Assignment 5: Applied Programming Lab

Saurabh Vaishampayan, EP17B028

3 March 2020

1 Abstract

To Solve Laplace's equations numerically over a discrete grid and obtain potential, current and temperature distributions. This code takes in arguments such as dimensions of the resistive sheet, number of iterations, number of points in the mesh through the command line and solves the Laplace's equations

2 Code for command line input

2.1 Description

We define a class to take in command line argument inputs using sys module in python. We provide initial default values to begin with which can be edited by the user. 5mm

2.2 Codes

```
import numpy as np
import matplotlib.pyplot as plt
from cmd import Cmd
import scipy.optimize
from mpl_toolkits import mplot3d

Nx = 25
Ny = 25
radius = 0.35
Niter = 1500

#cmd Prompt(shell subroutine) was borrowed from https://coderwall.com/p/
    w78iva/give-your-python-program-a-shell-with-the-cmd-module
class MyPrompt(Cmd):
    print("Type take_input Nx Ny radius Niter")
    def do_take_input(self, args):
        """Takes as input Nx, Ny,radius,Niter"""
        args = args.split(" ")

        if (len(args)!=4):
            print('Insufficient arguments')
        else:
            Nx = int(args[0])
```

```
            Ny = int(args[1])
            radius = float(args[2])
            Niter = int(args[3])
            x,y,X,Y,phi = potential_calc(Nx,Ny,radius,Niter)
            Jx,Jy = current_calc(x,y,Nx,Ny,radius,Niter,phi,X,Y)
            temperature_calc(Nx,Ny,Niter,Jx,Jy,X,Y)
            return True

    def do_quit(self, args):
        """Quits the program."""
        print ("Quitting.")
        raise SystemExit
if __name__ == '__main__':
    prompt = MyPrompt()
    prompt.prompt = '> '
    prompt.cmdloop('Starting prompt...')
```

3 Calculation of potential distribution

3.1 Description

We define a function that iteratively calculates the potential. Algorithm is as follows

1. Create a meshgrid of dimensions Ny, Nx taken as input
2. Calculate array indices where potential is 1V(Disc)
3. Initialise the potential to be 0 everywhere and 1V inside the disc. Plot contour and surface plots for the initial potential
4. Iteratively update the potential based on Laplace's equation. Add boundary conditions and calculate error as a function of iteration
5. Surface Plot final potential after Niter iterations, semilog plot of errors vs iterations. Assuming an exponential form for error, calculate the desired parameters.

3.2 Codes

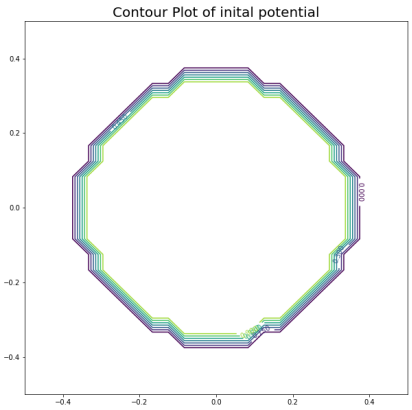
```
def linear_fit(x,a,b):
    return a*x+b

def potential_calc(Nx,Ny,radius,Niter):
    phi =np.zeros((Ny,Nx))
    x = np.linspace(-0.5,0.5,Nx)
    y = np.linspace(-0.5,0.5,Ny)

    Y,X = np.meshgrid(y,x)

    ii = np.where(X*X+Y*Y<=radius*radius)
    phi[ii] = 1.00

    plt.figure(figsize=(10,10))
    ax = plt.axes(projection='3d')
```

