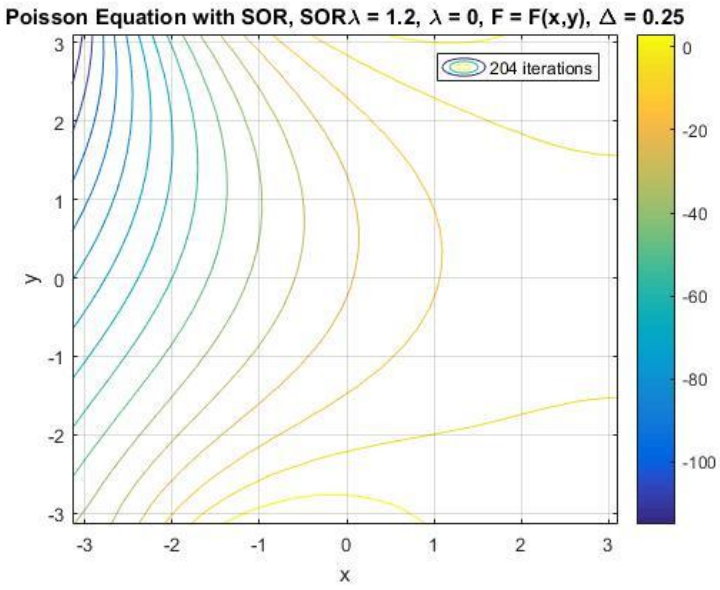

Poisson Equation, λ = 0, F = F(x,y), Δ = 0.25
462 iterations
x
y

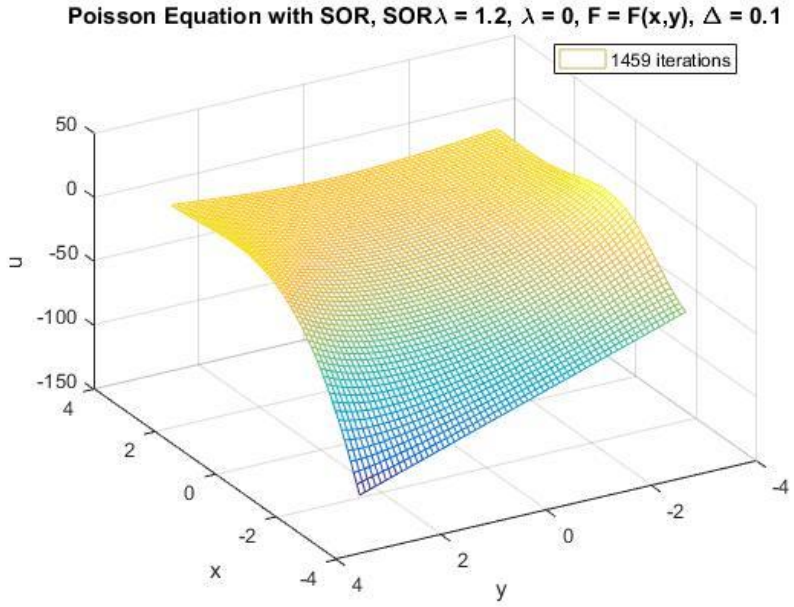

Poisson Equation, λ = 0, F = F(x,y), Δ = 0.025
32566 iterations
50
0
-50
-100
-150
u
4
2
0
-2
-4
x
4
2
0
-2
-4
y

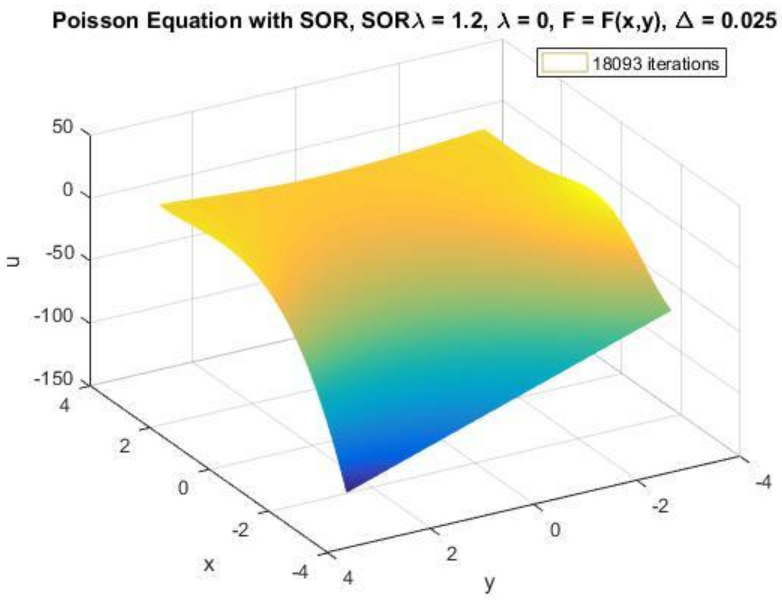

Poisson Equation, λ = 0, F = F(x,y), Δ = 0.25
462 iterations
50
0
-50
-100
-150
u
4
2
0
-2
-4
x
4
2
0
-2
-4
y

