EE2703: Assignment 5

Sagar (ee20b115)

March 3, 2022

1 Aim

In this assignment, we will solve for the currents in a resistor and discusses which part of the resistor is likely to get hottest.

Here we analyse the currents in a square copper plate to which a wire is soldered to the middle of it having 1V potential.

Then we will discuss how to find stopping condition after certain iterations,and model the errors obtained using Least Squares after analysing the actual errors.

And finally we will find the currents in the resistor after applying boundary conditions and analyse the vector plot of current flow and conclude which part of resistor will become hot.

2 Equations

A wire is soldered to the middle of a copper plate and its voltage is held at 1 Volt. One side of the plate is rounded, while the remaining are not.

The dimension of the plate can be chose by the user of program. In this report we have used a 25 units by 25 units plate. And the no. of iterations will be 1500 with the radius of 1V regions as 8 units. Also the error tolerance is chosen to be $1e-7$.

To solve for currents in resistor,we use following equations given below:

1. Differential form of ohm's law
$$\vec{J} = \sigma\vec{E} \tag{1}$$

2. Electric field is the gradient of the potential
$$\vec{E} = -\nabla\phi \tag{2}$$

3. Charge Continuity equation
$$\nabla.\vec{J} = -\frac{\partial\rho}{\partial t} \tag{3}$$

4. Combining the above equations above, we get
$$\nabla.(-\sigma\nabla\phi) = -\frac{\partial\rho}{\partial t} \tag{4}$$

5. For a resistor of constant conductivity, the equation becomes
$$\nabla^2\phi = \frac{1}{\sigma}\frac{\partial\rho}{\partial t} \tag{5}$$

6. For DC current, we have
$$\nabla^2\phi = 0 \tag{6}$$

3 Procedure and Observations

3.1 Solution for 2D plate

First we have to solve this equation to obtain Potential as ϕ

$$\nabla^2\phi = 0 \tag{7}$$

For a 2D surface this reduces to
$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{8}$$

$$\frac{\partial \phi}{\partial x}_{(x_i,y_j)} = \frac{\phi(x_{i+1/2}, y_j) - \phi(x_{i-1/2}, y_j)}{\Delta x} \tag{9}$$

$$\frac{\partial^2 \phi}{\partial x^2}_{(x_i,y_j)} = \frac{\phi(x_{i+1}, y_j) - 2\phi(x_i, y_j) + \phi(x_{i-1}, y_j)}{(\Delta x)^2} \tag{10}$$

For points on a 2D cartesian plane, this can be turned into a difference equation

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{11}$$

For the points on the boundary we have

$$\frac{\partial \phi}{\partial n} = 0 \tag{12}$$

Also ϕ for lower side of the square is 0.

The Error after k^{th} iteration, E_k can be written as

$$E_k = \phi_{k+1} - \phi_k \tag{13}$$

3.2 Error

3.2.1 CODE:

```
x = array(linspace(int(-Nx/2), int(Nx/2), Nx))
y = array(linspace(int(Ny/2), int(-Ny/2), Ny))

phi = zeros([Nx,Ny])
[Y,X] = meshgrid(x,y)

Z = where((X*X + Y*Y) <= (radius)**2)
phi[Z] = 1.0
phi1 = phi.copy()

Err = 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:,1:-1])
phi[0,1:-1] = phi[1,1:-1]                                    phi[1:-1,0] =  phi[1:-1,1]
phi[1:-1,-1] = phi[1:-1,-2]
phi[-1,1:-1] = 0
phi[Z] = 1.0
Err[k] = (np.max(absolute(phi-oldphi)))
def func_fit(S,c):
A = zeros([S.size, 2])
A[:,0] = 1
A[:,1] = S
b = lstsq(A, log(c), rcond = None)[0]
fit = exp(dot(A,b))
return b, fit

Err2 = []
for i in range(int(Niter/50)):
Err2.append(Err[50*i])

figure(0)
title("semilog plot of error after "+str(Niter)+" iterations")
L1 = linspace(1,len(Err2),len(Err2))
semilogy(50*L1,Err2,'b.-')
xlabel("iterations")
ylabel("error")
legend(["error"], loc = 'upper right')
```

