Assignment 5: The Resistor Problems

Chollangi Dheeraj Sai EE20B029

March 7, 2022

Overview

We wish to solve for the currents in a resistor. The currents depend on the shape of the resistor and we also want to know which part of the resistor is likely to get hottest.

1 The poisson's equation

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 grounded, while the remaining are floating. The plate is 1 cm by 1 cm in size
Using the continuity equation

$$\vec{J} = \sigma\vec{E} \tag{1}$$

From ohm's law

$$\vec{E} = -\nabla\phi \tag{2}$$

From Maxwell's equation

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

Using the above equations we obtain

$$\nabla.(-\sigma\nabla\phi) = -\frac{\partial\rho}{\partial t} \tag{4}$$

Assuming that our resistor contains a material of constant

$$\nabla^2\phi = \frac{1}{\sigma}\frac{\partial\rho}{\partial t} \tag{5}$$

For DC currents, the right side is zero, and we obtain

$$\nabla^2\phi = 0 \tag{6}$$

2 Taking input from user

To start solving for potential and other quantities, we need to initialize parameters for the resister. These parameters are ontained from command line arguments using the below code.

```
print(f"Usage: python {sys.argv[0]} <Nx> <Ny> <radius> <Niter>")

# Taking input
if len(sys.argv) == 5:
    Nx = int(sys.argv[1])
    Ny = int(sys.argv[2])
    radius = int(sys.argv[3])
    Niter = int(sys.argv[4])
else:
    Nx = 25
    Ny = 25
    radius = 8
    Niter = 1500
```

3 Initializing potential

We will create a 2-D matrix using the parameters from the user and visualise the contour plot of the potential. We will use the below code to do the same.

```
figure()
contourf(X, Y, phi)
colorbar()
plot((ind[0] + 0.5 - Nx / 2) / Nx, (ind[1] + 0.5 - Ny / 2) / Ny, "ro",label="V =
title("Potential Configuration", fontsize=16)
xlabel("x \u2192", fontsize=12)
ylabel("y \u2192", fontsize=12)
legend()
```

The red dots indicate that the potential is unity.

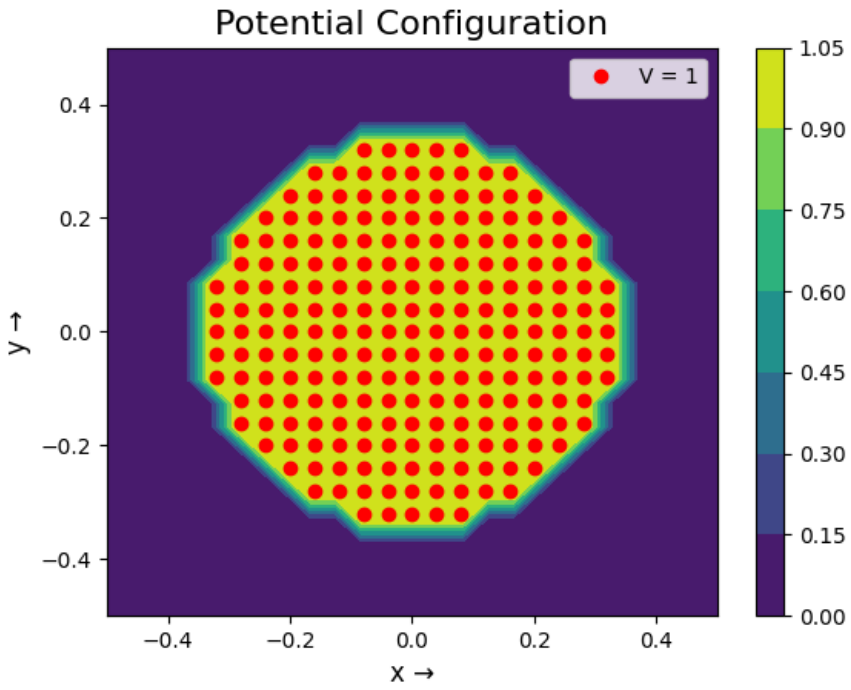

Figure 1: Contour plot of potential

4 Updating potential

Using the poisson's equation discreetly to the points in the matrix, we will arrive at the below equation.