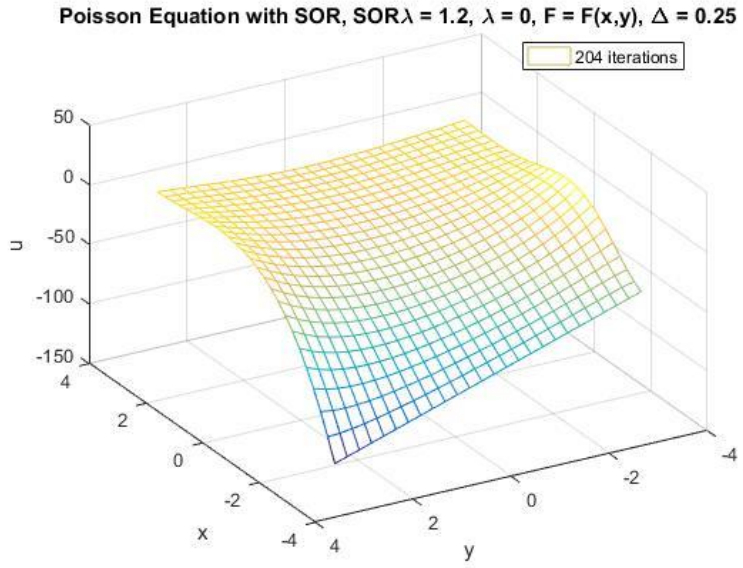

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.1
1459 iterations
x
y

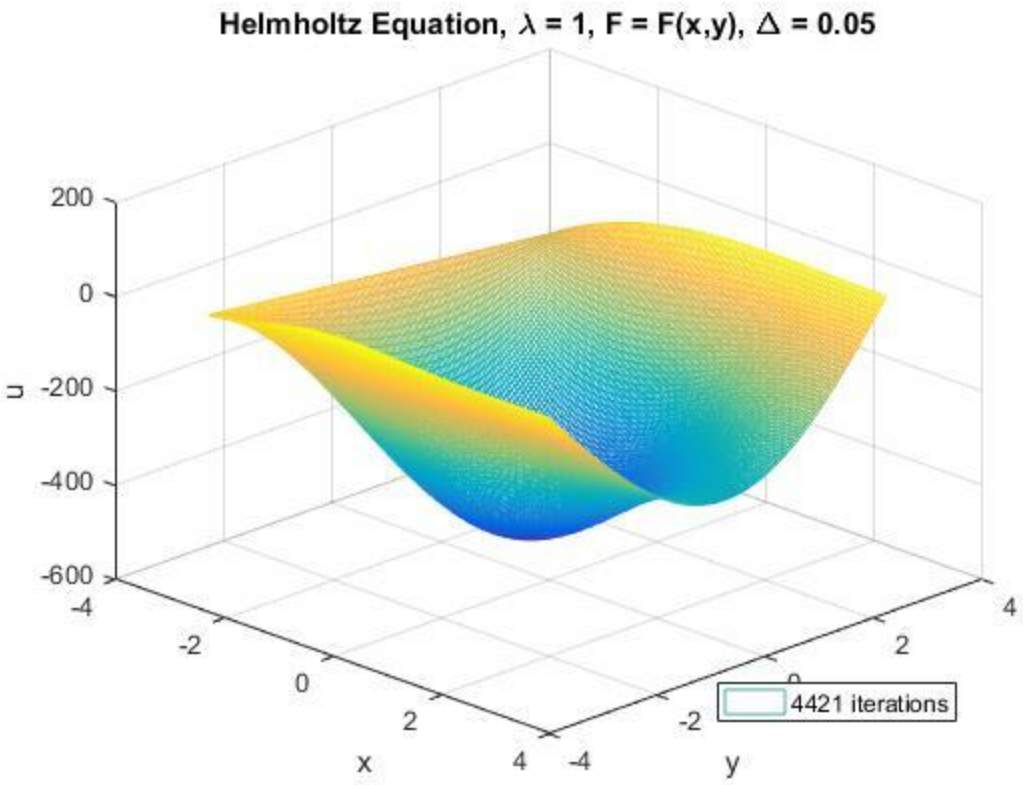

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.025
