Solving the laplace equation numerically

Viroshaan Uthayamoorthy, NTNU 01/01/2014

Introduction

The motivation for solving the laplace equation numerically is that this will help provide insight into how the electrostatic laws work and gives a certain application of the theories taught in lectures. This report will provide some of the theoretical background into why we choose to solve the laplace equation, and how our numerical solutions relfect expected behaviour of the electric potential.

Theory

There are two particular versions of maxwells equations that govern the realm of electrostatics.

$$\nabla \cdot \vec{E} = \frac{\rho}{\epsilon_0} \tag{1}$$

$$\nabla \times \vec{E} = 0 \tag{2}$$

The last of these equations implies that the electric field can be written as a scalar field with the relation $\vec{E} = -\nabla V$ due to the fact that the curl of a gradient equals zero, and thus the electric field will still fufill the second maxwell equation. In our project, we choose to focus on regions in which the charge density equals zero. Inserting this into the first equations and using the above rewriting of the electric field we recieve the Laplace equation:

$$\nabla^2 V = 0 \tag{3}$$

In this project we are going to solve the laplace equation as a means to simulate the potential in a box given to a certain set of boundary conditions. In our project, the boundary conditions are given as:

$$V(x = 0, y) = 0 \tag{4}$$

$$V(x = L, y) = 0 \tag{5}$$

$$V(x, y = L) = 0 \tag{6}$$

$$V(x, y = L) = V_0(x) \tag{7}$$

The potential $V_0(x)$ can be arbitrary and gives rise to different potentials. Using seperation of variables and following the steps provided in Griffiths, it is possible to solve the laplace equation given our set of boundary conditions. This gives rise to the following potential:

$$V(x,y) = \sum_{n=0}^{N} A_n \sin(\frac{\pi n}{L}x)\sinh(\frac{\pi n}{L}y) \tag{8}$$

Introducing the dimensionless variable $\xi = \frac{x}{l}$, the above expression can be rewritten as:

$$V(x,y) = \sum_{n=0}^{N} A_n \sin(\pi n \xi)\sinh(\frac{\pi n}{L}y) \tag{9}$$

As shown above, the potential can be looked at as a fourier series, and increasing n gives more order contributions to the solution. A complete solution is then found by finding the coeficients A_n which varies with different n. Using the fourth boundary condition and the orthogonality of each order contribution of the fourier series, one can find each contribution A_n, thus giving us a complete solution to our problem, up to a determined order N.

$$A_n = \frac{2}{\sinh(\pi n)} \int_0^1 V_0(\xi)\sin(\pi n \xi) d\xi \tag{10}$$

The solution that this potential provides is unique by the uniqueness theorem. With arbitrary potentials, the above integral can be cumbersome, and thus this project will focus on solving this integral and simulate the resulting potential numerically.

Numerical solution

This section will provide different aspects of the code and an explanation of each implementation.

Listing 1: The integrand of A_n

```
1 #The integrand of the A_n expression. Inputs an arbitrary function V_0(x) ↩
    and returns the integrand of A_n.
2 def integrand(x, n, V0):
3     return 2*V0(x)*np.sin(np.pi*n*x)
```

This section of the code is used for integration purposes since the integration-function used requires the integrand expression as a function in python. Note that $\frac{1}{sinh(\pi n)}$ is excluded in this expression. The first implementation of this code had this expression included in the integrand-function. This however creates numerical errors which escalates with increasing n when implemented later in the code.

Listing 2: Termination criteria of calculated potential

```
1 #Input: potential; V_0, tolerance; tol and length of the axes; N_dim
2 #Output: number of necessary coefficients i
3 def Necessary_coefficients(v0,tol,N_dim):
```

```
4       x = np.linspace(0,1,N_dim)
5       V = v0(x)
6       V_calc = np.zeros(N_dim)
7       i = 1
8       while (np.linalg.norm(V-V_calc,ord = np.inf) > tol):
9           V_calc = np.zeros(N_dim)
10          A = np.zeros(i)
11          for n in range(1,i+1):
12              A[n-1] = quad(integrand, 0, 1, args=(n,v0))[0]
13              V_calc += A[n-1]*np.sin(np.pi*n*x)
14          if (i == 200):
15              break
16          i+= 1
17      print("The Necessary number of fourier coefficients are: ",i-1)
18      return i-1
```

This function calculates the necessary amount of coefficients to approximate the V_0 potential within a given tolerance. The logic behind this is that the code runs a while loop where every iteration checks that the infinity norm of the difference vector between the calculated and initial potential is bigger than some given tolerance. If so, it includes the next fourier term in the calculated potential, increments the counter-variable i and runs the while-loop once again. Lines 11-13 does the A_n calculations, where the integration-function quad from the scipy library is used. At the edge $y = L$, the potential-expression is reduced to:

$$V(x,y) = \sum_{n=0}^{N} A_n \sin(\pi n \xi_y) \tag{11}$$

This explaines the expression for the calculated potential in line 13 in the above code. However this code experiences convergence issues when the amount of fourier coefficients terms exceed ~ 200 terms. The above block of code does not need to have this limitation, however later calculations of the potential $V(x,y)$ includes $\sinh(\pi n \xi)$ expressions which quickly blow up. Therefore a limitation on the number of included fourier coefficients is set to 200.