$$\phi_{i,j} = \frac{\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1}}{4} \tag{7}$$

The python function for doing the same is given by

```
def update_phi(phi, phiold):
    phi[1:-1, 1:-1] = 0.25 * (
        phiold[1:-1, 0:-2] + phiold[1:-1, 2:] + phiold[0:-2, 1:-1] + phiold[2:,
    )
    return phi
```

5 Using boundary conditions

The bottom surface of the resistor is grounded, it's potential is taken as 0. The other three surfaces are left floating, since the current can't travel normally along these three surfaces, gradient of the potential can be said to

be in tangential direction.
The below function is used to apply the boundary conditions

```
def update_boundary(phi, ind):
    phi[:, 0] = phi[:, 1]  #             Left Boundary
    phi[:, Nx - 1] = phi[:, Nx - 2]  #   Right Boundary
    phi[0, :] = phi[1, :]  #             Top Boundary
    phi[Ny - 1, :] = 0  #                Bottom Boundary
    phi[ind] = 1.0
    return phi
```

6 Calculating error

For each iteration we calculate the error involved and store it in the array err.It can done by the following code

```
err = np.zeros(Niter)
for k in range(Niter):
    phiold = phi.copy()
    phi = update_phi(phi, phiold)
    phi = update_boundary(phi, ind)
    err[k] = (abs(phi - phiold)).max()
```