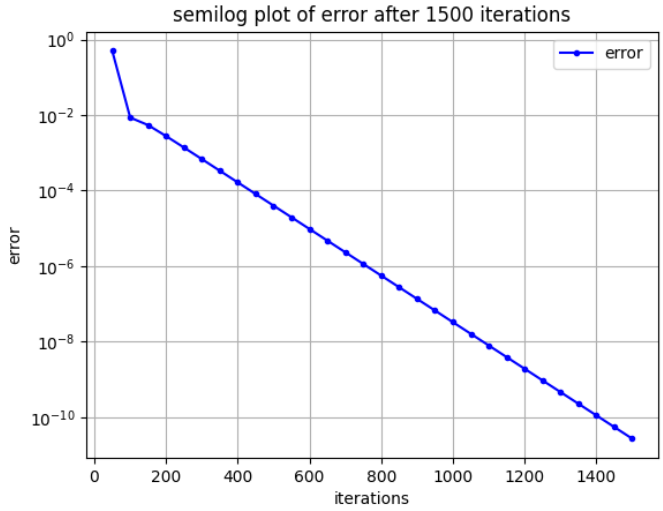
```
grid(True)
savefig("figure_0")
show()

figure(1)
title("loglog plot of error after "+str(Niter)+" iterations")
loglog(50*L1,Err2,'b.-')
xlabel("iterations")
ylabel("error")
legend(["error"], loc = 'upper right')
grid(True)
savefig("figure_1")
show()
```

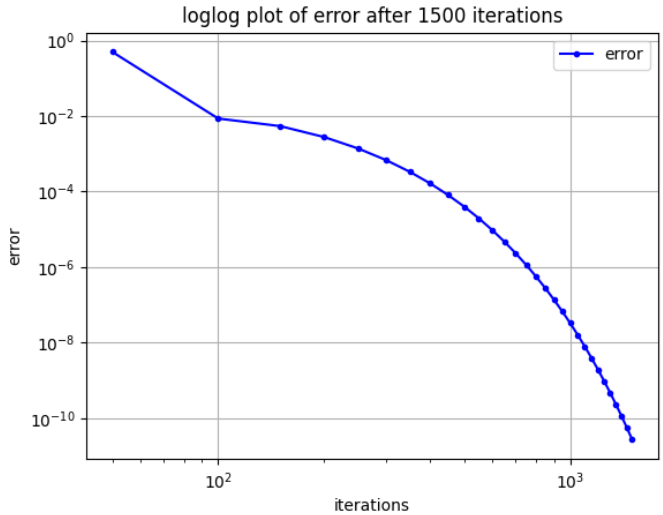
3.2.2 PLOTS:

We plot the error vs the no. of iterations to see the how the error is evolving in this problem.

First, we'll use a *semilog* graph for the plot:



Now, we'll use a *loglog* graph for the plot:



3.2.3 OBSERVATIONS:

Here we have plotted every 50th iteration for readability. From the plots we observe that:

1. The *loglog* plot gives a reasonably straight line upto about 100 iterations, but after that we get into a exponential curve. This is for a 25 by 25 grid.
2. The *semilog* plot gives a straight line for the plot after 100 iterations, before that it is hard to make any conclusion since we have only plotted 2 points.

3.3 Modelling Error

Since the *semilog* plot is a straight line, we can model the given error curve using the exponential function.

