Computational Physics
Fall 2020

Lalit Chaudhary

October 23, 2020

Laplace Problem on a Square

Contents

List of Figures

1 Laplace Equation

For a system of stationary charges, the electric field, $\vec{E}$ is related to the electric potential, V as

$$\vec{E}(\vec{r}) = -\vec{\nabla}V \tag{1}$$

Using Gauss' Law, we have the following identity:

$$\vec{\nabla} \cdot \vec{E} = \frac{\rho(\vec{r})}{\epsilon_0} \tag{2}$$

From equations (1) and (2), we have,

$$-\vec{\nabla}^2 V = \frac{\rho(\vec{r})}{\epsilon_0} \tag{3}$$

which is called the **Poisson's Equation**.

In a region with $\rho = 0$, the Poisson's Equation reduces to the **Laplace Equation** given by:

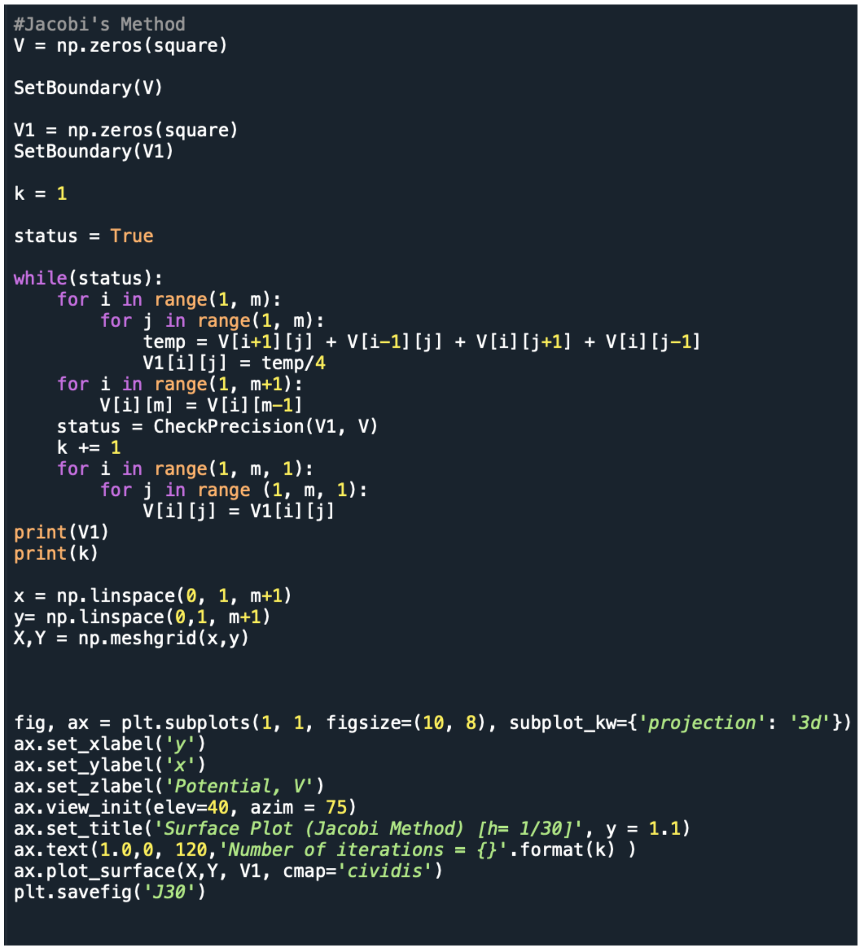
$$\vec{\nabla}^2 V = 0 \tag{4}$$

The **Uniqueness Theorem** states that a potential distribution obeying Poisson's equation (and consequently, the Laplace Equation) is completely specified within a volume V if the potential is specified over the boundary of V.

For the given setup of Laplace problem on a square with top side at potential $100V$, the left and bottom grounded and the right boundary open, we find a solution using various iterative methods.

2 Jacobi's Method

The above method was employed for various grid spacing $h = 1/5, 1/10, 1/20$ and $1/30$. The results are shown below.



```python
#Jacobi's Method
V = np.zeros(square)

SetBoundary(V)

V1 = np.zeros(square)
SetBoundary(V1)

k = 1

status = True

while(status):
    for i in range(1, m):
        for j in range(1, m):
            temp = V[i+1][j] + V[i-1][j] + V[i][j+1] + V[i][j-1]
            V1[i][j] = temp/4
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]
print(V1)
print(k)

x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (Jacobi Method) [h= 1/30]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('J30')
```