7 Error plots

We will now plot the error calculated before in loglog and semilog plots

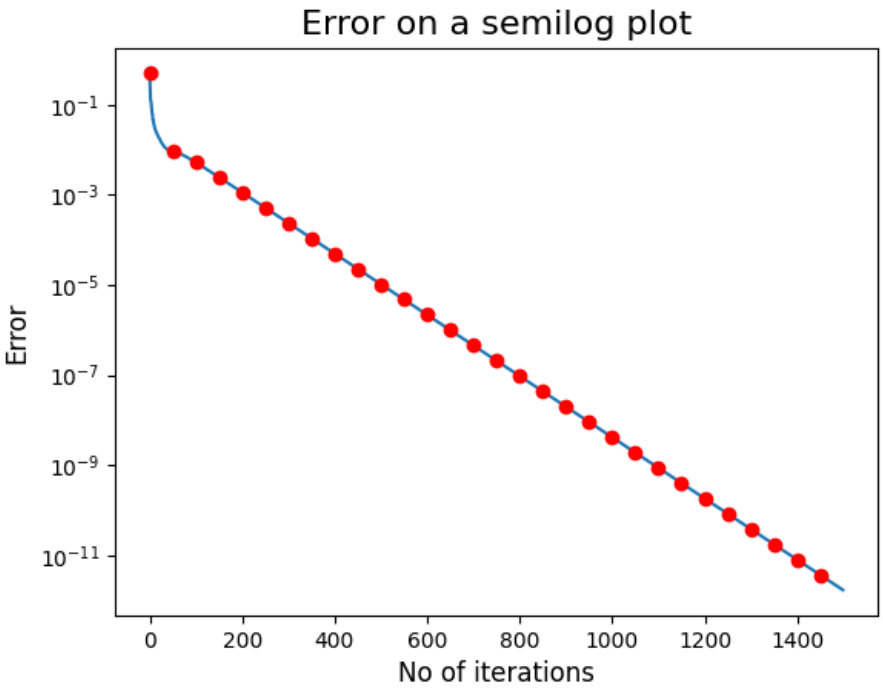

Figure 2: Error on semilog plot

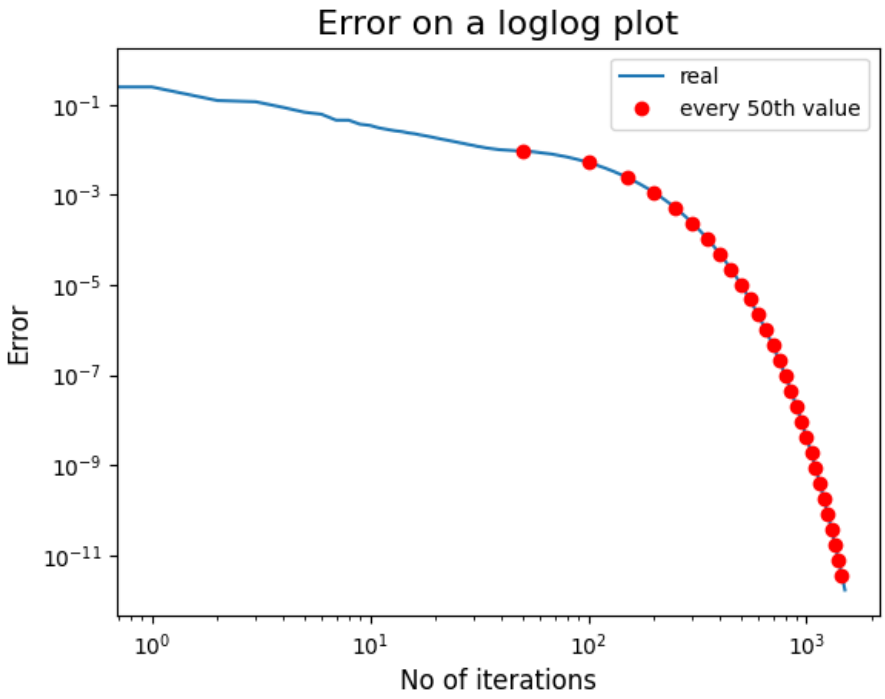

Figure 3: Error on loglog plot

It can be seen that there is an exponential decay for error for larger iteration numbers.(for iterations ¿ 500). We also notice that the fall in error with iteration is really slow.

8 Fitting the error

For iterations more than 500, it is observed that the error decays exponentially. We can see the variations of fitting the error for iterations in the plots ahead. We can construct the fitting values of error using the below code.

```
def error_fit(x, y):
    lny = log(y)
    x_vec = zeros((len(x), 2))
    x_vec[:, 0] = 1
    x_vec[:, 1] = x
    return lstsq(x_vec, lny)[0]
lnA, B = error_fit(range(Niter), err)  # fit1
lnA_500, B_500 = error_fit(range(Niter)[500:], err[500:])  # fit2
```

