Assignment 5: Laplace Equation

Siddharth D P
EE18B072

March 3, 2020

1 Abstract

The goal of this assignment is the following.

- To solve for currents in a system.
- To solve 2-D Laplace equation in an iterative manner.
- To understand how to vectorize code in python.
- To plot graphs corresponding to the 2-D Laplace equation.

2 Assignment

2.1 Defining all the variables we are going to use

Importing the standard libraries

```
from pylab import *
import mpl_toolkits.mplot3d.axes3d as p3
import numpy as np
import sys
```

Initialising the appropriate variables

```
Niter=1500 #number of iterations to perform
Nx = Ny = 25 #we set it rectangular
delta = 1
Tbottom=0
Trod=1.0
Tguess=0.35
```

Creating a uniform array 25 by 25 units and creating a meshgrid of these.

```
phi = np.zeros((Ny, Nx))
y = np.linspace(-0.5,0.5,Ny)
x = np.linspace(-0.5,0.5,Nx)
Y,X = np.meshgrid(y,x)
```

Since we have a radius of 0.35 in the 1*1 grid, we first find out the points corresponding to that radius and plot them.

```
ii=where(X**2+Y**2<=0.35*0.35)
phi[ii]=1.0
figure(1)
contourf(x,y,phi,cmap=cm.coolwarm)
plt.plot(X, Y, marker=',', color='b', linestyle='none')
title('Contour of potential')
xlabel('X-axis')
ylabel('Y-axis')
colorbar()
```

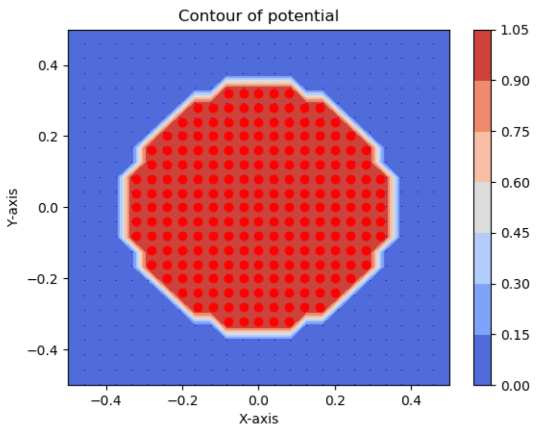


Figure 1:

2.2 Solving the Laplace Equation

We use an iterative approach to solving Laplace's equation. The iterative algorithm that we are going to use is not the most efficient one, but it is easy to implement and hence, we use it.

```
errors=np.zeros(Niter)

for k in range(Niter):
    oldphi=phi.copy()
    phi[1:-1,1:-1] = 0.25 * (phi[1:-1,0:-2] + phi[1:-1,2:] + phi[0:-2,1:-1] + phi[2
    phi[1:-1,0]=phi[1:-1,1]       #left  # Set Boundary conditions
    phi[1:-1,Nx-1]=phi[1:-1,Nx-2] #right
    phi[0,1:-1]=phi[1,1:-1]       #top
    phi[0,0]=phi[0,1]
    phi[0,Nx-1]=phi[0,Nx-2]
    phi[Ny-1:,1:-1]=Tbottom       #bottom=0
    phi[ii]=Trod                  #central portion=1V
    errors[k]=(abs(phi-oldphi)).max()
```

2.3 Plotting the error

We now plot the error in various scales such as semilog and loglog. This is done for both the entire range of iterations and for iterations above 500.

```
k=list(range(Niter))

figure(2)
grid(True)
semilogy(k[::50],errors[::50],'o-')
plt.title('Errors in y-axis Semilog versus iteration number k')
plt.xlabel('Iteration')
plt.ylabel('Error')

figure(3)
grid(True)
semilogy(k[500::50],errors[500::50],'o-')
title('Error versus iteration number above 500')
plt.xlabel('Iteration')
plt.ylabel('Error')

figure(4)
grid(True)
loglog(k[::50],errors[::50],'o-')
title('Errors in y-axis Loglog versus iteration number k')
plt.xlabel('Iteration')
plt.ylabel('Error')
```

2.4 Getting a fit for the error

We observe the error (after about 500 iterations) is approximately linear on a semilog scale We now use a least squares approach to find the values of this.

