EE2703:
APPLIED PROGRAMMING LAB

Assignment 5

Laplace Equation

Arjun R
EE20B016

March 7, 2022

Aim

This assignment focuses on finding the currents in a 1cm X 1cm square resistor by solving ***Laplace's equation*** for potential.

Stating the Problem

The problem involves a 1cm X 1cm square resistor with its centre held at a potential of 1 V by a cylindrical copper wire of given radius and one of its sides grounded as shown in figure 1.

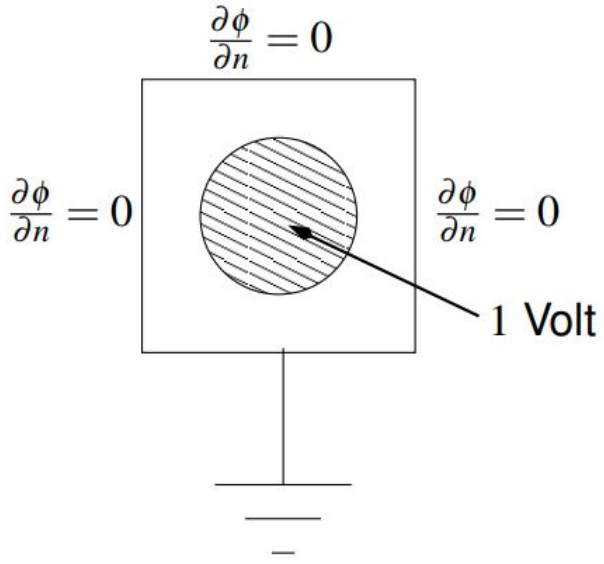


Figure 1: The circuit

Laplace's equation for potential can be derived as follows:

$$\vec{j} = \sigma \vec{E}$$

$$\vec{E} = -\nabla \phi$$

$$\nabla . \vec{j} = -\frac{\partial \rho}{\partial t}$$

$$\nabla . (-\sigma \nabla \phi) = -\frac{\partial \rho}{\partial t}$$

Assuming constant conductivity,

$$\nabla^2 \phi = \frac{1}{\sigma} \frac{\partial \rho}{\partial t}$$

For DC currents, RHS is zero, which gives us the *Laplace's equation*:

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

The resulting Laplace's equation can be transformed into a difference equation:

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4}$$

Thus, if the solution holds, the potential at any point will be as simple as the average of its neighbours.

Numerical Solution in 2 Dimensions

The default parameters that control the run are `Nx`, `Ny`, `radius`, and `Niter`.

0.1 Potential Array

0.1.1 Initialization

We can initialize a potential array of `Ny` rows and `Nx` columns with zeros. Then, update the circular wire region in the centre with potential value of 1. Figure 2 gives the contour plot of the potential array.

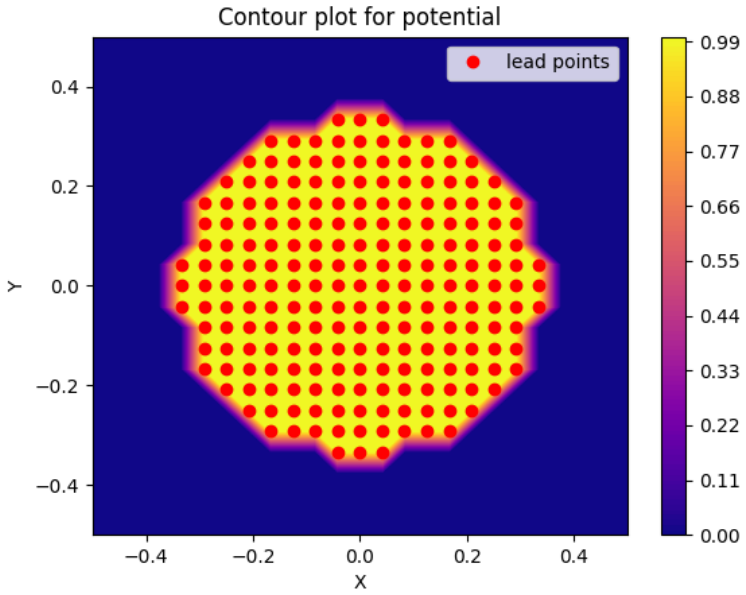


Figure 2: Potential contour plot

0.1.2 Updation

The Laplace difference equation obtained before can be used to update the array. Proper use of slicing and sub-arrays will greatly reduce the time complexity. We will be updating all the interior elements of `phi`, i.e., `phi[1:-1,1:-1]`. While performing the iterations, we will keep track of errors using `errors` vector with `Niter` elements.