Figure 1: Code Snippet for Jacobi's Method

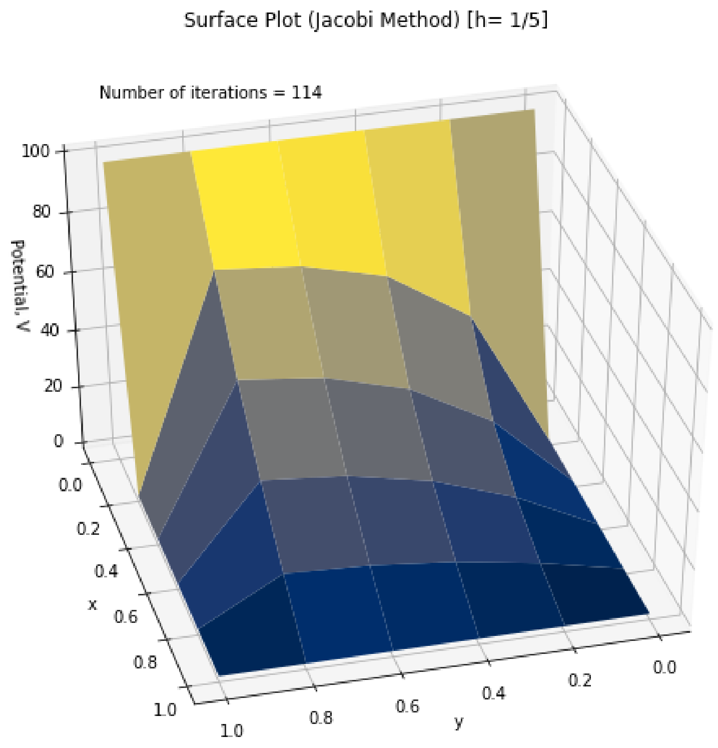


Figure 2: Variation of potential in the square using Jacobi's Method [h=1/5]

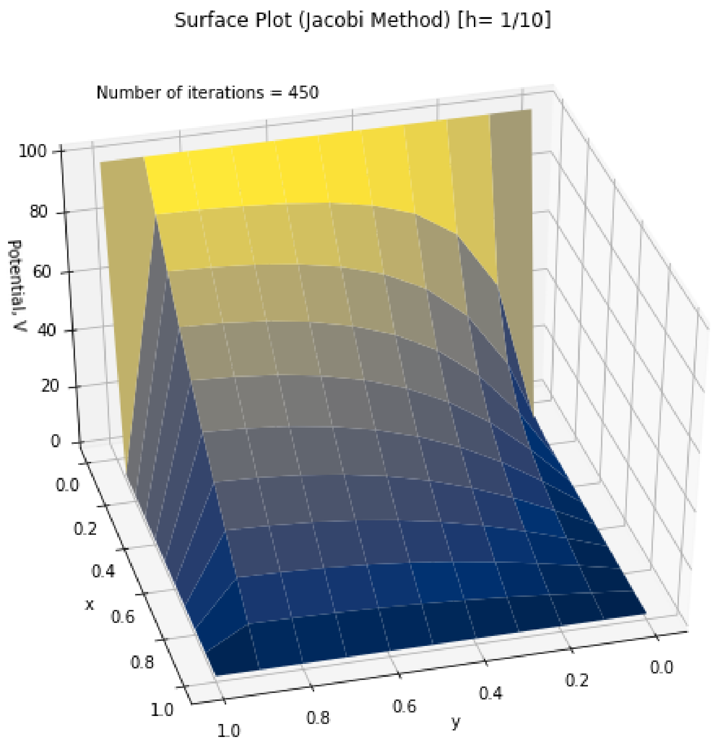


Figure 3: Variation of potential in the square using Jacobi's Method [h=1/10]