18093 iterations
x
y

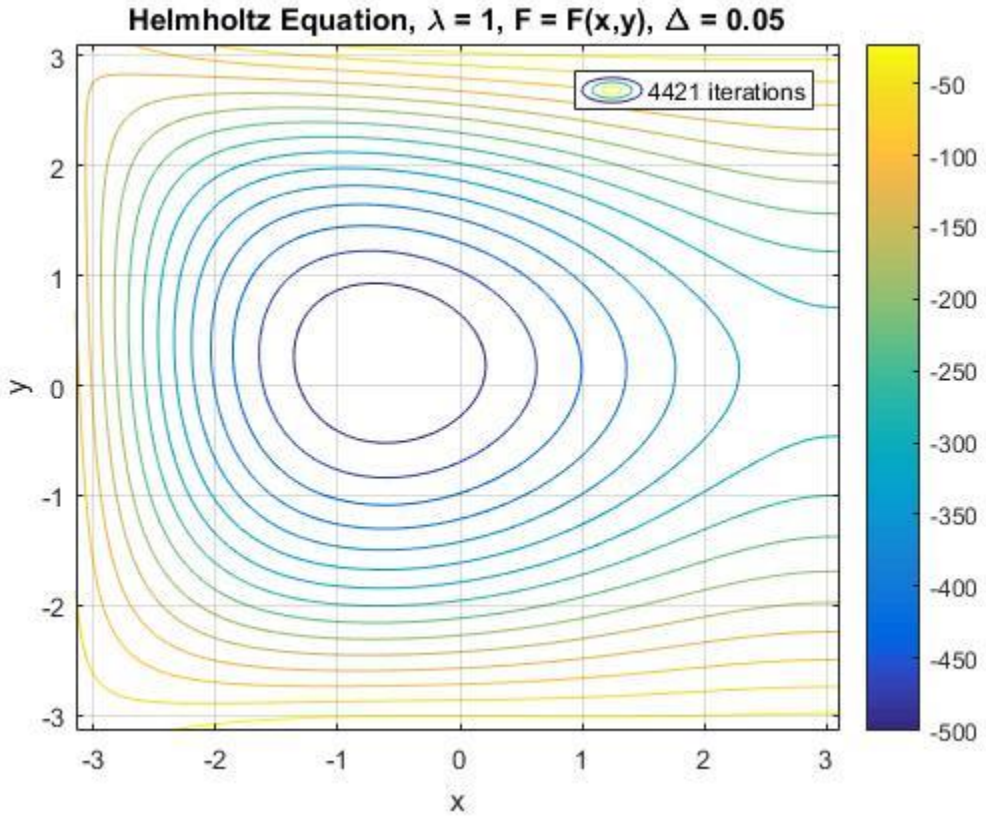

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.25
204 iterations
x
y

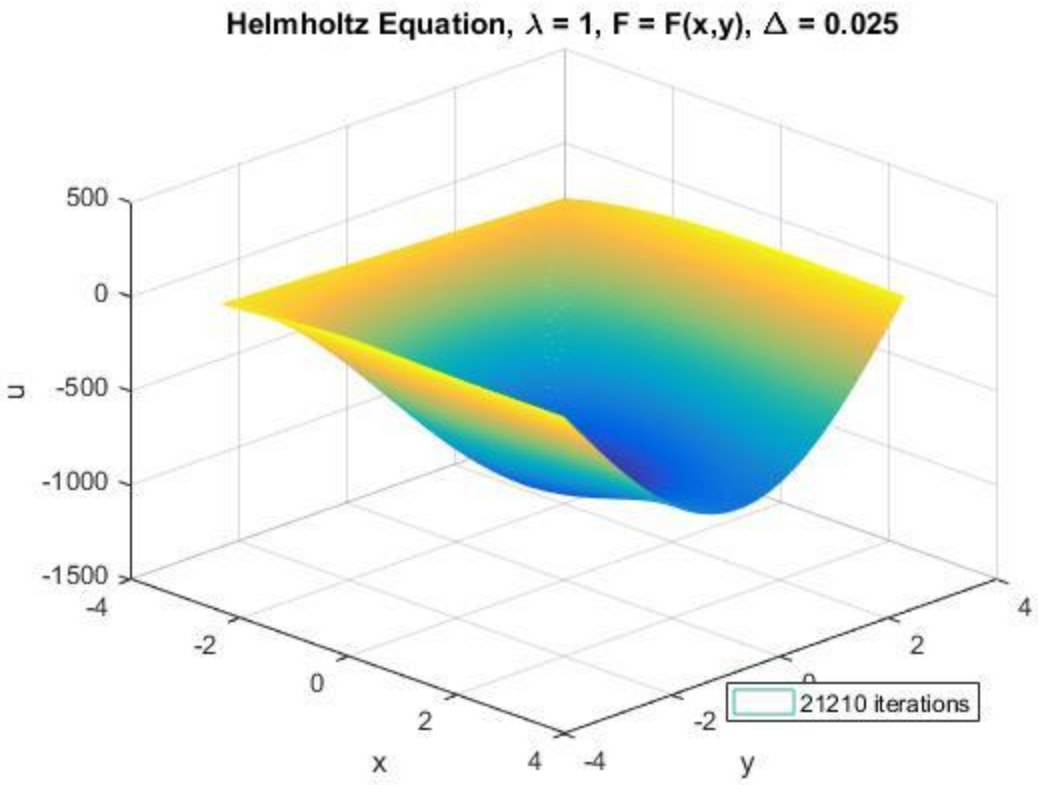

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.1