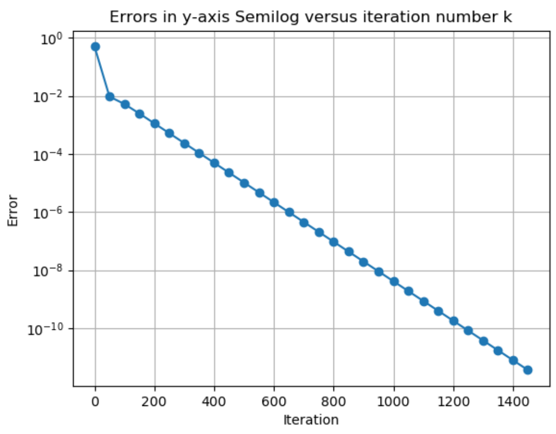


Figure 2:

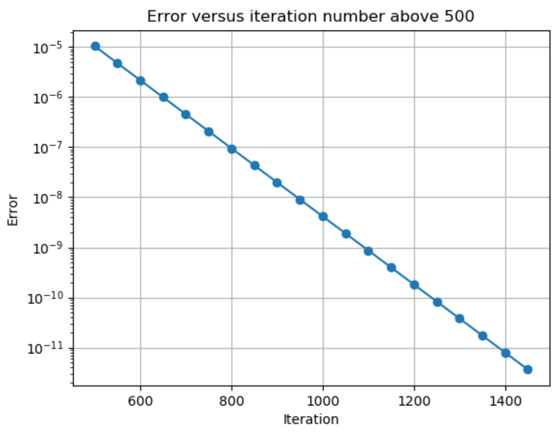


Figure 3:

```
diffexp = np.absolute(cexp-frexp)
diffcos = np.absolute(ccoscos-frcos)
print(np.amax(diffexp),np.amax(diffcos))
```

It can be seen that the difference between the two models is almost negligible for the linear region of the semilog scale. Values of A and B are 1.04177e-05 -0.01565 0.026216 -0.01566 for iterations above 500 and combined respectively. The reason for the difference in A values is that we take out a constant of e 500B out from the second expression to get the first. Since the slopes are the same, B is also the same.

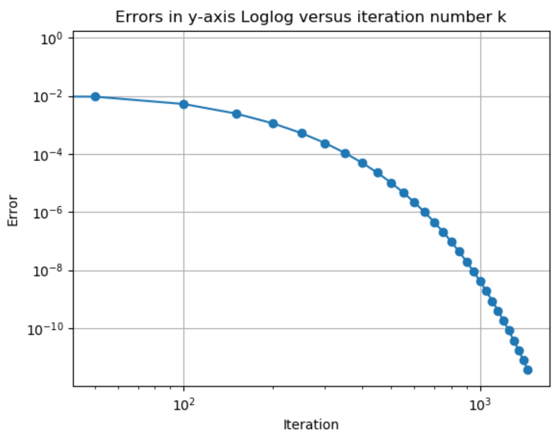


Figure 4:

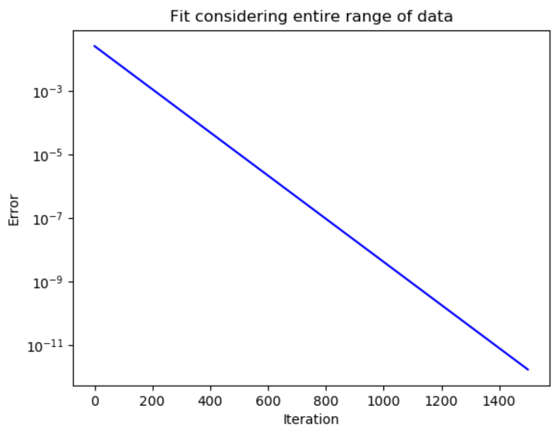


Figure 5:

2.5 Plotting the Potential

We plot the potential as a contour plot as well as on a 3D surface plot.

```
figure(8)
title('The contour plot of potential')
contourf(x,-y,phi,cmap=cm.hot)
plt.plot(ii[0]/Nx-0.48,ii[1]/Ny-0.48,'ro')
colorbar()
grid()
```