Let the error be:

$$y = Ae^{Bx}$$

which can be re written as:

$$logy = logA + Bx$$

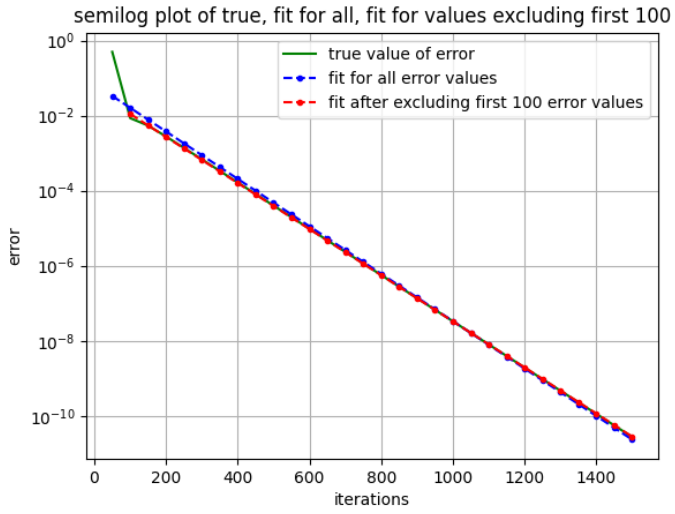
3.3.1 CODE:

```
Err3 = np.delete(Err2,[0])
L2 = np.delete(L1,[0])

figure(2)
title("semilog plot of true, fit for all, fit for values excluding first 100")
semilogy(50*L1,Err2,'g')
b1, fit1 = func_fit(L1,Err2)
b2, fit2 = func_fit(L2,Err3)
semilogy(50*L1, fit1, 'b.--')
semilogy(50*L2, fit2, 'r.--')
legend(["true value of error", "fit for all error values", "fit after excluding first 100 erro
xlabel("iterations")
ylabel("error")
grid(True)
savefig("figure_2")
show()
```

3.3.2 PLOTS:

We extract A and B for the whole 1500 iterations and then after removing the first 100 iterations and plot the curve $y = Ae^{Bx}$ for both cases on a *semilog* graph.



3.3.3 OBSERVATIONS:

1. We observe in the beginning the fit for all values(**blue**) is slightly higher than the true value(**green**) and fit after removing first 100 values(**red**). But later it goes slightly lower towards the end.
2. The fit after removing first 100 iterations follows the true value very accurately and doesn't seem to deviate even towards the end of iterations.

3.4 Stopping Condition

The maximum error scales as

$$Error = Ae^{Bk}$$

where k is the iteration number.

Summing up the terms, we have

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

For the given data (fit for all errors), we have

$$A = 0.024044$$
$$B = -0.014195$$
$$Tolerance = 1e-7$$

which gives

$$N = 1173$$
$$Error = 9.869772e-8$$

3.4.1 CODE:

```
x = linspace(1,Niter,Niter)
b3, fit3 = func_fit(x, Err)

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

def stopCon(Niter, tol):
cumErr = []
for n in range(1, Niter):
    cumErr.append(cumError(n, exp(b3[0]), b3[1]))
    if(absolute(cumErr[n-1]) <= tol):
        return cumErr[-1], n
return cumErr[-1], Niter

finErr, nStop = stopCon(Niter, tol)
print("After stopping, we have %g iterations and error is %g" %(nStop, finErr))
```

3.4.2 OBSERVATIONS:

1. We observe that the error changes very little every iteration, but changes continuously. So the cumulative error is still large.
2. This method of solving Laplace's Equation is very slow. This is because of the very low coefficient with which the error reduces.

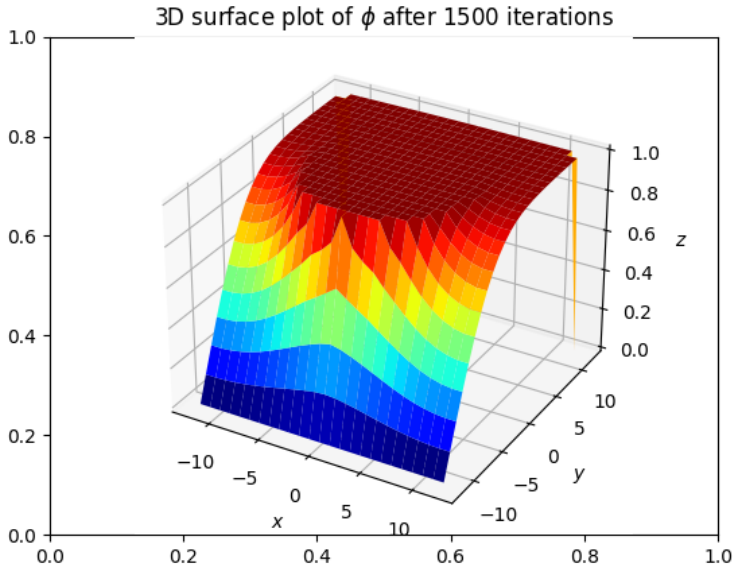
3.5 Surface Plot of Potential

3.5.1 CODE:

```
figure(3)
title("3D surface plot of $\phi$ after "+str(Niter)+" iterations")
ax = axes(projection ='3d')
ax.plot_surface(Y, X, phi, cmap ='jet')
ax.set_xlabel('$x$')
ax.set_ylabel('$y$')
ax.set_zlabel('$z$')
savefig("figure_3")
show()
```

3.5.2 PLOT:

Next we plot the final potential after 1500 iterations on a 3D surface plot.



3.5.3 OBSERVATIONS:

1. We see that the Potential is higher(**red**) towards 1 and it slowly becomes lower(**blue**) as we move towards the opposite end.
2. The maximum values at **red** end is 1V while the minimum value at **blue** end is 0V.

3.6 Contour Plot of Potential

3.6.1 CODE:

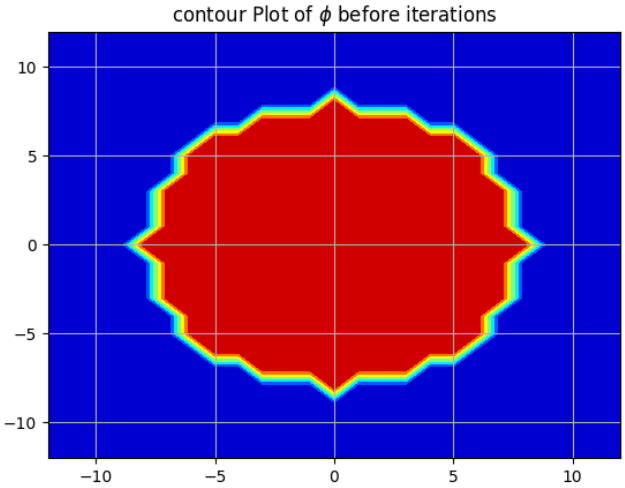
```
figure(4)
title("contour Plot of $\phi$ before iterations")
contourf(Y, X, phi1, cmap = cm.jet)
grid(True)
savefig("figure_4")
show()

figure(5)
title("contour Plot of $\phi$ after "+str(Niter)+" iterations")
contourf(Y, X, phi, cmap = cm.jet)
grid(True)
savefig("figure_5")
```

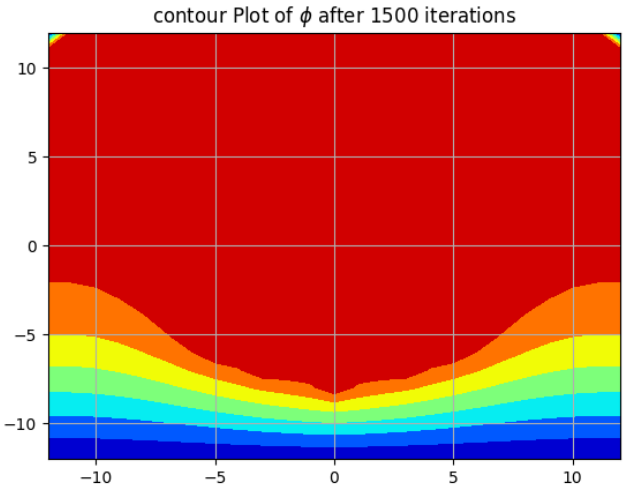
show ( )

3.6.2 PLOTS:

First we plot the initial potential before the iterations on a contour plot.