0.1.3 Boundary conditions

At the bottom boundary which is grounded, the edge row is not changed and the electrode potential is static, i.e., remains zero. But at the other boundaries, we need to set the normal derivative of potential to zero, i.e., the values in the edge row/column will be same as the corresponding adjacent row/column. For example, for the left boundary, in code, this would mean

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

0.1.4 Central Electrode Potential Correction

While applying the difference equation, the values in the region of the potential array held at 1 V by the circular wire is changed. This has to be corrected back to 1 V.

The code implementation of the updation, the boundary condition, and the central potential correction is as follows:

```
def update_phi(phi):
    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[:,0] = phi[:,1] # left boundary
    phi[:,-1] = phi[:,-2] # right boundary
    phi[0,:] = phi[1,:] # top boundary
    # bottom boundary is not updated at all, hence stays zero

errors = zeros(Niter)
for k in range(Niter):
    oldphi=phi.copy()
    update_phi(phi)
    phi[ii] = 1.0 # over-written values at the electrodes

    errors[k] = (abs(phi-oldphi)).max()
```

0.2 Graphing the results

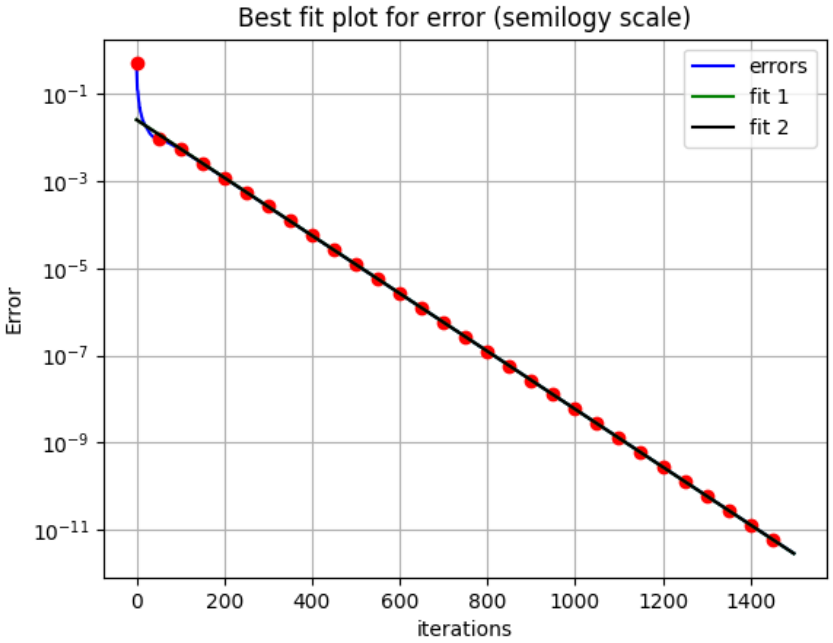
0.2.1 Exponential Fit

Figure 3 gives the plot of maximum error in the potential array values in each iteration and shows how the errors are evolving. A semi-log plot gives a decreasing straight line after some point implying that the errors are, in fact, exponentially decreasing. We can try to fit this data to an exponential curve. Two fit plots are obtained, one for all error values, and one for larger iteration numbers, say, above 500. The decay is exponential only for larger iteration numbers although there is not much difference between the fits obtained. Code implementation:

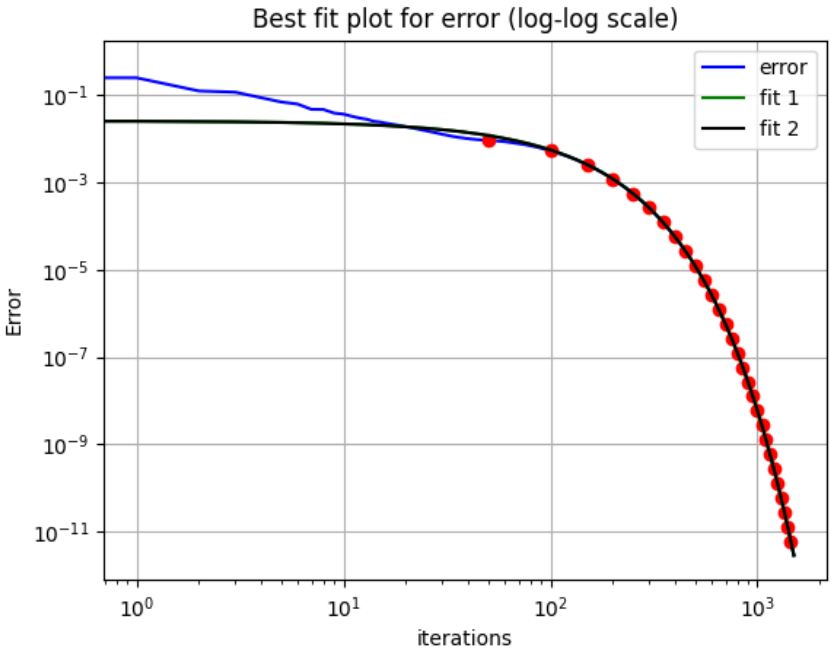
```
M = np.zeros((Niter, 2))
M[:, 0] = 1
p = np.zeros(Niter)
for i in range(Niter):
    M[i, 1] = i
    p[i] = log(errors[i])

x1 = lstsq(M, p)[0] # for whole errors list
x2 = lstsq(M[500:,:], p[500:])[0] # after 500 iterations
```