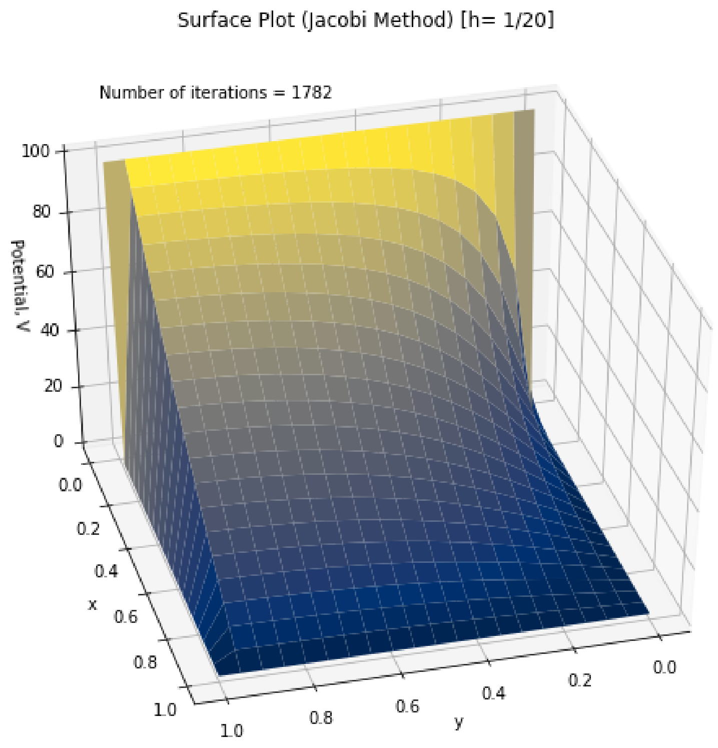


Figure 4: Variation of potential in the square using Jacobi's Method [h=1/20]

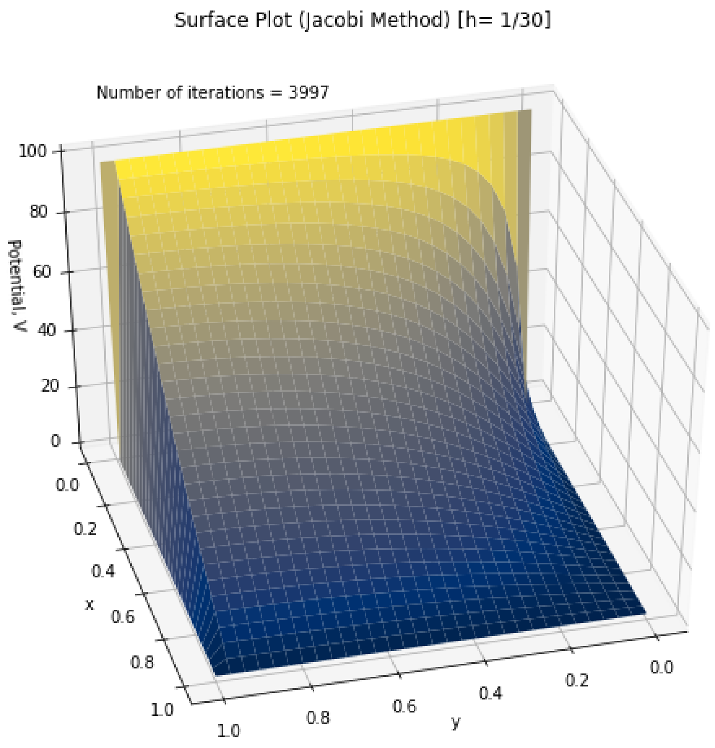
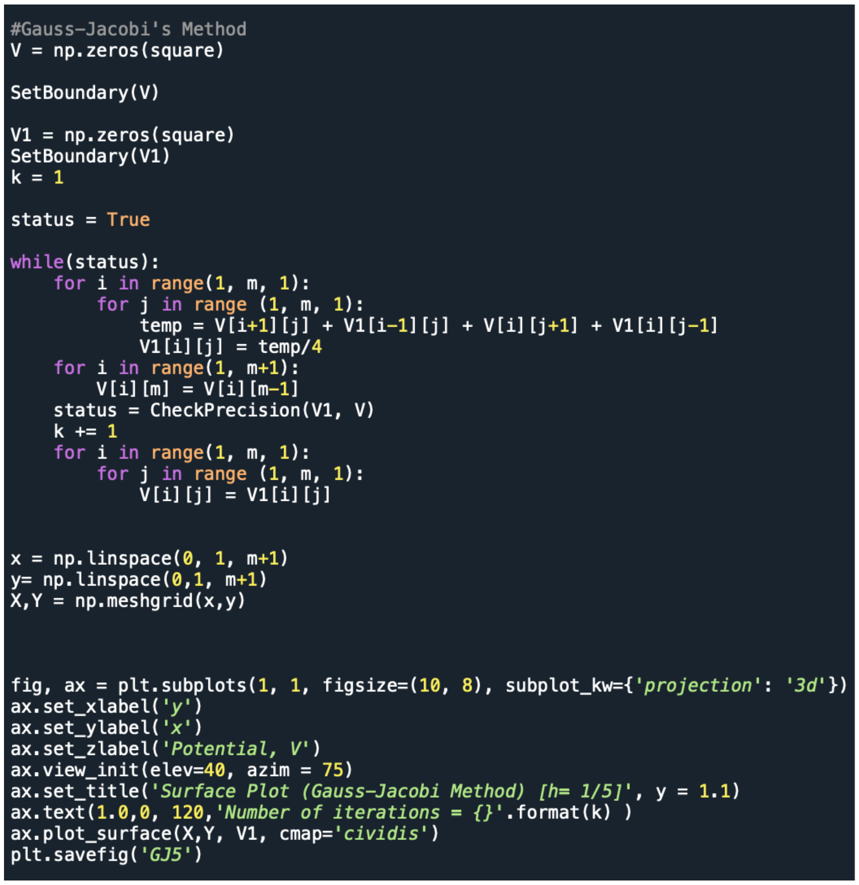


