EE2703 Applied Programming Lab - Assignment No 5

Name: Atishay Ganesh
Roll Number: EE17B155

March 5, 2019

1 Abstract

The goal of this assignment is the following.

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

2 Assignment

2.1 Setting up the variables

Importing the standard libraries

```
import numpy as np
import mpl_toolkits.mplot3d.axes3d as p3
import matplotlib.pyplot as plt
import matplotlib
import sys
from scipy.linalg import lstsq
```

Initialising the appropriate variables

```
Nx = 25
Ny = 25
Niter = 1500
if len(sys.argv)>1:
    try:
        Ny = int(sys.argv[1])
        Nx = int(sys.argv[2])
        Niter = int(sys.argv[3])
    except:
        pass
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)
```

We define a plotting function to help simplify the code.

```
def plotter(fig_no,arg1,arg2,label_x,label_y,
type=plt.plot,arg3='b-',title="",cmap = matplotlib.cm.jet):
    plt.figure(fig_no)
    plt.grid()
    if type==plt.contourf:
        type(arg1,arg2,arg3,cmap=cmap)
    else:
        type(arg1,arg2,arg3)
    plt.xlabel(label_x,size =19)
    plt.ylabel(label_y,size =19)
    plt.title(title)
```

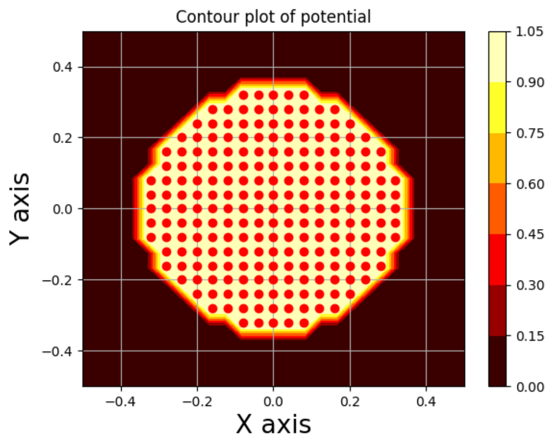
Since we have the radius as 0.35 cm of a 1*1 grid, we first find out the points in the radius and plot it.

```
ii = np.where(X*X+Y*Y<=0.35*0.35)
phi[ii]=1.0
plotter(1,x,y,"X axis","Y axis",plt.contourf,
phi,"Contour plot of potential",cmap=matplotlib.cm.hot)
plt.plot(ii[0]/Nx-0.48,ii[1]/Ny-0.48,'ro')
plt.show()
```

2.2 Solving the Laplace Equation

We use an iterative approach to solving Laplace's equation. It is not a very good approach but it is easy to implement, hence we use it.

```
errors=np.zeros(Niter)
for i 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:,1:-1]);
    phi[1:-1,0]=phi[1:-1,1]
    phi[1:-1,-1]=phi[1:-1,-2]
```



```
    phi[-1,:]=phi[-2,:]
    phi[ii]=1.0
    errors[i]=(abs(phi-oldphi)).max();
```

2.3 Plotting the error

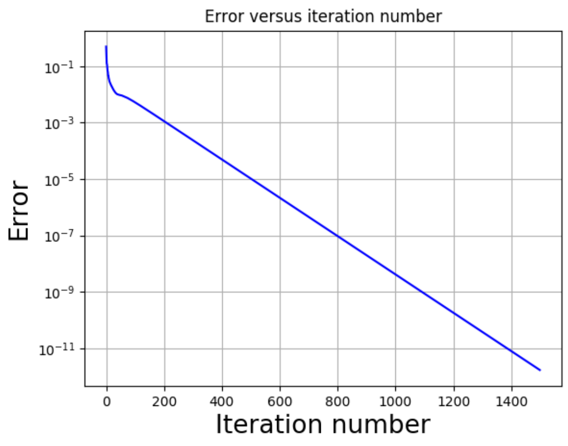
And now we plot the error in various scales.

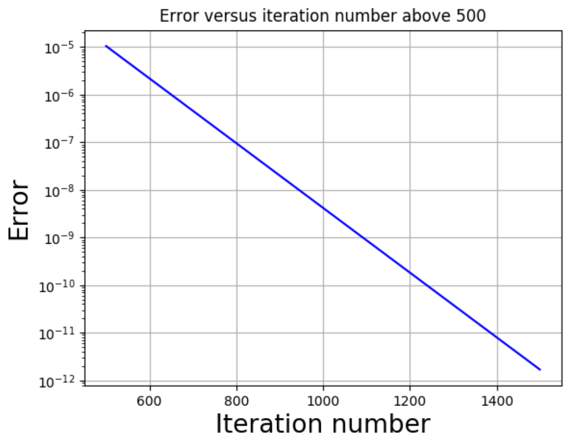
```
plotter(1,np.arange(Niter),errors,"Iteration number",
"Error",type=plt.semilogy,title='Error versus iteration number')
plotter(2,np.arange(500,Niter),errors[500:],
"Iteration number","Error",
type=plt.semilogy,title='Error versus iteration number above 500')
plotter(3,np.arange(Niter),errors,"Iteration number",
"Error",type=plt.loglog,title='Error versus iteration number loglog')
plt.show()
```