0.2.2 Cumulative Sum of Errors

Cumulative sum of errors can be calculated as:



(a) Semi-log scale



(b) Log-log scale

Figure 3: Error plot and exponential fit

$$
\begin{aligned}
Error &= \sum_{k=N+1}^{\infty} error_k \\
&< \sum_{k=N+1}^{\infty} Ae^{Bk} \\
&\approx \int_{N+0.5}^{\infty} Ae^{Bk} dk \\
&= -\frac{A}{B} exp(B(N+0.5))
\end{aligned}
$$

Code implementation:

```
def get_cum_error(A, B, N):
    return -(A/B)*exp(B*(N+0.5))

cum_error = get_cum_error(exp(x2[0]), x2[1], np.arange(Niter))

loglog(np.arange(100, Niter, 100), cum_error[100::100], 'ro')
xlabel('iterations')
ylabel('Maximum cumulative error')
grid()
title('Log-log plot for cumulative error')
show()
```

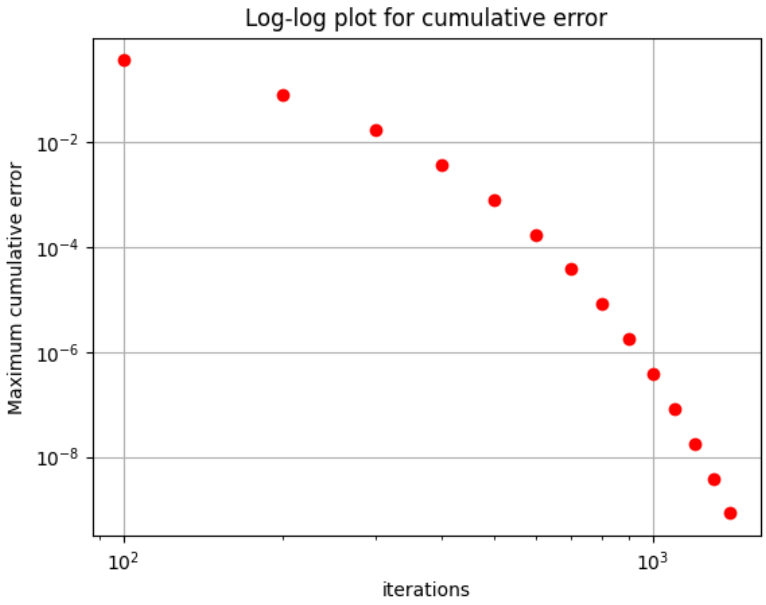


Figure 4: Cumulative error on log-log scale

0.2.3 3D Surface Plot of Potential

The function `Axes3D` is required to plot 3D plots. The function `plot_surface` gives the surface plot.

```
fig5 = figure(5) # open a new figure
ax = p3.Axes3D(fig5, auto_add_to_figure=False)
# Axes3D is the means to do a surface plot
fig5.add_axes(ax)
title('The 3-D surface plot of the potential')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Potential')
surf = ax.plot_surface(X, Y[::-1], phi, rstride=1, cstride=1, cmap=cm.jet)
show()
```

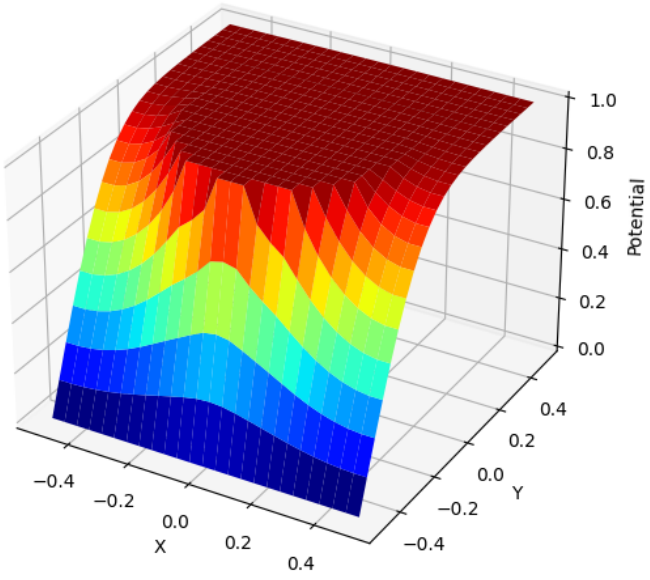


Figure 5: 3D surface plot

0.2.4 Contour Plot of Potential

The `contourf` function was used to obtain a contour plot of the final potential array. Figure 6 shows this.