Figure 5: Variation of potential in the square using Jacobi's Method [h=1/30]

3 Gauss-Jacobi's Method



```
#Gauss-Jacobi's Method
V = np.zeros(square)

SetBoundary(V)

V1 = np.zeros(square)
SetBoundary(V1)
k = 1

status = True

while(status):
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            temp = V[i+1][j] + V1[i-1][j] + V[i][j+1] + V1[i][j-1]
            V1[i][j] = temp/4
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]


x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (Gauss-Jacobi Method) [h= 1/5]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('GJ5')
```

Figure 6: Code Snippet for Gauss-Jacobi's Method

The above method was employed for various grid spacing $h = 1/5, 1/10, 1/20$ and $1/30$. The results are shown below.

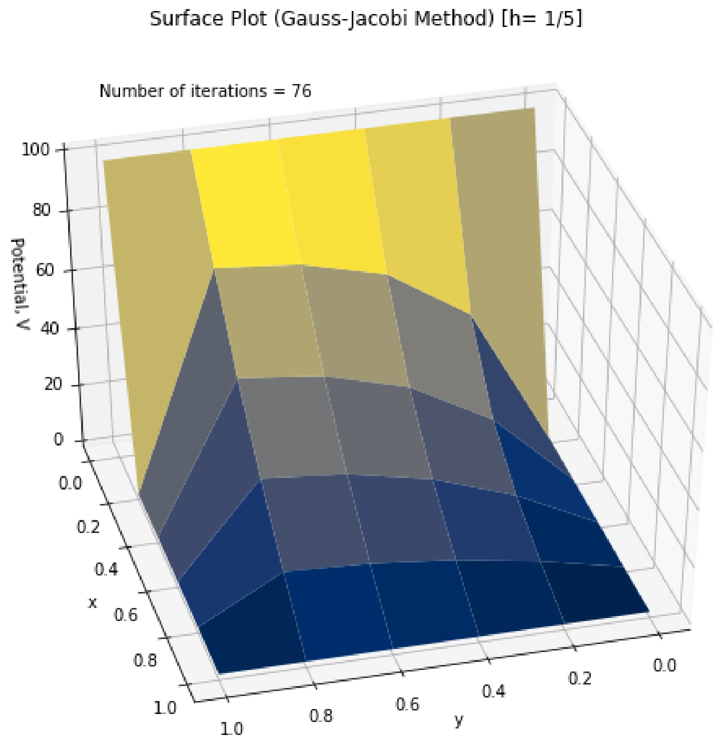


Figure 7: Variation of potential in the square using Gauss-Jacobi's Method [h=1/5]

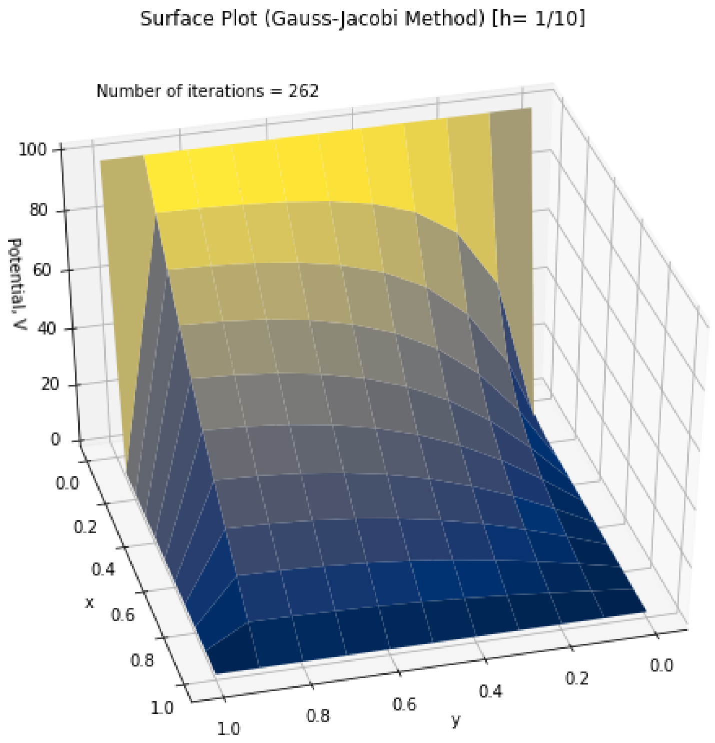


Figure 8: Variation of potential in the square using Gauss-Jacobi's Method [h=1/10]