1459 iterations
u
x
y

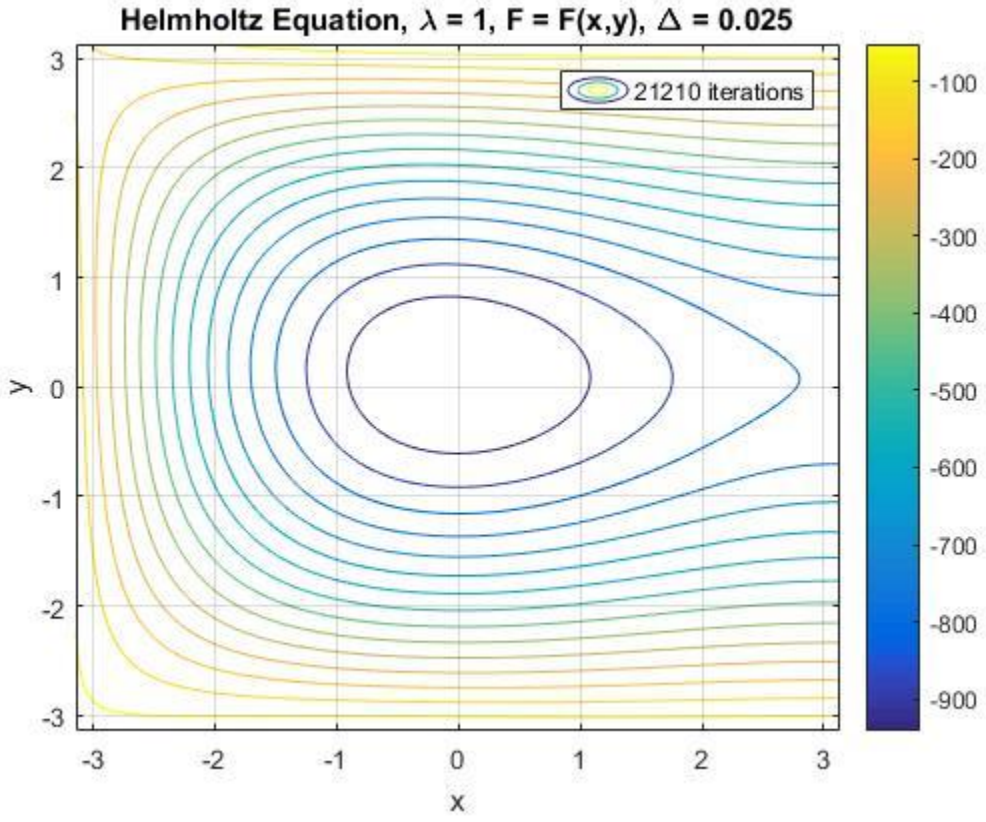

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.025
18093 iterations
u
50
0
-50
-100
-150
x
4
2
0
-2
-4
y
4
2
0
-2
-4