0.2.5 Vector Plot of Currents

We have

$$\vec{j} = \sigma \vec{E}$$
$$\vec{E} = -\nabla\phi$$

Setting $\sigma = 1$ gives:

$$\vec{j} = -\nabla\phi$$

This can be numerically translated to the difference equation:

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1})$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j})$$

The `quiver` function can be used to obtain the vector plot.

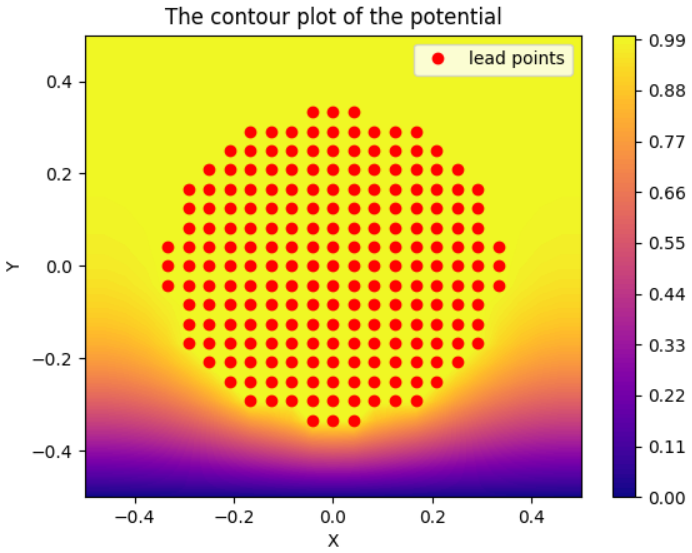


Figure 6: Contour plot of final potential values

```
Jx = 0.5*(phi[1:-1,0:-2]-phi[1:-1,2:]) # left - right
Jy = 0.5*(phi[0:-2,1:-1]-phi[2:,1:-1]) # top - bottom

fig5, ax = plt.subplots()
quiver(X[1:-1,1:-1], Y[-1:1:-1,-1:1:-1], Jx, -Jy, label = r'$\vec{J}$')
plot((x_lead-(Nx-1)/2)/(Nx-1), (y_lead-(Ny-1)/2)/(Ny-1),
'ro', label = 'Lead points')
xlabel('X')
ylabel('Y')
legend()
title('The vector plot of the current flow')
show()
```

Note that in the code, while plotting, `Y` sub-array is retrieved in the reverse order. This is due to differences between the array indexing and plotting axis dimensions. Due to this, `Jy` array is also taken with a negative sign.

From figure 7, it is clear that the current fills the entire cross-section and then flows in the direction of the grounded electrode.

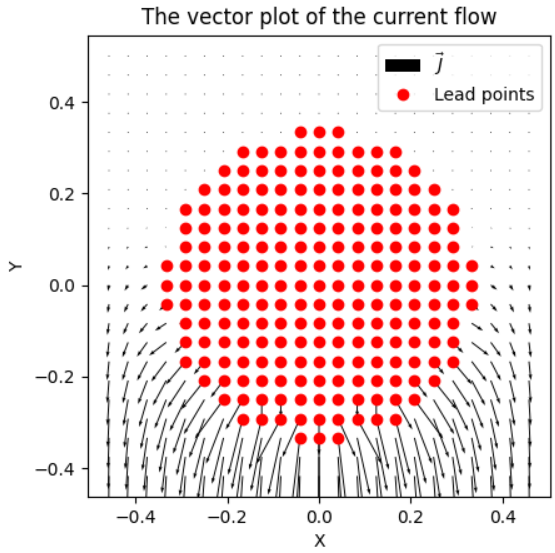


Figure 7: Vector plot of current flow

Conclusion

We have solved the resistor problem by solving *Laplace equations* using finite-difference method. This algorithm is cited to be one of the worst available because of the very slow coefficient with which the error reduces. We have seen that the maximum error reduces exponentially. The 3D surface and contour plots better helps to build an understanding of how the potential varies and how its gradient is proportion to the current flow.