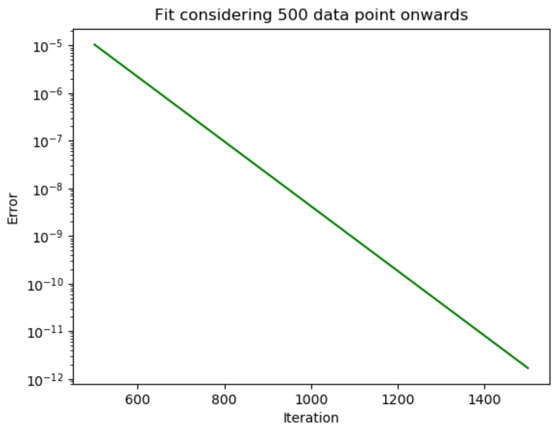


Figure 6:

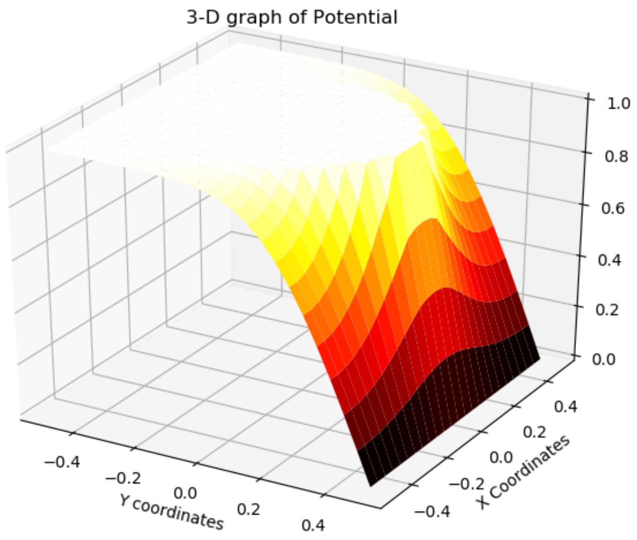


Figure 7:

2.6 Calculating the current

We can calculate the current in the region by using the negative gradient of the potential ϕ. A quiver plot of the current is made.

```
Jx = np.zeros((Ny,Nx))
Jy = np.zeros((Ny,Nx))
Jx[:,1:-1] = 0.5*(phi[:,0:-2]-phi[:,2:])
Jy[1:-1,:] = 0.5*(phi[0:-2,:]-phi[2:,:])
```

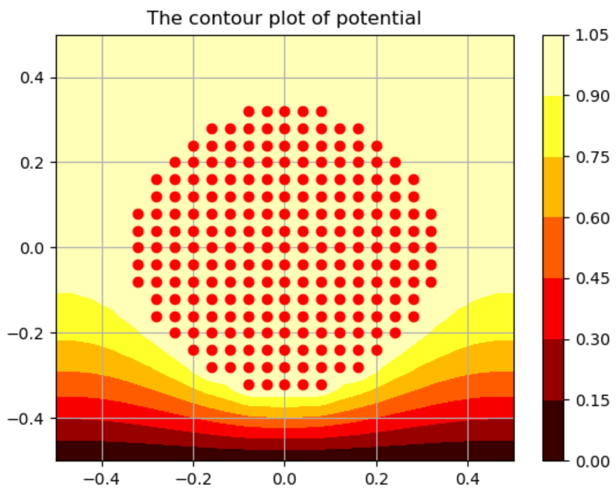


Figure 8:

```
figure(8)
plt.quiver(x,-y,Jx,-Jy,scale=5)
scatter(b1,b2,s=6,c='r')
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title("Current vectors in the plate")
```

2.7 Temperature Calculations

The temperature can be calculated in a similar way, except for the fact that there is a source of heat in the picture, and hence the final form of the expression will have to account for that as well. Hence this is solving the 2D Poisson's equation. We calculate and plot this temperature.

```
plt.ion()
phij = np.ones((Nx,Ny))*300
for i in range(Niter*2):
    phij[1:-1,1:-1]=0.25*(phij[1:-1,0:-2]+ phij[1:-1,2:]+
        phij[0:-2,1:-1]+ phij[2:,1:-1] + Jx[1:-1,1:-1]*Jx[1:-1,1:-1]+Jy[1:-1,1:-1]*
    phij[1:-1,0]=phij[1:-1,1]
    phij[1:-1,-1]=phij[1:-1,-2]
    phij[-1,:]=phij[-2,:]
    phij[0,:] = 300
    phij[ii] =300
plt.ioff()
```