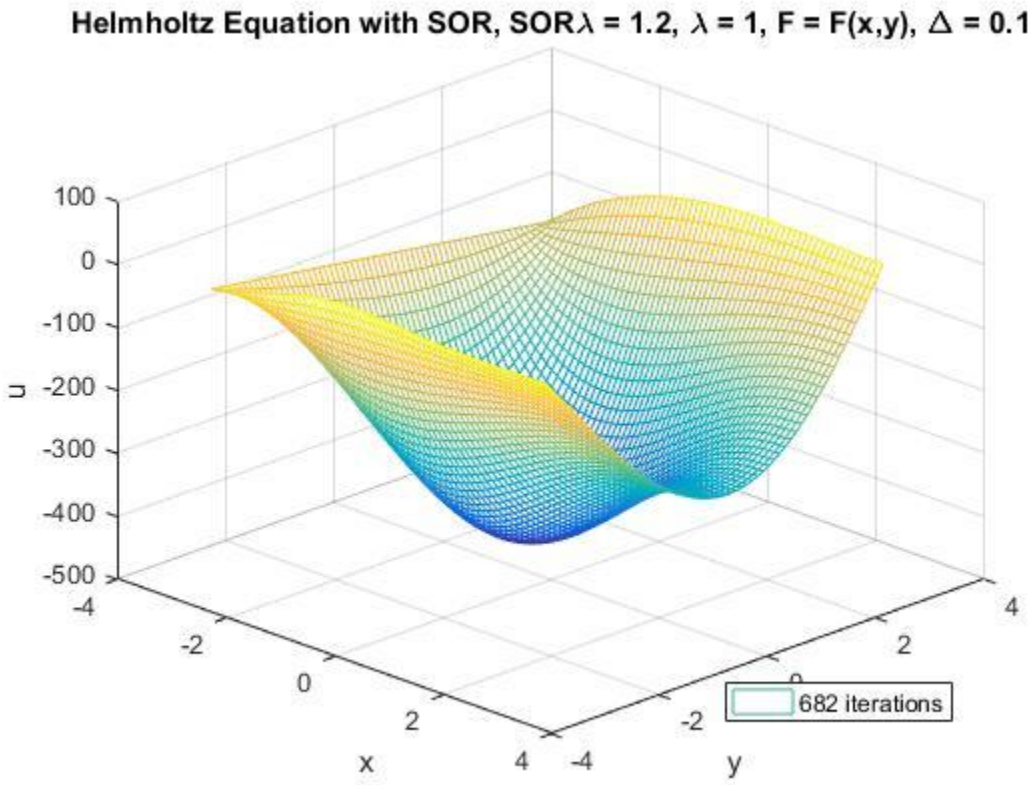

Poisson Equation with SOR, SORλ = 1.2, λ = 0, F = F(x,y), Δ = 0.25
204 iterations
u
50
0
-50
-100
-150
x
4
2
0
-2
-4
y
4
2
0
-2
-4