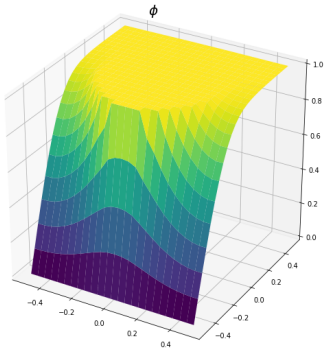
```
    ax.plot_surface(Y, X, phi, rstride=1, cstride=1,cmap='viridis',
    edgecolor='none')
    plt.title('Surface Plot of initial conditions',size=20)
    plt.savefig('intitalphi.png')
    plt.show()

    plt.figure(figsize=(10,10))
    cp = plt.contour(Y,X,phi)
    plt.clabel(cp)
    plt.title('Contour Plot of inital phi',size=20)
    plt.savefig('Contourplotinitialpotential.png')
    plt.show()

    errors = np.zeros(Niter)

    for k in range(Niter):
        oldphi = phi.copy()
        phi[1:-1,1:-1] = 0.25*(phi[0:-2,1:-1]+phi[2:,1:-1]+phi[1:-1,0:-2]+
    phi[1:-1,2:])
        phi[1:-1,0] = phi[1:-1,1]   #Left edge
        phi[1:-1,-1] = phi[1:-1,-2]    #Right edge
        phi[-1,1:-1] = phi[-2,1:-1] #Top edge
        phi[ii] = 1.0
        phi[-1,0] = phi[-2,0]
        phi[-1,-1] = phi[-2,-1]
        phi[-1,-1] = phi[-1,-2]
        errors[k] = np.abs(oldphi-phi).max()

    plt.figure(figsize=(10,10))
    plt.title(r'$\phi$',size=20)
    ax = plt.axes(projection='3d')
    ax.plot_surface(Y, X, phi, rstride=1, cstride=1,cmap='viridis',
    edgecolor='none')
    plt.show()

    plt.plot(errors)
    plt.yscale('log')
    plt.title('Semilog error plot')
    plt.xlabel('No of iterations')
    plt.ylabel('Error')
    plt.savefig('Errorssemilog.png')
    plt.show()

    n = np.arange(1,Niter+1,1)
    param, cov = scipy.optimize.curve_fit(linear_fit,n,np.log(errors))
    print('A is',np.exp(param[0]))
    print('B is',param[1])
    return x,y,X,Y,phi
```

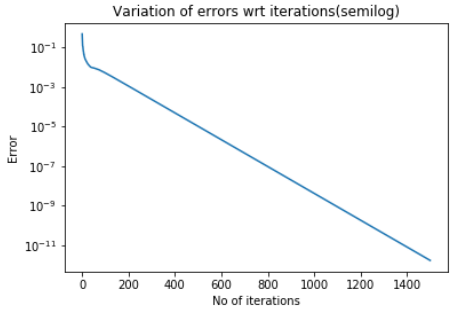
3.3 Output

A is 0.9844666459393313
B is -3.625748092895402

3.4 Plots







4 Calculation of current distribution

4.1 Description

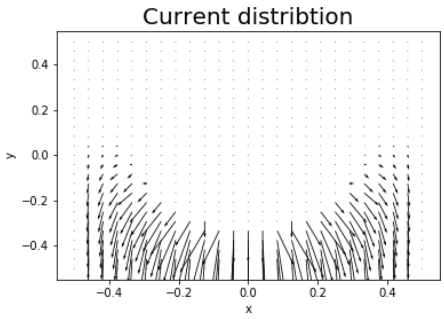
We calculate current from Ohm's law. A function is defined taking required arguments as input, and solved by finite difference method

4.2 Codes

```
def current_calc(x,y,Nx,Ny,radius,Niter,phi,X,Y):
    #Steady state currents
    Jx = np.zeros((Nx,Ny))
    Jy = np.zeros((Nx,Ny))

    Jx[1:-1,1:-1] = 0.5*(phi[1:-1,0:-2]-phi[1:-1,2:])
```

```
    Jx[0,0:] = 0
    Jx[-1,0:] = 0
    Jx[0:,0] = 0
    Jx[0:,-1] = 0

    Jy[1:-1,1:-1] = 0.5*(phi[0:-2,1:-1,]-phi[2:,1:-1,])
    Jy[0,0:] = 0
    Jy[-1,0:] = 0
    Jy[0:,0] = 0
    Jy[0:,-1] = 0

    plt.quiver(x,y,Jx,Jy)
    plt.title('Current')
    plt.show()
    return Jx,Jy
```

4.3 Plots



5 Temperature calculation

5.1 Description

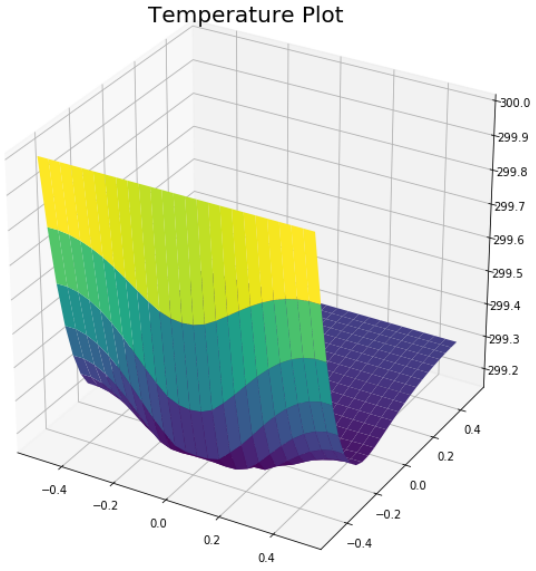
Solved in a similar way as potential calculation, as here Laplacian of temperature equals magnitude squared of current