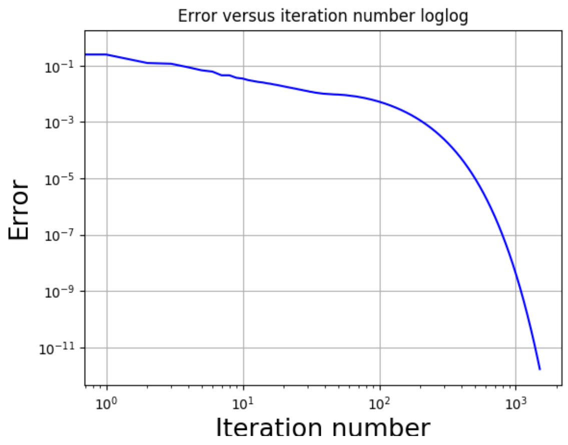
2.4 Getting a fit for the error

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.

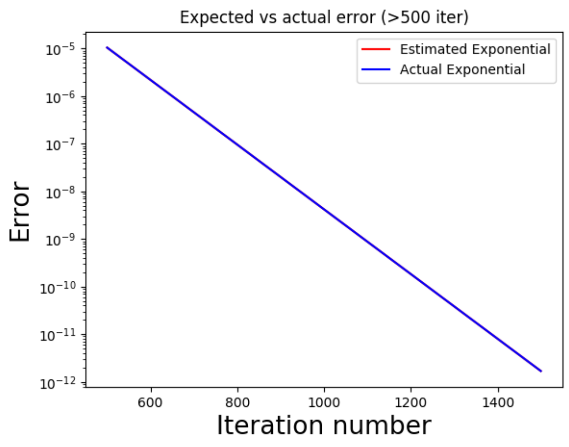
```
lst =lstsq(np.c_[np.ones(Niter-500),
np.arange(Niter-500)],np.log(errors[500:]))
a,b =np.exp(lst[0][0]),lst[0][1]
print(a,b)
plotter(1,np.arange(500,Niter),
a*np.exp(b*np.arange(Niter-500)),
"Iteration number","error",type=plt.semilogy,arg3="r-")
```

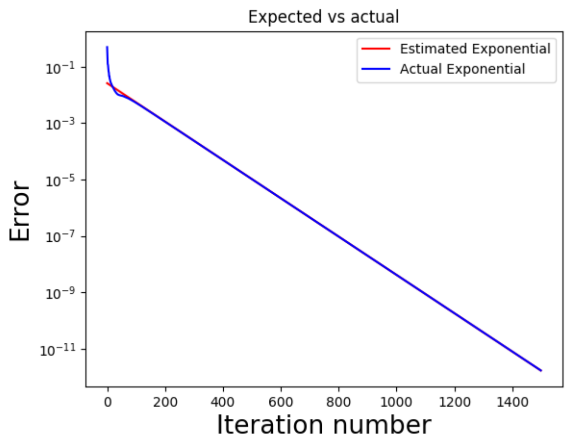






```
plotter(1,np.arange(500,Niter),errors[500:],
"Iteration number","Error",
type=plt.semilogy,title='Expected vs actual error (>500 iter)')
plt.legend(("Estimated Exponential","Actual Exponential"))
lstapprox =lstsq(np.c_[np.ones(Niter),np.arange(Niter)],np.log(errors))
a,b = np.exp(lstapprox[0][0]),lstapprox[0][1]
print(a,b)
plotter(2,np.arange(Niter),a*np.exp(b*np.arange(Niter)),
"Iteration number","Error",type=plt.semilogy,
title='Error versus iteration number',arg3 = 'r-')
plotter(2,np.arange(Niter),errors,"Iteration number",
"Error",type=plt.semilogy,title='Expected vs actual')
plt.legend(("Estimated Exponential","Actual Exponential"))
plt.show()
```



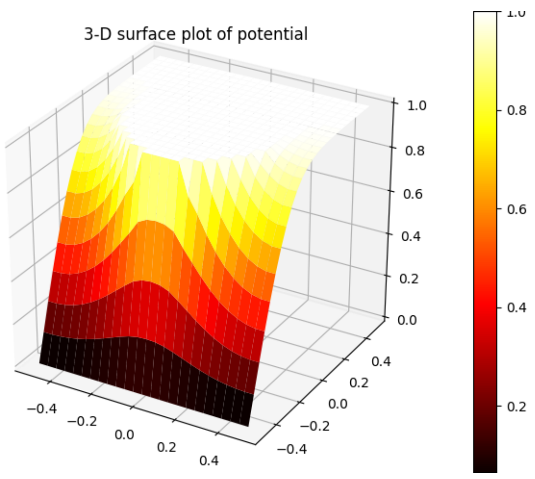


It can be seen that the difference between the two models is almost negligible for the linear region of the semilog scale. Values of A and B are 1.04177087000089392e-05 -0.015648068116154056 0.026215563815085274 -0.015655263499125868 for iterations above 500 and combined respectively. The reason for 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.