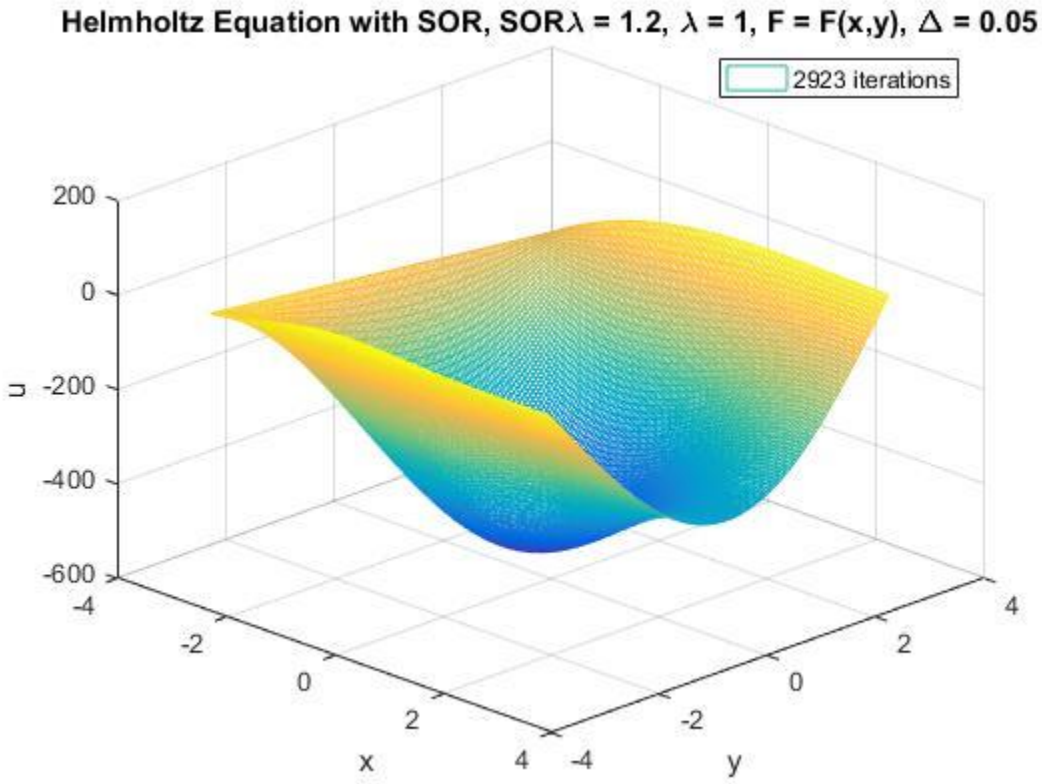

Helmholtz Equation, λ = 1, F = F(x,y), Δ = 0.05
200
0
-200
-400
-600
u
-4
-2
0
2
4
x
y
4421 iterations

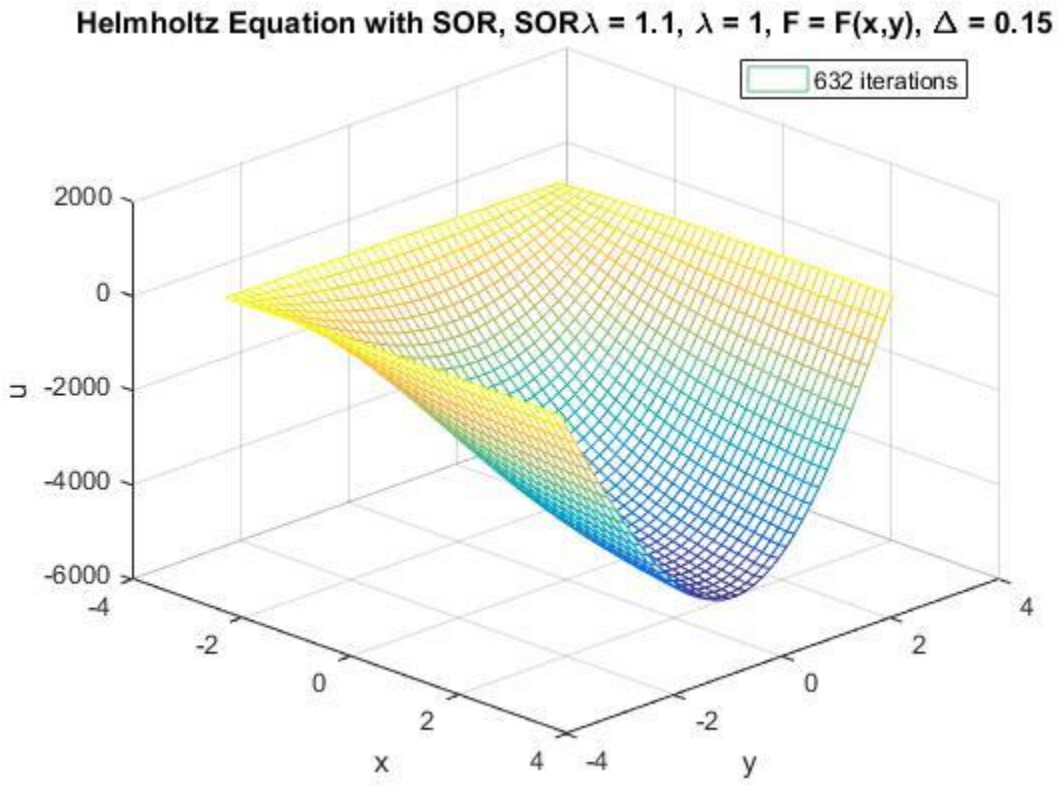

Helmholtz Equation, λ = 1, F = F(x,y), Δ = 0.05
4421 iterations
3
2
1
0
-1
-2
-3
y
x
-50
-100
-150
-200
-250
-300
-350
-400
-450
-500