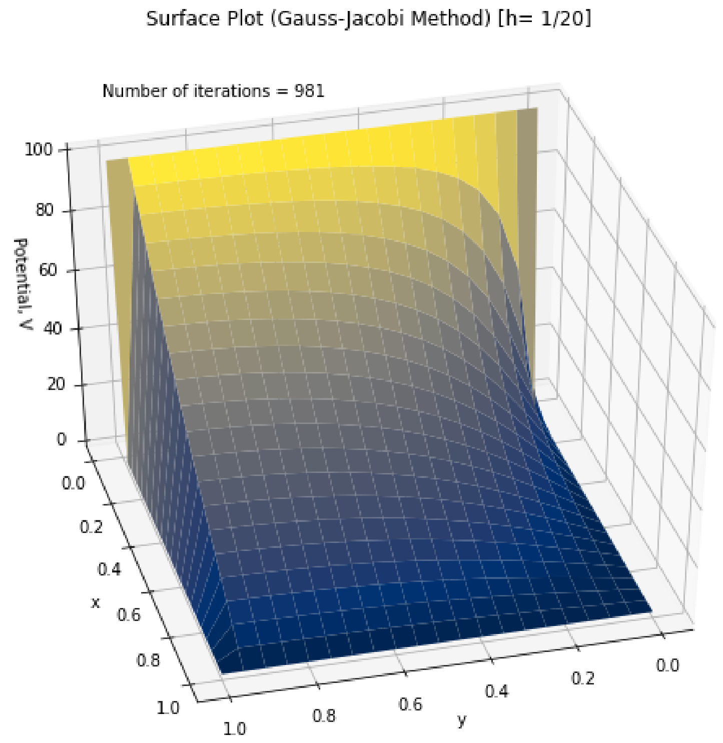


Figure 9: Variation of potential in the square using Gauss-Jacobi's Method [h=1/20]

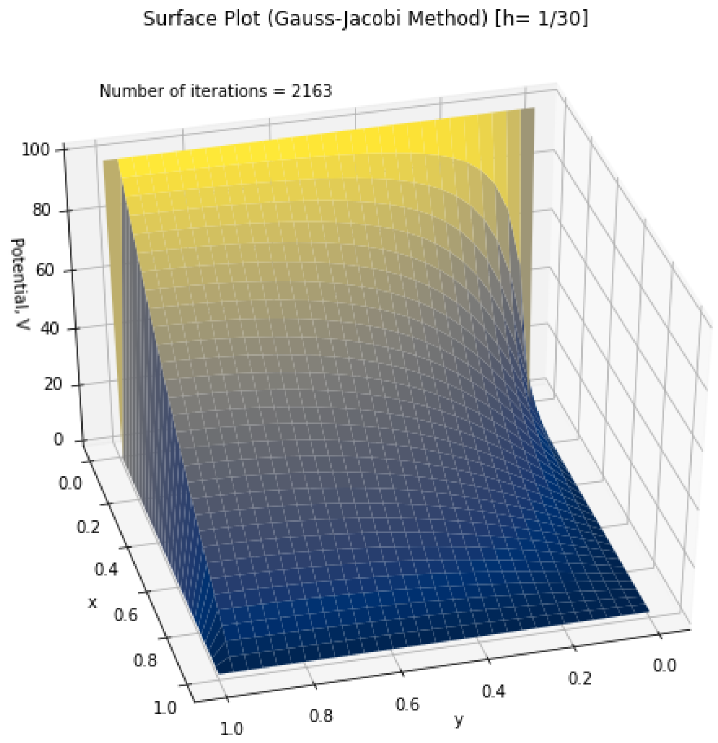
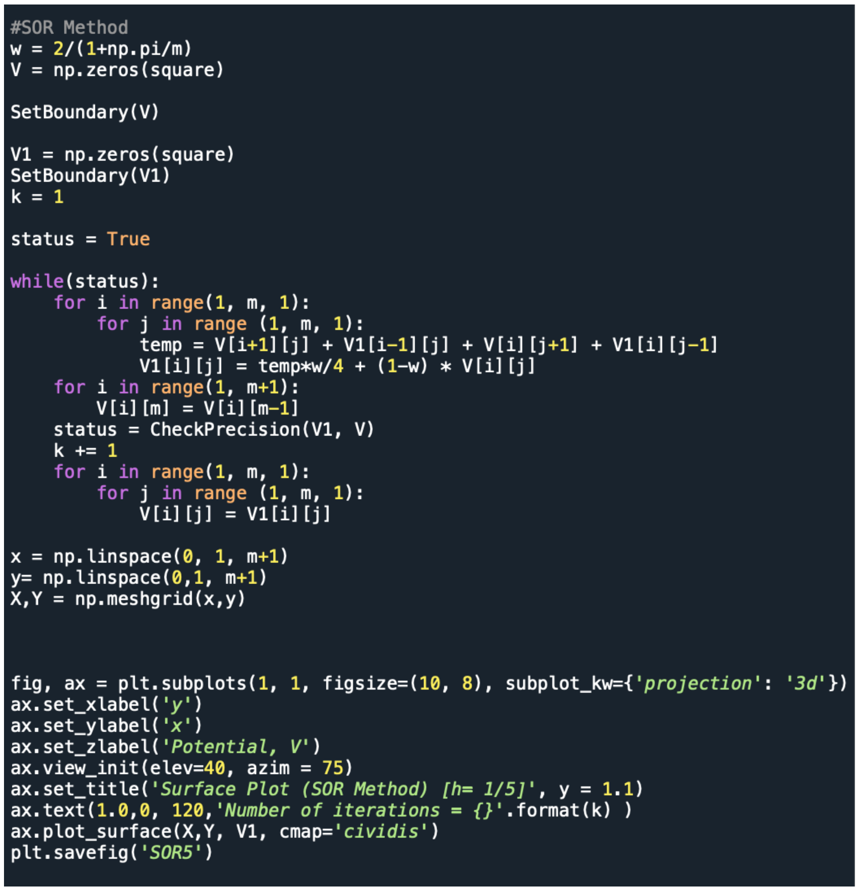