2.5 Plotting the Potential

We plot the Potential as a contour plot as well as an 3D surface plot.

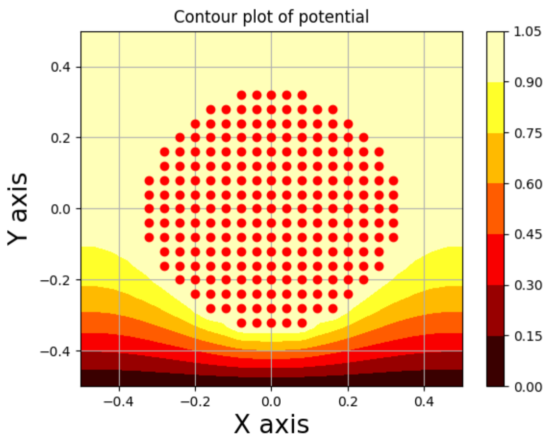
```
fig4=plt.figure(4)
ax=p3.Axes3D(fig4)
plt.title('3-D surface plot of potential')
surf = ax.plot_surface(Y, X, phi, rstride=1,
cstride=1, cmap=matplotlib.cm.hot,linewidth=0)
plt.show()
plotter(3,x,y,"X axis","Y axis",plt.contourf,phi,
"Contour plot of potential",cmap=matplotlib.cm.hot)
plt.plot(ii[0]/Nx-0.48,ii[1]/Ny-0.48,'ro')
plt.show()
```



2.6 Calculating the Current

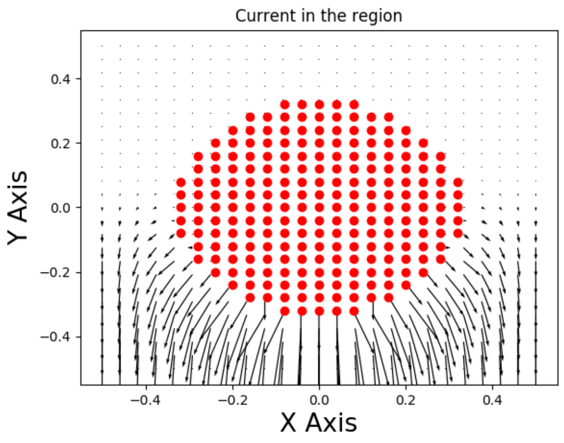
We can calculate the current in the region by using the negative gradient of the potential phi, assuming the conductivity is 1. A quiver plot of the current is made.

```
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:,:])
plt.figure(0)
plt.quiver(x,y,Jx,Jy)
plt.plot(ii[0]/Nx-0.48,ii[1]/Ny-0.48,'ro')
plt.xlabel("X Axis",size = 19)
plt.ylabel("Y Axis",size=19)
plt.title("Current in the region")
plt.show()
```

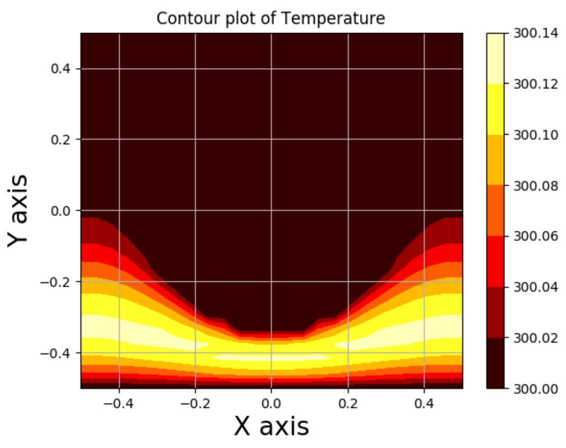
It is apparent that the current is concentrated in the bottom half between the plate and the ground. This is because most of the field is in the region between the two, and this is similar to the case of a capacitor under breakdown condition. Thus most of the current will flow in that region, and as we will see, will also be the most heated up region.



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.

```
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]*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
plotter(1,x,y,"X axis","Y axis",plt.contourf,phij,
"Contour plot of Temperature",cmap=matplotlib.cm.hot)
plt.show()
```



3 Conclusions

- We solved for currents in a system using the 2-D Laplace's equation.
- We saw how least squares fitting can be used to approximate the error.
- We vectorized python code to make it fast.
- We plotted graphs to understand the above.