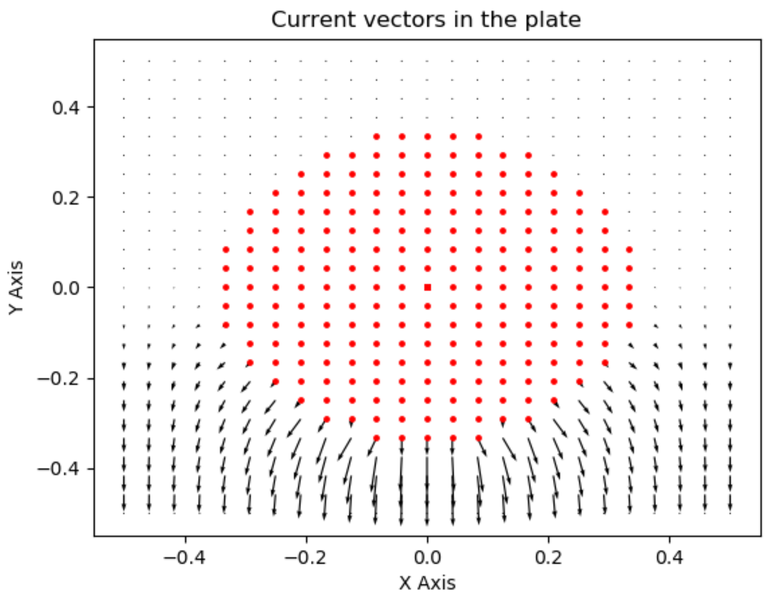
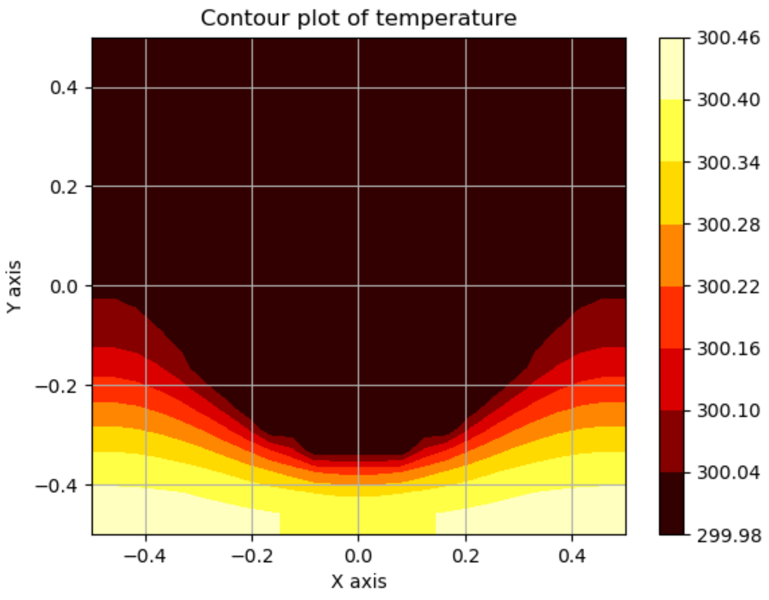


Figure 9:

```
figure(9)
grid(True)
contourf(y,x,phij,cmap=cm.hot)
plt.title('Contour plot of temperature')
plt.xlabel('X axis')
plt.ylabel('Y axis')
colorbar()
```

3 Conclusion

- Currents in a system solved using the 2-D Laplace equation are largely focused near the edge of the surface at the bottom.
- Least squares fitting can be used to approximate the error and obtain the value of the coefficients of error terms.
- Vectorized python code allows for faster, efficient computation, almost at the speed of C.
- Contour and surface plots of all the parameters helped visualize the current and temperature.



Contour plot of temperature
0.4
0.2
0.0
−0.2
−0.4
Y axis
−0.4
−0.2
0.0
0.2
0.4
X axis
300.46
300.40
300.34
300.28
300.22
300.16
300.10
300.04
299.98

Figure 10: