Assignment-5 : Laplace Equation

Sai Gautham Ravipati - EE19B053

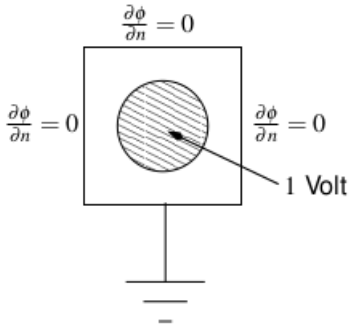
March 25, 2021

Abstract

The currents through the resistor depend on the shape of the resistor, further several regions of the resistor are likely to get hottest when compared to the other. Insights are drawn on several such topics by solving 2D Laplace equation, which when expressed as a difference equation can be solved numerically. The rest of the analysis follows, using Python programming.

1 Introduction

Consider the following arrangement of a cylindrical wire soldered to the center of a square copper plate of side length 1 cm. The wire is held at 1V, and one side of the plate is grounded while the rest remain floated. Ohms



law reads :

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

By expressing Electric field as a gradient of potential and using continuity equation the following results are obtained :

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

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

Assuming that the resistor has constant conductivity,

$$\nabla^2\phi = \frac{1}{\sigma}\frac{\partial\rho}{\partial t} \tag{4}$$

For DC Currents, right side is zero. Hence:

$$\nabla^2\phi = 0 \tag{5}$$

which is nothing but the Laplace equation. This particular equation considered for 2D is :

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \tag{6}$$

This can be expressed as a difference equation, which can be solved numerically, further it can be seen that potential at any point is expressed as the average of its neighbours.

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

The other boundary conditions on ϕ are analysed in detail later. Thus solving this equation numerically, helps analyse the current through resistor.

2 Input Arguments

The input arguments to the code are :

- Nx (size along x) :
  The number of points along x being considered from $(-0.5, 0.5)$. This argument is parsed to assert that it is an integer, and is passed as : '–Nx value' , default value is 25.

- Ny (size along y) :
  The number of points along y being considered from $(-0.5, 0.5)$. This argument is parsed to assert that it is an integer, and is passed as : '–Ny value' , default value is 25.

- R (radius of the wire) :
  The radius of the wire is a float, and is asserted that its less than 0.5 cm, since the square plate is of length 1 cm. Passed as '–R value' , default value being 0.35 cm.

- n (No. of Iterations) :
  Number of iterations to be performed. Passed as '–n value', default values is 1500.

The following lines of code does the work of parsing the input arguments:

```
parser = argparse.ArgumentParser()
parser.add_argument('--Nx', metavar = 'x', type = int, default =
    25, help = 'Size along x')
parser.add_argument('--Ny', metavar = 'y', type = int, default =
    25, help = 'Size along y')
parser.add_argument('--R', metavar = 'r', type = float, default
   = 0.35, help = 'Radius of central lead')
parser.add_argument('--n', metavar = 'Niter', type = int,
    default = 1500, help = 'Number of iterations to perform')
args = parser.parse_args()
[Nx, Ny, r, Niter] = [args.Nx, args.Ny, args.R, args.n]
```

3 Potential Initialization

An array phi of size (Ny, Nx) is initialized and all the points within a distance of radius form the center are assigned a value of 1, as the central lead is always held at 1 V. The same when plotted using a contour plot, looks as shown in the figure below :

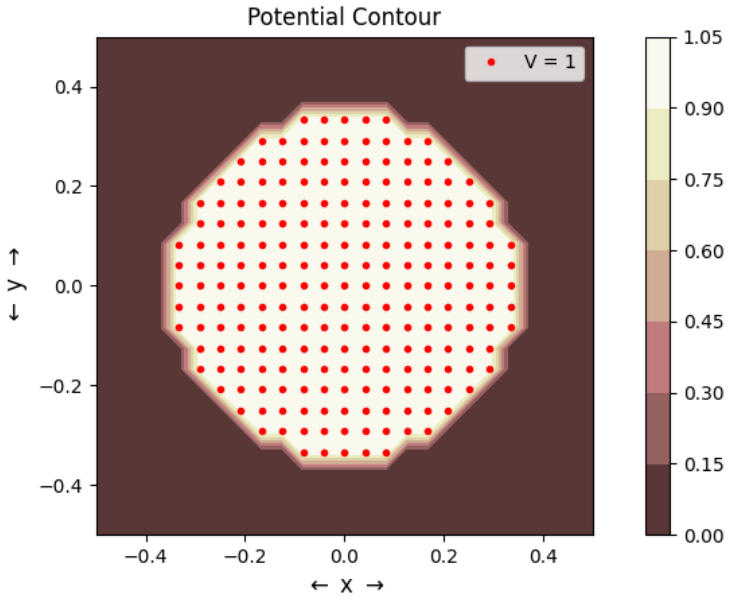


Figure 1: Initialised Potential Contour

The piece of code used is as follows, where 'np.where' has been used to set the entries of phi to one. In the last line meshgrid indices are translated to the range (-0.5,0.5) to plot as red dots in the contour.

```
import numpy as np
```

```
2 phi = np.zeros((Ny,Nx))
3 a = np.linspace(-0.5,0.5,Ny)
4 b = np.linspace(-0.5,0.5,Nx)
5 Y,X = np.meshgrid(b,-a)
6 ii  = np.where((X*X + Y*Y) <= (r)**2)
7 phi[ii] = 1.0
8 x_ii, y_ii = ii[1]/(Nx-1) - 0.5, ii[0]/(Ny-1) - 0.5
```

4 Updating potential

The snippet used to perform iterations and update the potential at every stage is given below. The previous copy of phi is being copied into an array phi_old, and the arrays N, error are initialised to store the values while iterating.

```
1  phi_old = phi.copy()
2  error = np.zeros(Niter)
3  N     = np.arange(Niter) + 1
4  for i in N:
5      phi_old = phi.copy()
6      phi[1:-1,1:-1] = (phi[1:-1,:-2]+phi[1:-1,2:]+phi[:-2,1:-1]+
       phi[2:,1:-1])/4
7      phi[1:-1,0]    = phi[1:-1,1]
8      phi[1:-1,-1]   = phi[1:-1,-2]
9      phi[0,:]       = phi[1,:]
10     phi[-1,1:-1]   = 0
11     phi[ii]  = 1.0
12     error[i-1] = (np.abs(phi - phi_old).max())
```

4.1 Vectorised Implementation

As mentioned previously the Laplace equation when translated to a difference equation, potential at each point is the average of all the neighbouring points. The same is used to update the potential at every stage of iteration. From the code block above, line 6 corresponds to vectorised way of updating the array phi. One added up advantage of this implementation is that, it fastens the iteration unlike using nested loops for the same.

4.2 Boundary Conditions

To assert boundary conditions lines 7-11 have been passed, the central lead is always at a potential of 1 V, and the same is done by line 11. For the boundaries that are left floated, clearly current is tangential, as it can't move out into air. So the gradient of ϕ is tangential to the surface and ϕ doesn't vary in the normal direction. Lines 7,8,9 assert this particular condition at left, right and top boundaries respectively. The bottom side is grounded, and line 10 does the same.

5 Error while Iterating

5.1 Ground truth error

The values of the error are stored after every iteration, which is given by the maximum element of the array formed by absolute difference of phi between successive iterations. The variation of error with iteration has been portrayed in the successive plots. To provide a better visualization every 50th point has been plotted in log-log scale.

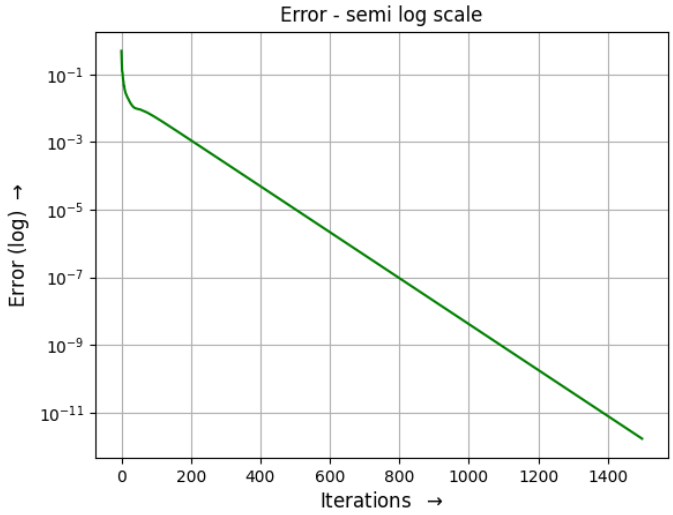


Figure 2: Semilog plot of error with Iteration

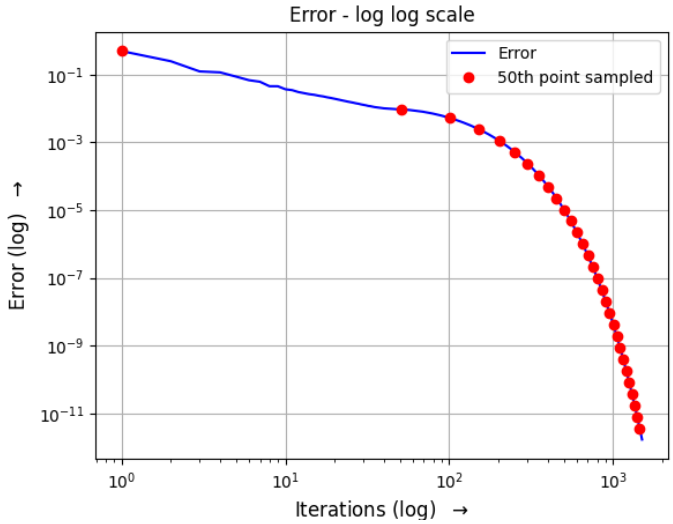


Figure 3: log-log plot of error with Iteration

As inferred from the plots, the error decays very slowly after a certain number of iterations, because of which this method of Laplace solving is not appreciated.

5.2 Error fitting

From the previous plots it is evident that after certain number of iterations, the error varies exponentially, the same can be visualised by fitting the error. Suppose the fit is of the form,

$$y = Ae^{Bx} \tag{8}$$

when log is taken throughout the equation, it reads,

$$logy = logA + Bx \tag{9}$$

The array representation of the same works, where the right side can be expressed as vector product of (log A, B) and (1, x) to give log y. Thus when least squares regression is performed, (log A, B) can be obtained. The following snippet of code implements the same.

```
def fit(x,y):
    x_in = np.vstack([x, np.ones(len(x))]).T
    y_in = np.log(y).T
    b,a  = np.linalg.lstsq(x_in,y_in,rcond=None)[0]
    return np.exp(a),b

a_500,b_500   = fit(N[500:],error[500:])
a_1500,b_1500 = fit(N,error)
error_fit_1 = a_500*np.exp(b_500*N)
error_fit_2 = a_1500*np.exp(b_1500*N)
```

The following plots are obtained when only entries in the error array after 500th iteration (fit1), as well as the whole data-points in the error vector (fit2) are fit. As inferred from the plots both the fits nearly overlap each other, thus the initial shift in the true curve from the exponential caused by initial points is very much less. The values of A,B obtained in both the cases are as follows:

- Points after 500 iterations considered :

  A = 0.026454730872053113
  B = -0.015648069369668786

- All points considered :

  A = 0.02662921218050018
  B = -0.015655264062372585

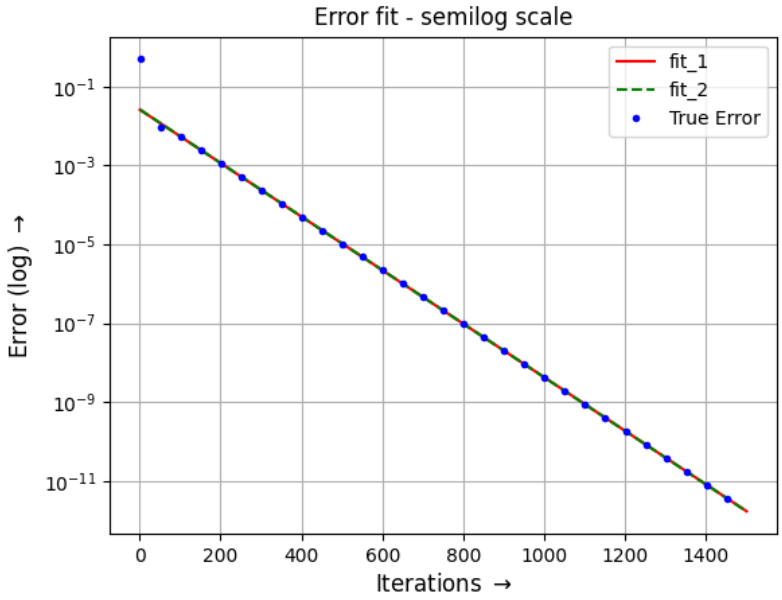


Figure 4: Semilog plot of error-fit with Iteration

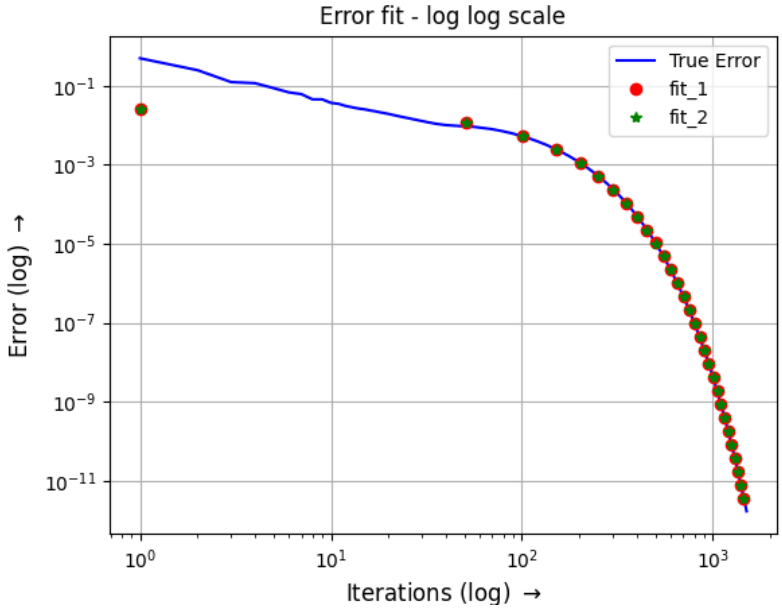


Figure 5: log-log plot of error-fit with Iteration

5.3 Error Bound

The upper bound on the error approximated by accumulating the absolute value, is given below and the same has been plotted. This particular esti-

mate is misleading, when compared to the absolute error, which was plotted previously, and it is evident from the following case where this is order of magnitudes higher even when the regular error went below 0.1 at 100 iterations.

$$-\frac{A}{B}exp(B(N+0.5))$$

The following snippet of code implements the same :

```
def error_estimate(a,b,N) :
    return  -a*np.exp(b*(N+0.5))/b
```

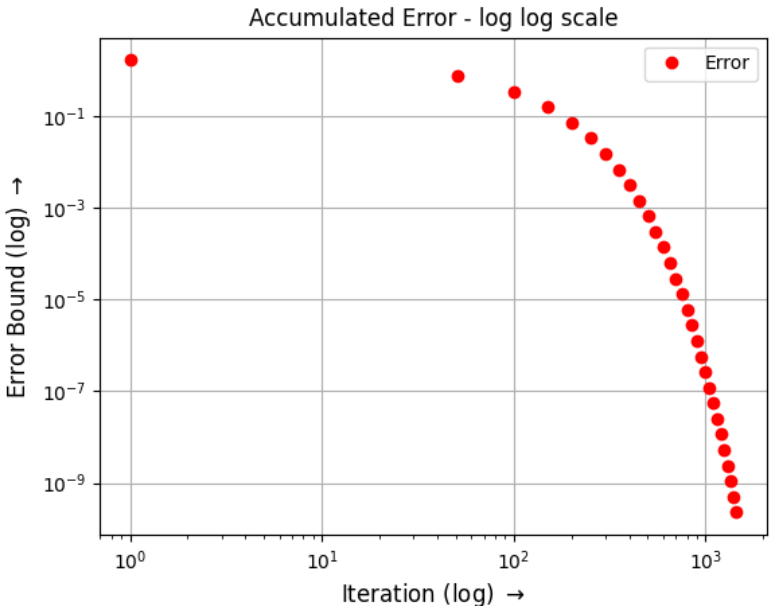


Figure 6: log-log plot of bound of error with Iteration

6 Plot of Updated Potential

After performing 1500 iterations, the array phi is updated, and since error, goes below 1e-11, we can consider that the Laplace equation has been solved, the contour of potential as well as the surface plots are presented in Fig 7.8.

7 Vector Current Plots

The currents can be obtained by taking the gradient of the potential, since σ doesn't effect the envelope of the plot it can be set to 1. The numerically

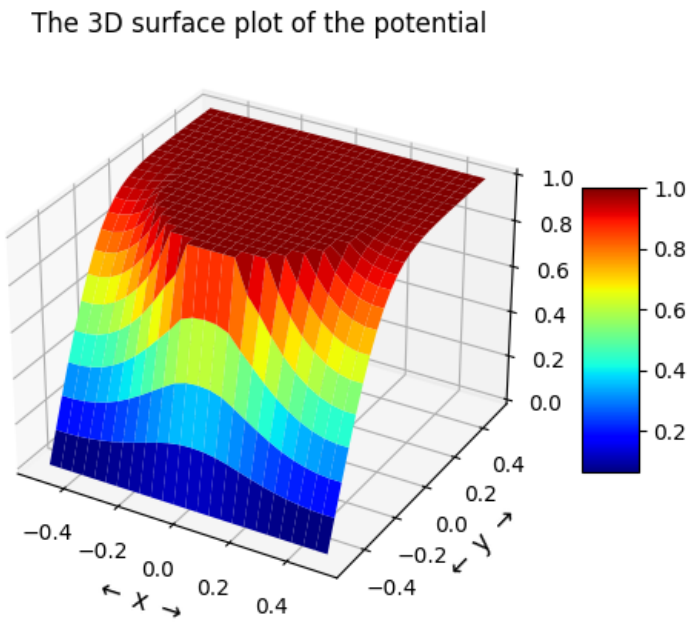


Figure 7: Surface plot of solved potential

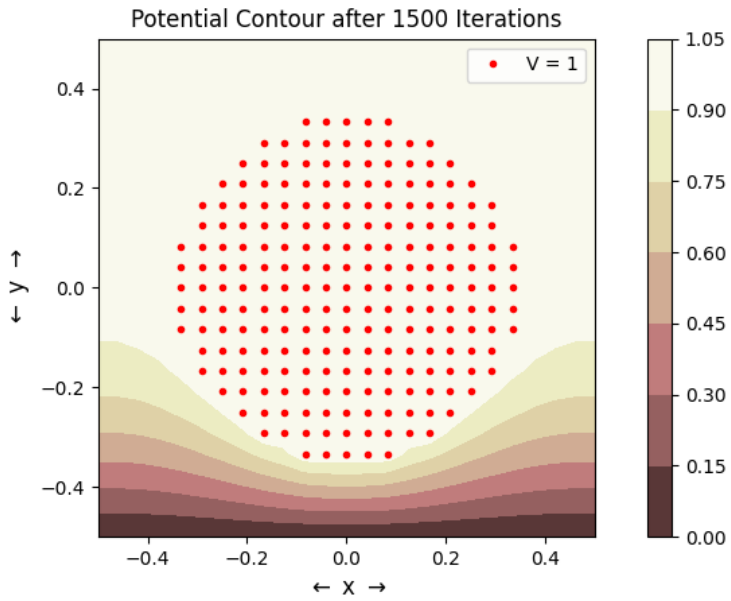


Figure 8: Contour plot of potential

translated expressions for the currents are :

$$J_{x,ij} = \frac{1}{2}(\phi_{i,j-1} - \phi_{i,j+1}) \tag{10}$$

$$J_{y,ij} = \frac{1}{2}(\phi_{i-1,j} - \phi_{i+1,j}) \tag{11}$$

The code snippet used to implement the same is :

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

The plot of vector currents is given by :