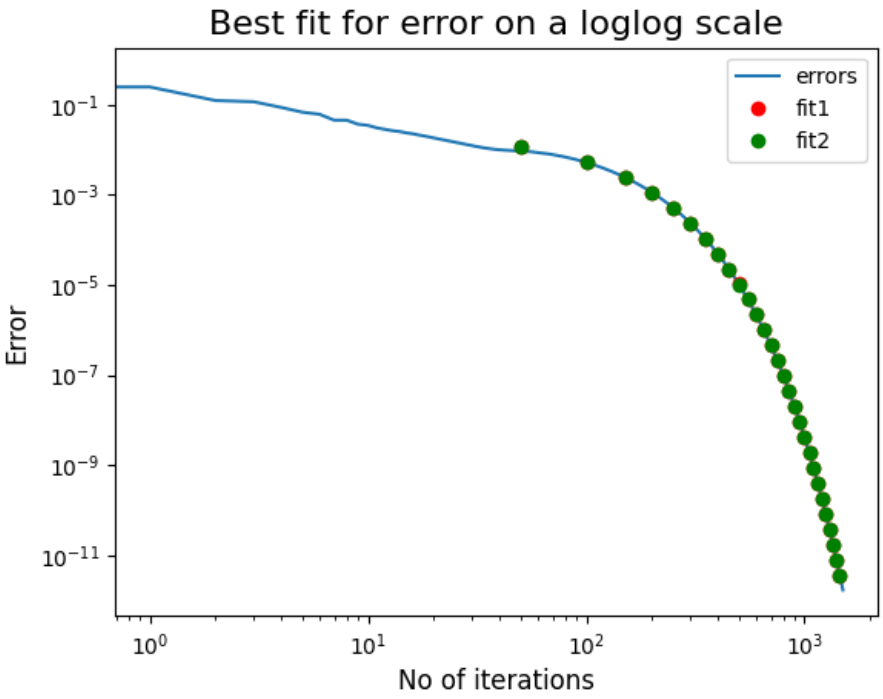

Figure 4: Best fit for error in loglog scale

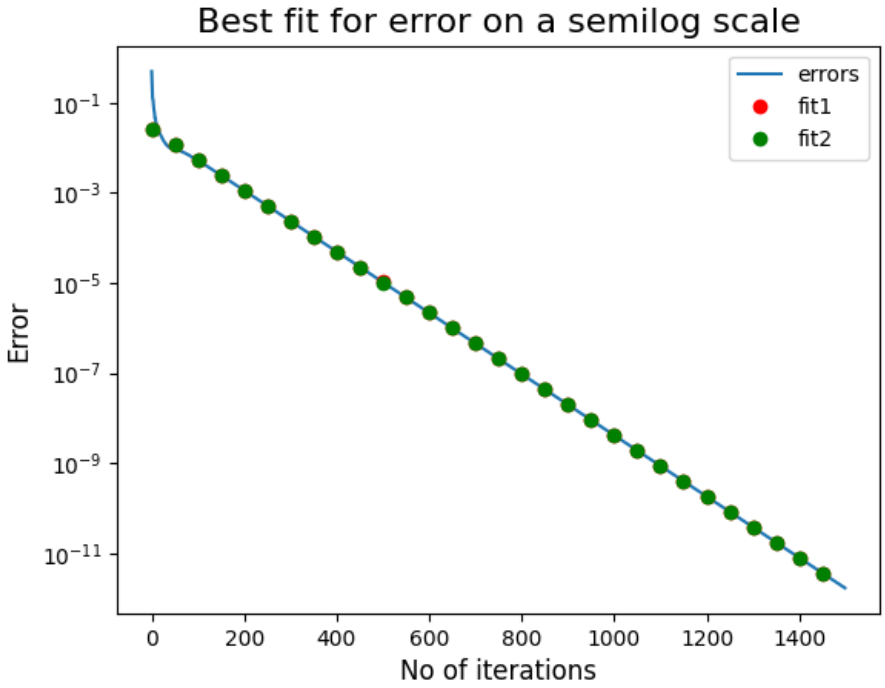

Figure 5: Best fit for error in semilog scale

9 Plotting maximum possible error

We can get the maximum error by adding all the error terms from the point at which error starts decaying exponentially. It is shown by the below code.

```
def max_error(lnA, B, Niter):
    return -exp(lnA + B * (Niter + 0.5)) / B
figure()
semilogy(x[500::50], max_error(lnA_500, B_500, x[500::50]), "ro")
title("Semilog plot of Cumulative Error vs number of iterations", fontsize=16)
xlabel("No of iterations", fontsize=12)
ylabel("Error", fontsize=12)
```