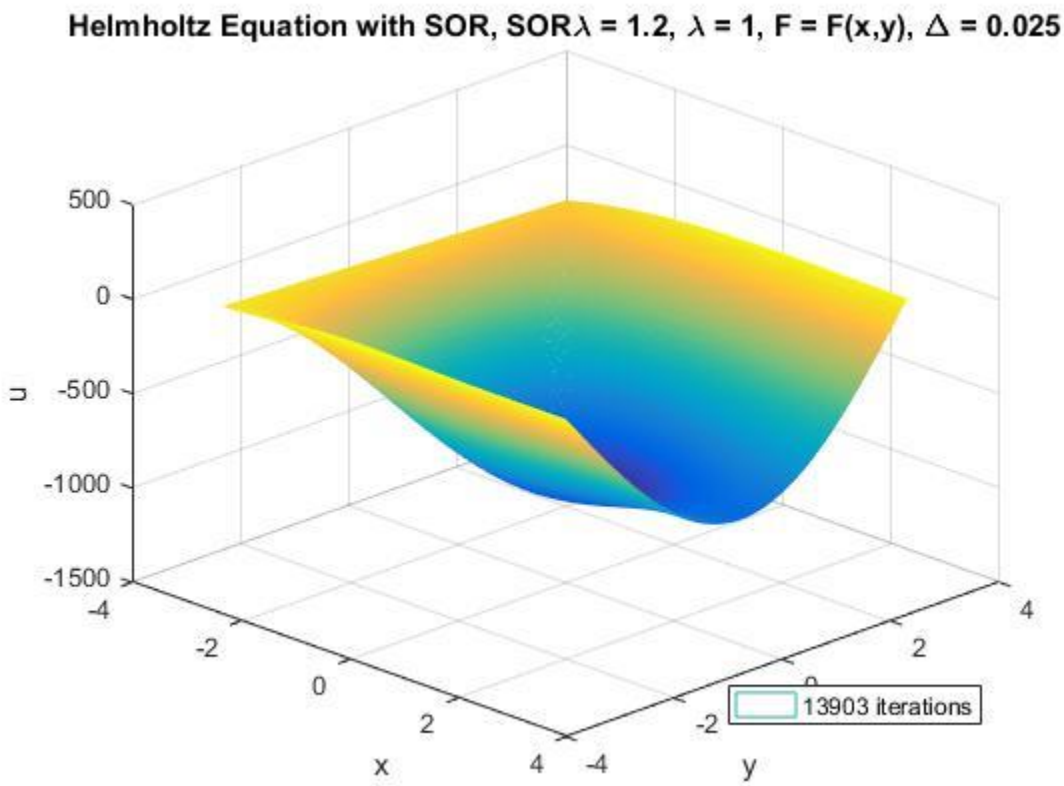

Helmholtz Equation, λ = 1, F = F(x,y), Δ = 0.025
500
0
-500
-1000
-1500
u
-4
-2
0
2
4
x
y
21210 iterations

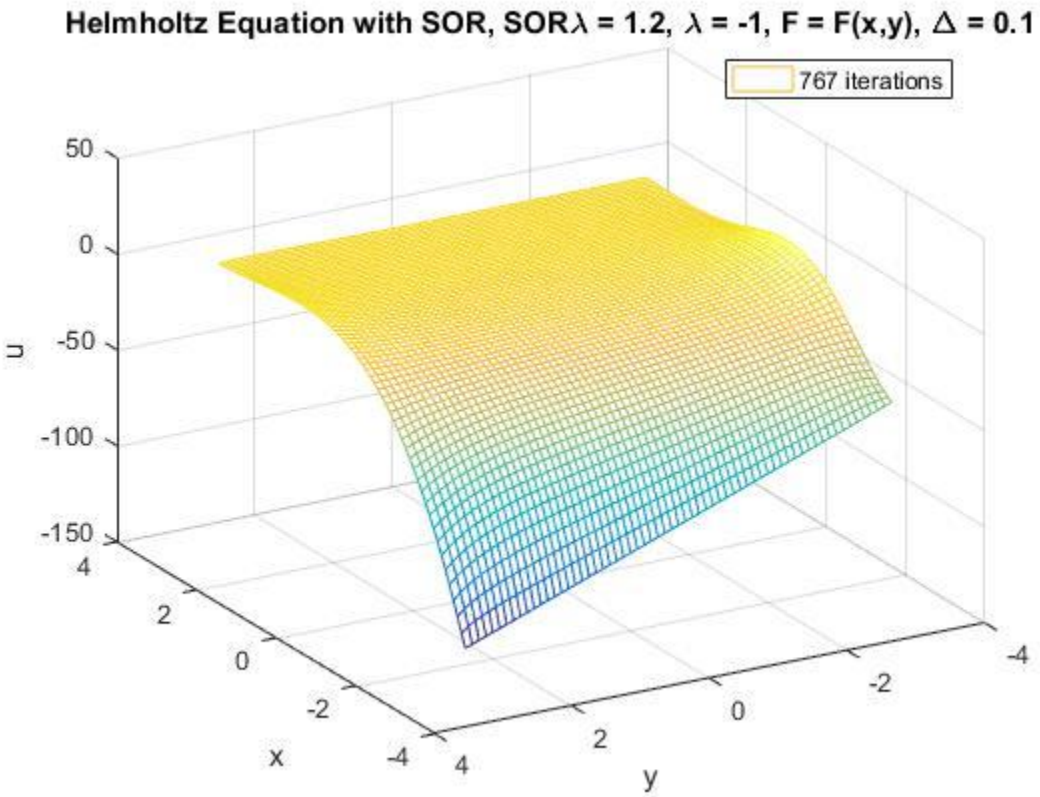

Helmholtz Equation, λ = 1, F = F(x,y), Δ = 0.025
21210 iterations
3
2
1
0
-1
-2
-3
y
x
-100
-200
-300
-400
-500
-600
-700
-800
-900