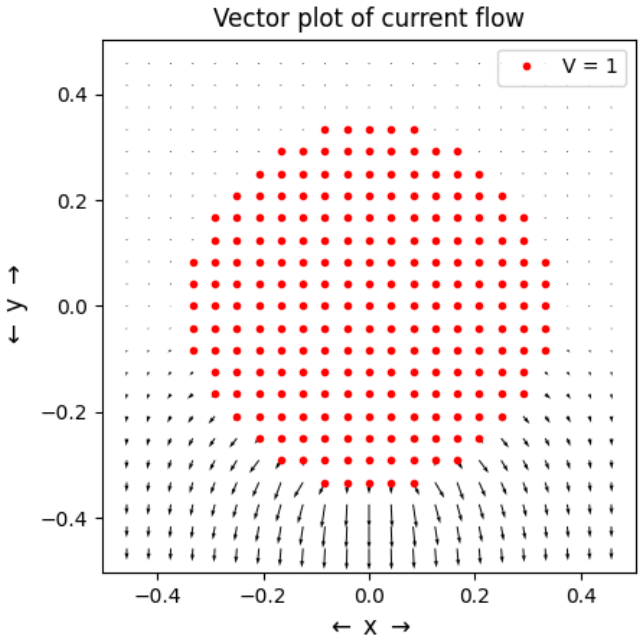


Figure 9: Vector plot of currents

It can be inferred that current fills the entire cross-section of the lead and flows towards the grounded electrode. Hardly any current flows through the upper half. This can be reasoned out analytically as well as through the potential contour plot. Clearly the gradient of the potential is strong in the lower half from fig. 8, and since current density is proportional to the same, the currents are higher in the lower half. Alternatively for the potential to drop from 1 V to 0 V at the ground electrode, the system tries to minimise the resistance to flow of charges, as a result of which, hardly no current flows through the upper half, and potential is dropped to 0 V by facing a smaller resistance.

8 Variation of Temperature

As most of the current, flows through the lower half, the ohmic losses are also high in the lower half, thus heating up the plate, so by intuition, temperature shall be higher in the lower half when compared to upper half. The following equation :

$$\nabla.(\kappa \nabla T) = \frac{|J|^2}{\sigma} \tag{12}$$

is solved in a similar way of updating the potential, to obtain the contour of temperature. The boundary conditions assume, no change of temperature along the normal direction at left, right and top sides of the sheet, while it is 300K at the location of electrode and the ground. The following code snippet does the work of solving the equation :

```
T      = np.zeros((Ny,Nx)) + 300
T_old = T.copy()
for  i  in  N:
    T_old = T.copy()
    T[1:-1,1:-1] = (T[1:-1,:-2] + T[1:-1,2:] + T[:-2,1:-1] +
    T[2:,1:-1] + (Jx)**2 + (Jy)**2)/4
    T[1:-1,0]     = T[1:-1,1]
    T[1:-1,-1]    = T[1:-1,-2]
    T[0,:]        = T[1,:]
    T[-1,1:-1]    = 300
    T[ii]         = 300
```

The contour plot of temperature is presented below in Fig. 10. Clearly the lower half region is at a higher temperature when compared to upper region.

9 Conclusion

Laplace equation solved by this particular approach, although provides a good solution, this method is not suggested because error decays exponentially, as a result of which it reduces very slowly. The fits of the error are good estimates, and don't deviate much from the same after certain number of iterations.

It can be inferred from the plots that most of the potential gradient occurs in lower half, as a result of which the current are also confined to the lower half, thus producing more heating and higher temperatures when compared to the upper half.

Note :

All the plots are plotted using the following functions :

- Contour plot - *plt.contour*()

- Quiver plot - $plt.quiver()$
- Loglog plot - $plt.loglog()$
- Semilog plot - $plt.semilogy()$
- Surface plot - $plot\_surface()$

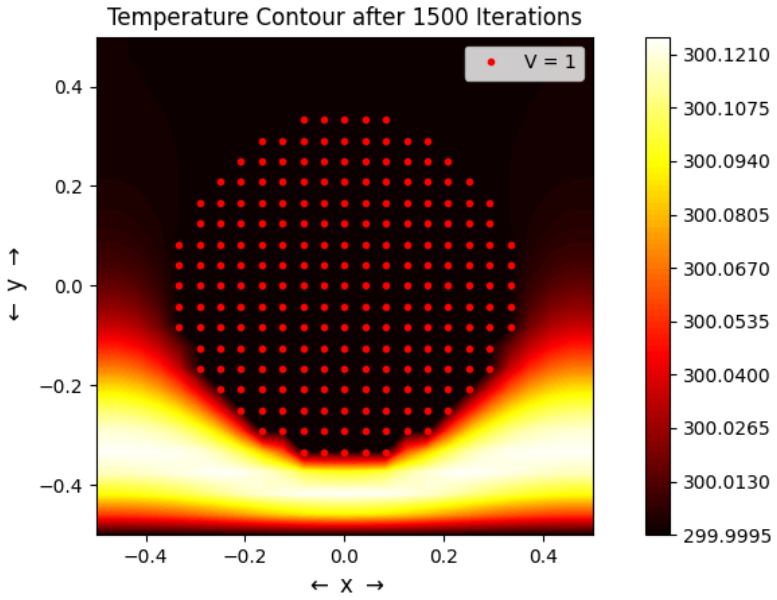


Figure 10: Contour plot of Temperature