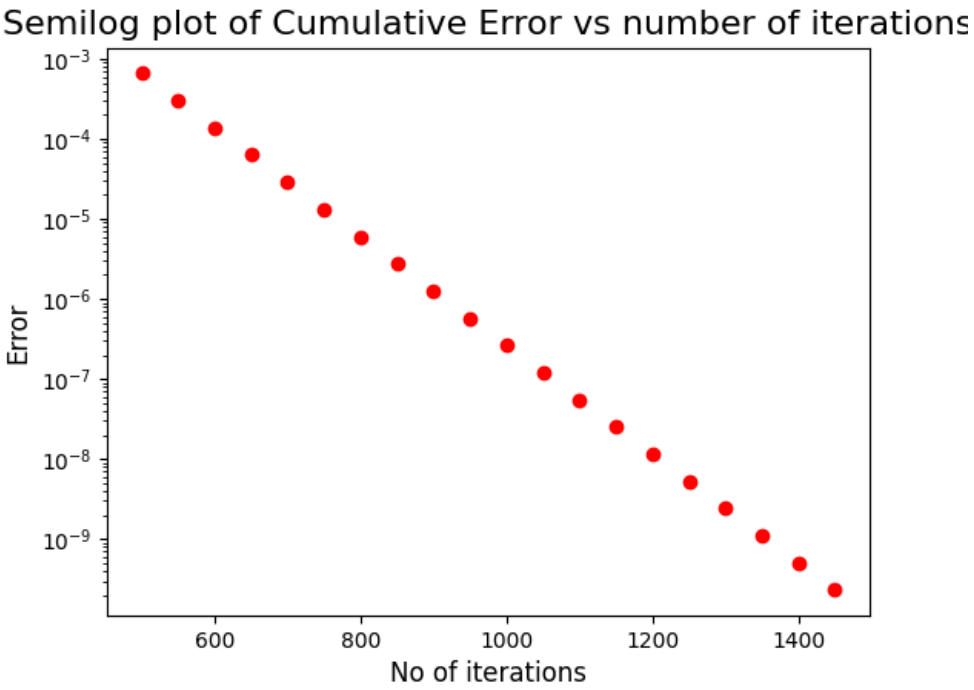

Figure 6: semilog plot of cumulative error

10 Plotting Potential

We can have 2-D contour plot as well as a 3-D surface plot for the potential.
We can construct a 3-D surface plot in python using the *mpl_toolkits* library.
It is done using the below code.

```
# 3-D plot of potential
fig1 = plt.figure(0)
ax = p3.Axes3D(fig1, auto_add_to_figure=False)
fig1.add_axes(ax)
title("The 3-D  surface plot of the potential", fontsize=16)
surf = ax.plot_surface(Y, X, phi.T, rstride=1, cstride=1, cmap=cm.jet)


# Contour plot of potential
figure()
contourf(Y, X[::-1], phi)
colorbar()
title("2D Contour plot of potential", fontsize=16)
xlabel("X", fontsize=12)
```

```
ylabel("Y", fontsize=12)
```