Figure 10: Variation of potential in the square using Gauss-Jacobi's Method [h=1/30]

4 Successive Over Relaxation (SOR) Method



```
#SOR Method
w = 2/(1+np.pi/m)
V = np.zeros(square)

SetBoundary(V)

V1 = np.zeros(square)
SetBoundary(V1)
k = 1

status = True

while(status):
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            temp = V[i+1][j] + V1[i-1][j] + V[i][j+1] + V1[i][j-1]
            V1[i][j] = temp*w/4 + (1-w) * V[i][j]
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]

x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (SOR Method) [h= 1/5]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('SOR5')
```

Figure 11: Code Snippet for SOR Method

The above method was employed for various grid spacing $h = 1/5, 1/10, 1/20$ and $1/30$. The results are shown below.

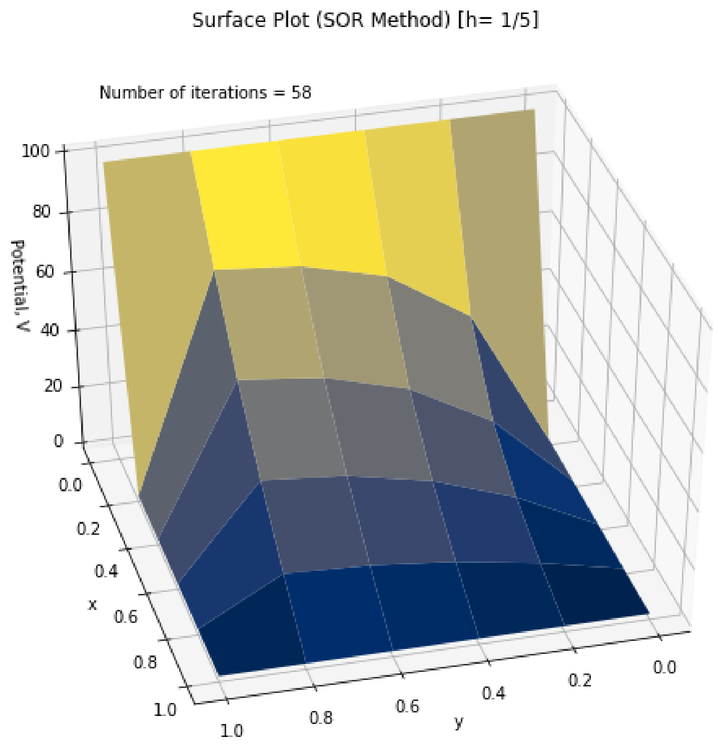


Figure 12: Variation of potential in the square using SOR Method [h=1/5]