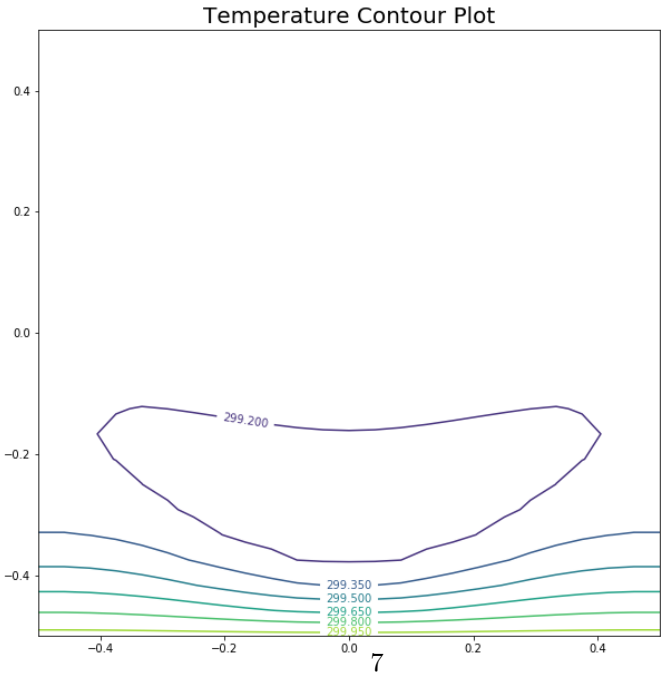
5.2 Codes

```
def temperature_calc(Nx,Ny,Niter,Jx,Jy,X,Y):
    #Solving for temperature distribution
    T = np.zeros((Ny,Nx))   #Initiallise to zero. Later add 300K to
    everything

    for k in range(Niter):
        T[1:-1,1:-1] = 0.25*(T[0:-2,1:-1]+T[2:,1:-1]+T[1:-1,0:-2]+T
    [1:-1,2:])-0.5*(Jx[1:-1,1:-1]*Jx[1:-1,1:-1]+Jy[1:-1,1:-1]*Jy
    [1:-1,1:-1])
        T[-1,1:-1] = T[-2,1:-1]
```

```
        T[1:-1,0] = T[1:-1,1]
        T[1:-1,-1] = T[1:-1,-2]
        T[-1,-1] = T[-1,-2]
        T[-1,0] = T[-1,1]
    T = T+300

    plt.figure(figsize=(10,10))
    ax = plt.axes(projection='3d')
    ax.plot_surface(Y, X, T, rstride=1, cstride=1,cmap='viridis', edgecolor
    ='none')
    plt.title('Temperature Plot',size=20)
    plt.savefig('Temperaturesurface.png')
    plt.show()

    plt.figure(figsize=(10,10))
    cp = plt.contour(Y,X,T)
    plt.clabel(cp)
    plt.title('Temperature Contour Plot',size=20)
    plt.savefig('Temperaturecontour.png')
    plt.show()
```

5.3 Plots


Temperature Plot
300.0
299.9
299.8
299.7
299.6
299.5
299.4
299.3
299.2
−0.4
−0.2
0.0
0.2
0.4


Temperature Contour Plot
299.200
299.350
299.500
299.650
299.800
299.950
0.4
0.2
0.0
−0.2
−0.4

6 Conclusions

1. This assignment gave a very nice way to explore vectorized programming in python
2. Current flow is higher in the bottom part of the plate as ground provides a sink/-source. There is no way for current to drain from the other edges, and the effect of this can also be seen in retrospect wrt potential
3. From temperature plots we can conclude that higher temperaute is found where source(s) of heating are present, ie wherever current is higher, temperature is higher too.
4. Errors follow exponential decay with a rate constant(wrt iterations) of around 0.27 Errors drop to around one in a trillion after 1500 iterations(which is moderately large iterations), signifying that the method used is not very efficient