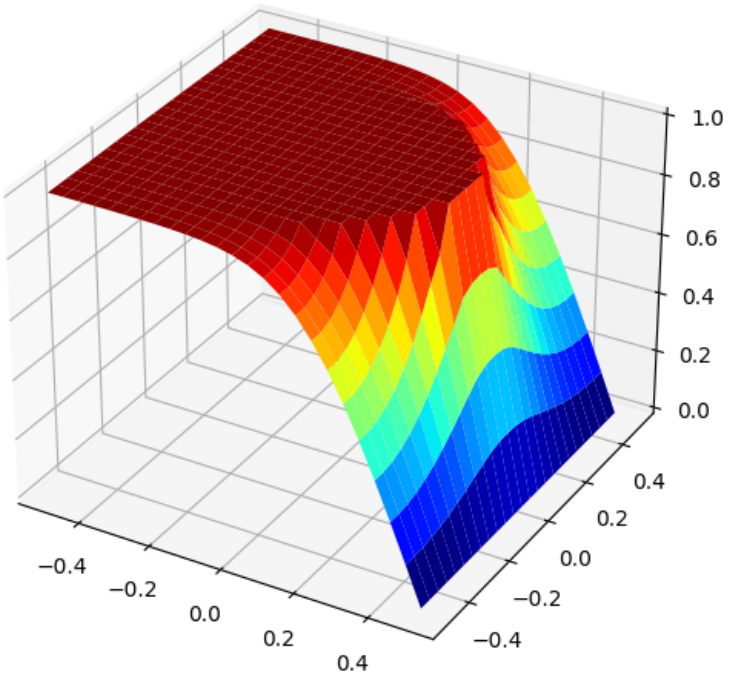

Figure 7: 3-D surface plot of potential

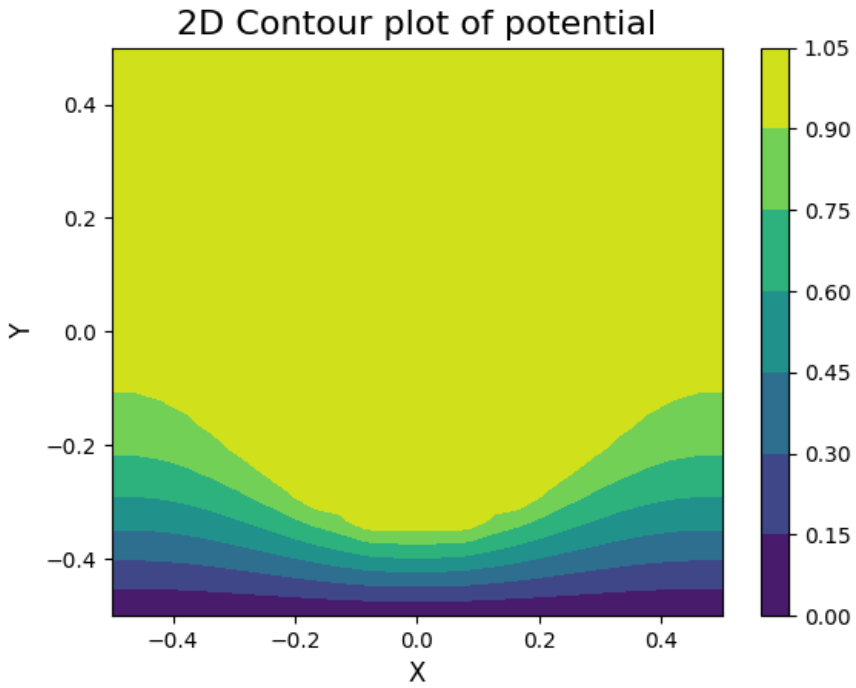

Figure 8: 2-D contour plot of potential

11 Vector plot of current density

The equations involved for finding the current densities are

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{8}$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{9}$$

It can be done in python using

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


# Vector plot of current flow
figure()
quiver(Y[1:-1, 1:-1], -X[1:-1, 1:-1], -Jx[:, ::-1], -Jy)
plot((ind[0] + 0.5 - Nx / 2) / Nx, (ind[1] + 0.5 - Ny / 2) / Ny, "ro")
title("The vector plot of the current flow", fontsize=16)
xlabel("X", fontsize=12)
ylabel("Y", fontsize=12)
```

We will obtain the following vector plot for current densities. The red dots indicate the terminals connected to the wire.

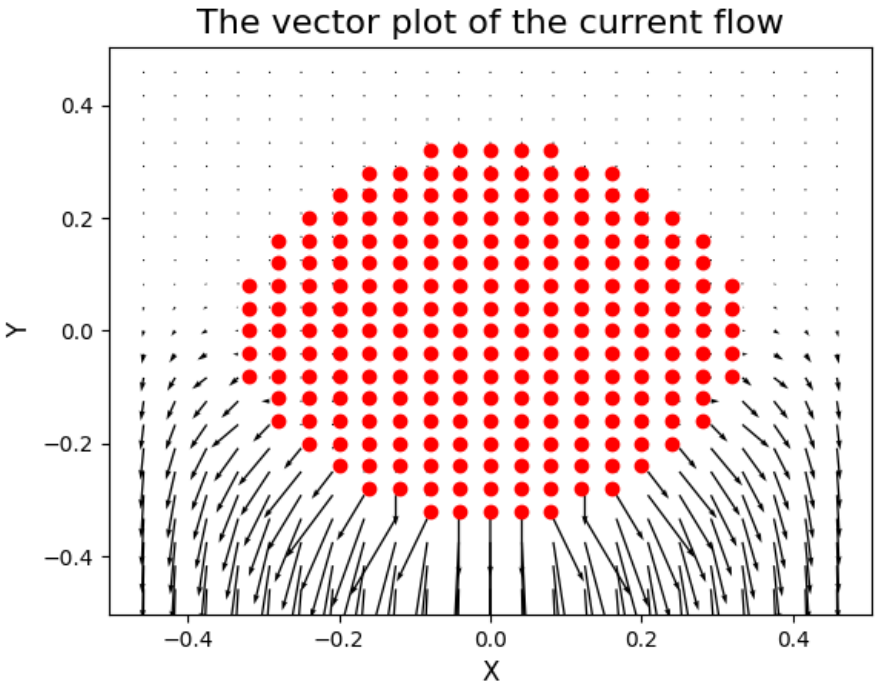

Figure 9: Vector plot of current density

12 Conclusion

- It can be seen that the error doesn't decay rapidly leaving unwanted large deviations and hence is not an efficient method to solve the laplace equation for calculating potentials.
- To get reasonable results from solving the laplace equation, we need to take large number of iterations and also we need to start by taking the first iteration to be a large value so that we get exponential decay.
- It can be seen that the magnitude of current is more at the bottom of the resistor. Hence we can conclude that most of the heating takes place at the bottom of the resistor where most of the current flows