Next we plot the final potential after 1500 iterations on a contour plot.



3.6.3 OBSERVATIONS:

1. We observe that before iterations the approximately circular area with radius of 8 units is marked **red**.
2. After the iterations, the **red** part spread out and the top part becomes **red** while the bottom remains **blue** with a gradual trend of **red** to **green** to **blue** in bottom half.

3.7 Vector Plot of Currents

The Current Density for a given Potential is obtained by

$$J_x = -\frac{\partial \phi}{\partial x} \tag{14}$$

$$J_y = -\frac{\partial \phi}{\partial y} \tag{15}$$

For points on a 2D plane, they can be written as following difference equations

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{16}$$

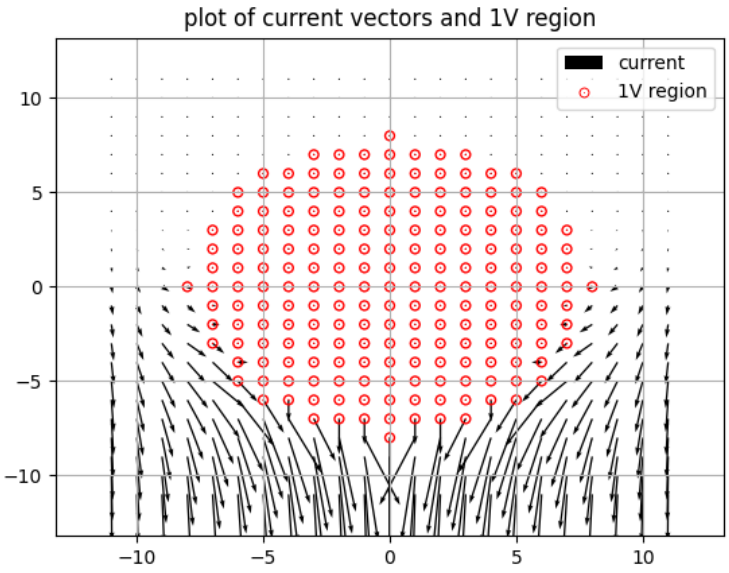
$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{17}$$

3.7.1 CODE:

```
figure(6)
title("plot of current vectors and 1V region")
Jy = (phi[2:,1:-1] - phi[0:-2,1:-1])
Jx = (phi[1:-1,0:-2] - phi[1:-1,2:])
quiver( Y[1:-1,1:-1], X[1:-1,1:-1], Jx[:,::-1], Jy[:,::-1])
scatter(Y, X, phi1, color = 'Red', linewidths=5)
legend(["current", "1V region"])
grid(True)
savefig("figure_6")
show()
```

3.7.2 PLOT:

Using these, we plot the vector plot of currents after 1500 iterations.



3.7.3 OBSERVATIONS:

1. We see that the current vectors are going from middle to bottom in lower half while they are essentially zero in the upper half.
2. The 1V potential points are marked in **red**. There is no current in the central region as all the points are at same potential.

4 Conclusions

1. We know that heat generated is from $\vec{J}.\vec{E}$ so since $\vec{J}$ and $\vec{E}$ are higher in the bottom region of the plate, there will more heat generation.
2. Since there is almost no current in the upper region of plate,the upper part will remain colder relative to the lower region ignoring changes in temperature distribution due to conduction.

3. We observe that the best method to solve this is to increase N_x and N_y and increase the no of iterations, so that we get accurate values of potential and current.

4. But the tradeoff with increasing N_x and N_y and the no of iterations is speed. For higher values of parameters, the calculation times is larger.