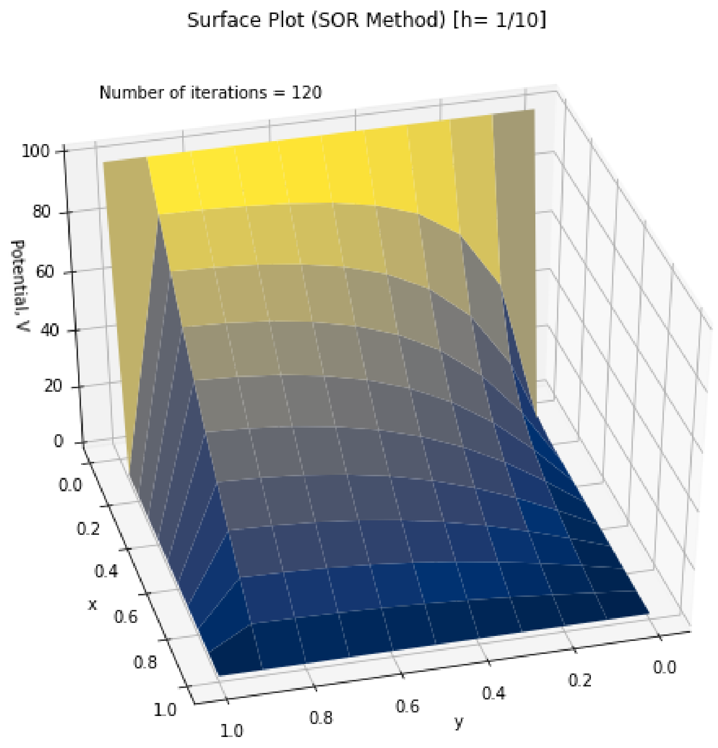


Figure 13: Variation of potential in the square using SOR Method [h=1/10]

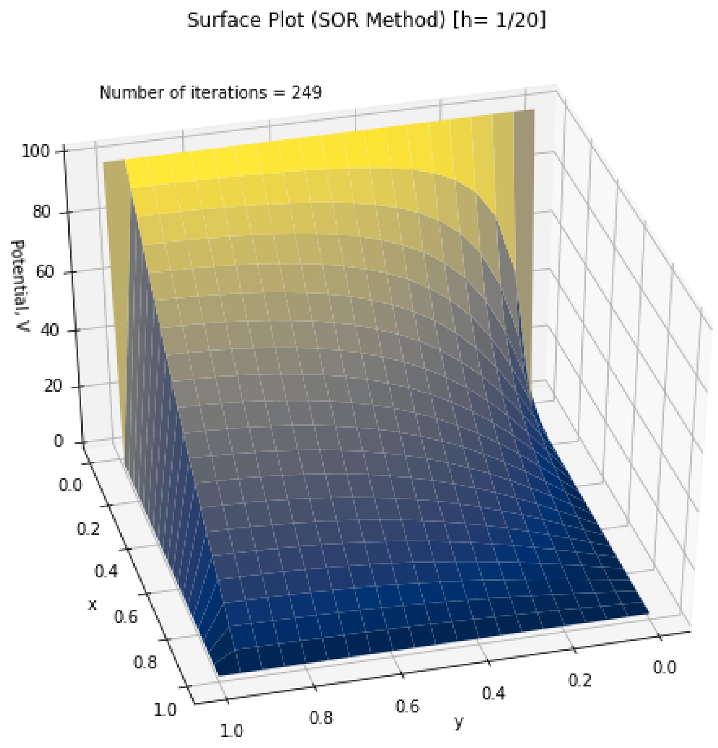


Figure 14: Variation of potential in the square using SOR Method [h=1/20]

