Assignment 5: Laplace Equation

Aravint Annamalai, EE18B125

March 3, 2020

1 Introduction

In this assignment, we analyze one method of numerically solving *Laplace's Equation*:

$$\nabla^2\phi = 0$$

This equation is a special case of *Poisson's Equation*:

$$\nabla^2\phi = f$$

where $\phi(x, y)$ is a scalar function describing some potential, and $f(x, y)$ is a scalar function describing some source density. If there are no sources, Poisson's equation reduces to Laplace's equation.

In this assignment, we look at one particular instance of Poisson's equation, which arises in electrostatics. The electrostatic potential V is given by:

$$\nabla^2 V = -\frac{\rho}{\epsilon}$$

where ρ is the charge density and ϵ is the permittivity of free space. Given that there are no charges in the region of interest, it reduces to:

$$\nabla^2 V = 0$$

This equation involving contiuous partial derivatives is converted to a discrete difference equation in 2 dimensions using the central difference approximation for second derivatives. On manipulating that equation we arrive at the condition that the potential at any point is equal to the average of the potential of its neighbors.

We use this fact to iteratively approach a solution, by replacing each point in our estimate of the potential by the average of its neighbors. This iterative process approaches the true solution, albeit extremely slowly. We analyse the rate at which the error decays as well.

We then find the current density $\vec{J}$ in the region by using the following equations:

$$\vec{E} = -\nabla V$$
$$\vec{J} = \sigma \vec{E}$$

We look at a particular situation where a circular conducting electrode at a fixed potential of 1 volt is connected to a 2 dimensional square plate. One of the sides of the plates is grounded to 0 volts.

2 Initialisation of parameters

The libraries `pylab` (for plotting graphs), `mpltoolkits.mplot3d.axes3d` (for plotting 3D graphs) and `numpy` are all imported. The parameters for the size along x-axis Nx, size along y-axis Ny, radius and the number of iterations $Niter$ are declared and are given initial values. These values are then asked to be given as input by the user. These variables are then type casted to `int` for further calculations

3 Declaring and initialising the potential matrix

The matrix *phi* of order $Ny * Nx$ is declared. The coordinates along x and y axes is declared using `np.linspace()` method by which equally spaced samples between-0.5 and 0.5 are assigned to the lists x and y respectively. Using `meshgrid()` method on y and x, matrices Y and X are generated. Now, the potential array *phi* is initialised as follows:

```
ii = np.where(X*X + Y*Y <= 0.35*0.35)
phi[ii] = 1.0
```

Now, the contour plot of *phi* is plotted.

4 Perform the iteration

An array *oldphi* is created which contains the value of *phi*. The method `np.copy()` is used to do this. A list for storing the errors is declared. A for loop is written which iterates $Niter$ number of times. This updates the value of potential as described in the Introduction section. First, the value of *phi* is copied to the array *oldphi*. Then the value of each element of *phi* is taken as the average of all the values surrounding it - left, right, bottom and top. Then the boundary conditions are applied. There are four boundary conditions to be applied. The first three boundary conditions make the normal derivative of *phi* at the left, right and the top boundaries to be 0. Then the fourth boundary condition makes the value of *phi* at the bottom

boundary to be zero, as the plate is grounded there. The value of phi which was made to be 1.0 in the previous section (stored in the matrix ii) is made 0 as well after every iteration. The error after every iteration is stored as the maximum change to an element in the *phi* matrix.

5 Plotting the error

The semilog plot of error is plotted as a scatter plot showing every 50 data points first. Then we attempt to fit the linear portion and the exponential portion separately. Here, it is assumed that the error for the first 500 iterations decreases linearly whereas after that, it decays exponentially. So we plot the entire error plot first, then plot the first 500 iterations as `fit1` and te subsequent iterations as `fit 2`. All the plots here are semilog plots.

6 3-D and contour plot of potential

The 3-D part of potential is plotted using the functions in the library `mpltoolkits.mplot3d.axes3d`. The plotting is done using the code below:

```
fig1 = figure(4)
ax = p3.Axes3D(fig1)
title('The 3D surface plot of potential')
surf = ax.plot_surface(Y, X, phi.T, rstride = 1, cstride = 1, cmap = cm.jet, linewi
```

Then we plot the contour plot of potential as well, using the function `contourf`.

7 Current density calculation and plotting

Current density is proportional to the partial derivative of potential wrt position. Here, derivstive is approximated as the average of difference between neighbouring quantities. Current density matrices Jx and Jy are declared with roder $Nx * Ny$. The values are assigned as per the following code:

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

Now, a `quiver` plot is plotted for the current densities which indicates the direction in which the current is flowing. The electrode, depicted by the area where *phi* is 1.0, is also plotted to show where exactly the current is flowing wrt the electrode.

8 Conclusion

In this assignment, I have learnt how to efficiently manipulate the values of 2-D arrays in Python, minimising the use of nested `for` loops. This saves a lot of code and we will be getting benefitted from the flexibility that Python offers. Also, I have learnt about the method of computation of derivatives as the average of the neighbouring values, as it is the only way to numerically compute the derivative. Here, I have also learnt about the new method of plotting, the `quiver` plot, which a=can be used to denote anything which flows. Overall, this assignment was a good learning experience for me.