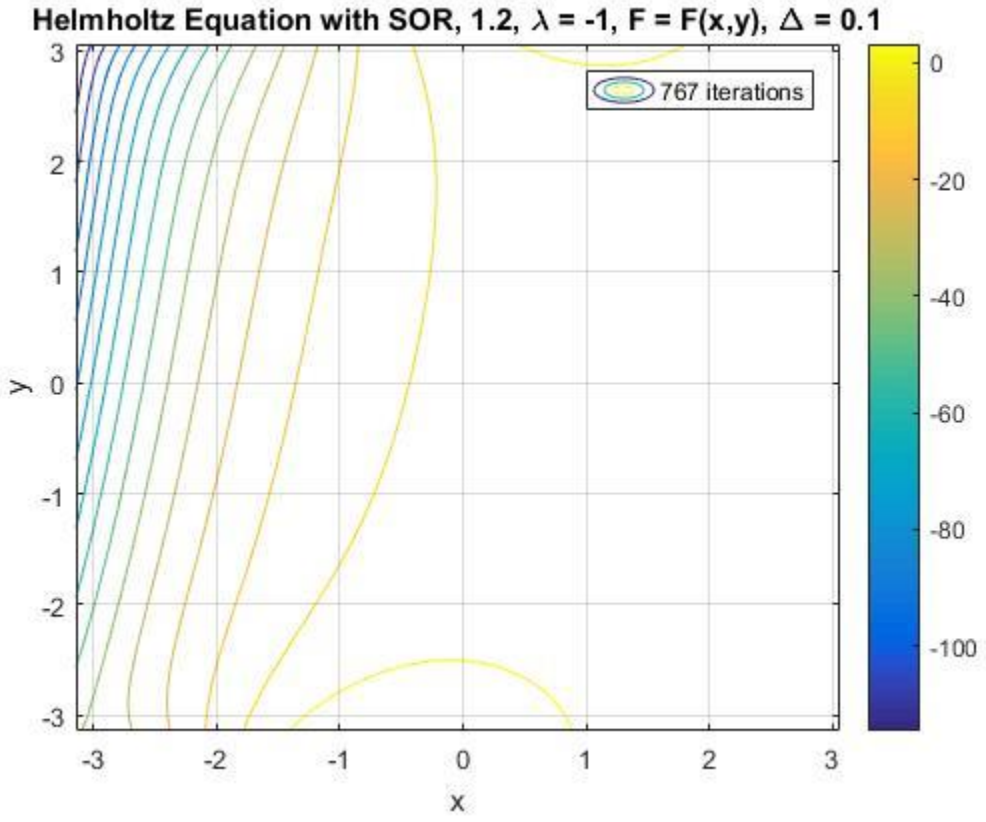

Helmholtz Equation with SOR, SORλ = 1.2, λ = 1, F = F(x,y), Δ = 0.1
u
x
y
682 iterations


Helmholtz Equation with SOR, SORλ = 1.2, λ = 1, F = F(x,y), Δ = 0.05
u
x
y
2923 iterations


Helmholtz Equation with SOR, SORλ = 1.1, λ = 1, F = F(x,y), Δ = 0.15
632 iterations
2000
0
-2000
-4000
-6000
u
-4
-2
0
2
4
x
y


Helmholtz Equation with SOR, SORλ = 1.2, λ = 1, F = F(x,y), Δ = 0.025
500
0
-500
-1000
-1500
u
-4
-2
0
2
4
x
y
13903 iterations


Helmholtz Equation with SOR, SORλ = 1.2, λ = -1, F = F(x,y), Δ = 0.1
767 iterations
u
50
0
-50
-100
-150
x
4
2
0
-2
-4
y
4
2
0
-2
-4


Helmholtz Equation with SOR, 1.2, λ = -1, F = F(x,y), Δ = 0.1
767 iterations
y
3
2
1
0
-1
-2
-3
x
-3
-2
-1
0
1
2
3
0
-20
-40
-60
-80
-100