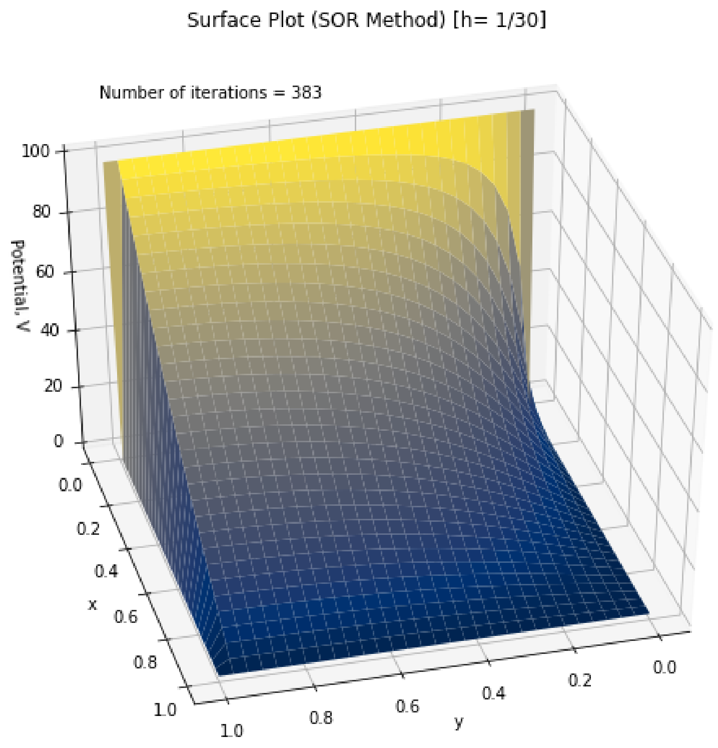


Figure 15: Variation of potential in the square using SOR Method [h=1/30]

5 Comparison and Conclusion

The following table summarises the stability of various method in terms of number of iterations:

Table 1: Comparison of Various Iterative Methods

Grid Spacing, h	Number of Iterations for Stability		
	Jacobi's Method	Gauss-Jacobi's Method	SOR Method
1/5	114	76	58
1/10	450	262	120
1/20	1782	981	249
1/30	3997	2163	383

From the above table, we can conclude that the SOR approach requires the fewest number of iterations to produce stable results irrespective of the grid spacing. Similarly, decreasing the grid spacing increases the number of iterations required for stable results. The higher grid spacing produces less accurate results. However, the general shape of the plot remains same through all grid spacing and different methods which validates relatively the accuracy of different methods. One can also observe that all the plots agree on the given set of boundary conditions.

A
Full Code

The general outline of the code looked like the following although some variables and plots were altered throughout the program to get desired graphs.

```
"""
Computational Physics
Project 3
@Lalit Chaudhary
l.chaudhary@jacobs-university.de
"""

import numpy as np
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt

h = 1/5
m = int(1/h)

square = (m+1, m+1)



def SetBoundary(V):
     V[0] = [100]*(m+1)
"""
NOTE: The other boundary cases can be ignored in this specific problem since the inital
matrix by default has all initial components as 0. Also, the iterations in various methods
specifically ignore the boundary lines and hence preserving the boundary conditions.
"""
def CheckPrecision(V1, V0):
    S = 0
    for i in range(1, m):
        for j in range(1, m):
            S += V1[i][j] - V0[i][j]
    if S < 10**(-4):
        return False
    else:
        return True


#Jacobi's Method
V = np.zeros(square)

SetBoundary(V)
```

```python
V1 = np.zeros(square)
SetBoundary(V1)

k = 1

status = True

while(status):
    for i in range(1, m):
        for j in range(1, m):
            temp = V[i+1][j] + V[i-1][j] + V[i][j+1] + V[i][j-1]
            V1[i][j] = temp/4
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]

x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (Jacobi Method) [h= 1/30]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('J30')



#Gauss-Jacobi's Method
V = np.zeros(square)

SetBoundary(V)

V1 = np.zeros(square)
SetBoundary(V1)
```

```
k = 1

status = True

while(status):
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            temp = V[i+1][j] + V1[i-1][j] + V[i][j+1] + V1[i][j-1]
            V1[i][j] = temp/4
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]


x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (Gauss-Jacobi Method) [h= 1/5]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('GJ5')


#SOR Method
w = 2/(1+np.pi/m)
V = np.zeros(square)

SetBoundary(V)

V1 = np.zeros(square)
SetBoundary(V1)
k = 1

status = True
```

```
while(status):
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            temp = V[i+1][j] + V1[i-1][j] + V[i][j+1] + V1[i][j-1]
            V1[i][j] = temp*w/4 + (1-w) * V[i][j]
    for i in range(1, m+1):
        V[i][m] = V[i][m-1]
    status = CheckPrecision(V1, V)
    k += 1
    for i in range(1, m, 1):
        for j in range (1, m, 1):
            V[i][j] = V1[i][j]

x = np.linspace(0, 1, m+1)
y= np.linspace(0,1, m+1)
X,Y = np.meshgrid(x,y)



fig, ax = plt.subplots(1, 1, figsize=(10, 8), subplot_kw={'projection': '3d'})
ax.set_xlabel('y')
ax.set_ylabel('x')
ax.set_zlabel('Potential, V')
ax.view_init(elev=40, azim = 75)
ax.set_title('Surface Plot (SOR Method) [h= 1/5]', y = 1.1)
ax.text(1.0,0, 120,'Number of iterations = {}'.format(k) )
ax.plot_surface(X,Y, V1, cmap='cividis')
plt.savefig('SOR5')
```