Listing 3: Calculation of potential

```
1   def Entire_potential(tol,v0,N_dim):
2       xx,yy = np.meshgrid(np.linspace(0,1,N_dim),np.linspace(0,1,N_dim))
3       # Creates a grid with x and y as coordinate points. Meshgrid makes ↩
            calculations quicker and easier
4       n_fourier = Necessary_coefficients(v0,tol,N_dim)
5
6       A = np.zeros(n_fourier) #Creates a list for all the A_n coefficients.
7       Z = np.sinh(np.pi*0*yy)*np.sin(np.pi*0*xx)
8       # Initialize a meshgrid with zeros in each position
9       for i in range(1,n_fourier +1):
```

```
10          A[i-1] = quad(integrand, 0, 1, args=(i,v0))[0]
11          #Calculate all the fourier coefficients using scipy integration
12          Z += A[i-1]*np.sinh(np.pi*i*yy)/(np.sinh(np.pi*i))*np.sin(np.pi*i*↩
                xx)
13          # To each coordinate we add on the fourier components.
14      return xx,yy,Z
```

In order to calculate the potential at evey x,y-point within the box, the function meshgrid was used. This effectively creates a coordinate system which is able to do similar calculation at every point within said grid. With this one can simply calculate the value of the potential at every single point within the grid using the for loop going from line 9-12 in the above block of code.

Listing 4: Plotting

```
1  fig = plt.figure() #3D plot
2  ax = fig.gca(projection='3d')
3  surf = ax.plot_surface(xx,yy,Z, cmap = 'gist_rainbow')
4  plt.contour(xx,yy,v0(xx),zdir = 'y',offset = 1, cmap = 'Dark2')
5  plt.show()
6
7  plt.figure() #Contour plot
8  plt.contour(xx,yy,Z)
9  plt.show()
10
11 #Calculating electric field
12 dy, dx = np.gradient(Z)
13
14 fig2 = plt.figure() #Electric field plot
15 fig2, ax2 = plt.subplots()
16 ax2.quiver(xx, yy, dx, dy)
17 ax2.set(aspect=1, title='Quiver Plot')
18 plt.show()
```

Results

From the upper left figures, we see that the numerical implementation matches the boundary conditions given by equations (4) through (7) very well. It also matches the boundary potential V_0 quite well. This is specially the case for sinusoidal potentials as in figure 1 and 2, where the difference between the calculated potential and actual boundary potential is of order 10^{-16} as shown in the lower right figures. This is as expected since we only expect one single contribution in the Fourier expansion of the potential since we are using sinusoidal contributions to approximate something that is also sinusoidal. However the numerical solution struggles to

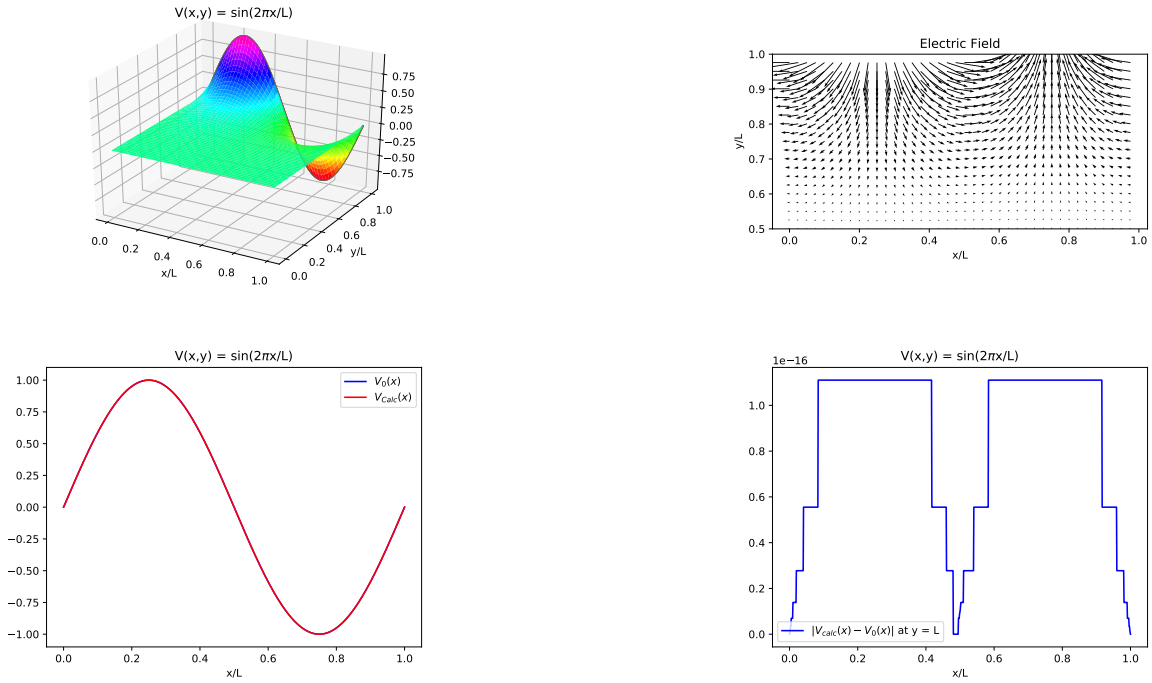

Figure 1: The upper left figure shows the calculated potential due to $V_0 = \sin(2\pi x/L)$. The upper right figure shows the corresponding calculated electric field. The lower left figure shows both the calculated potential, $V_{calc}(x, y = L)$, and the boundary potential $V_0(x, y = L)$ at y = L. The lower right figure plots the difference $|V_{calc}(x, y = L) - V_0(x, y = L)|$.

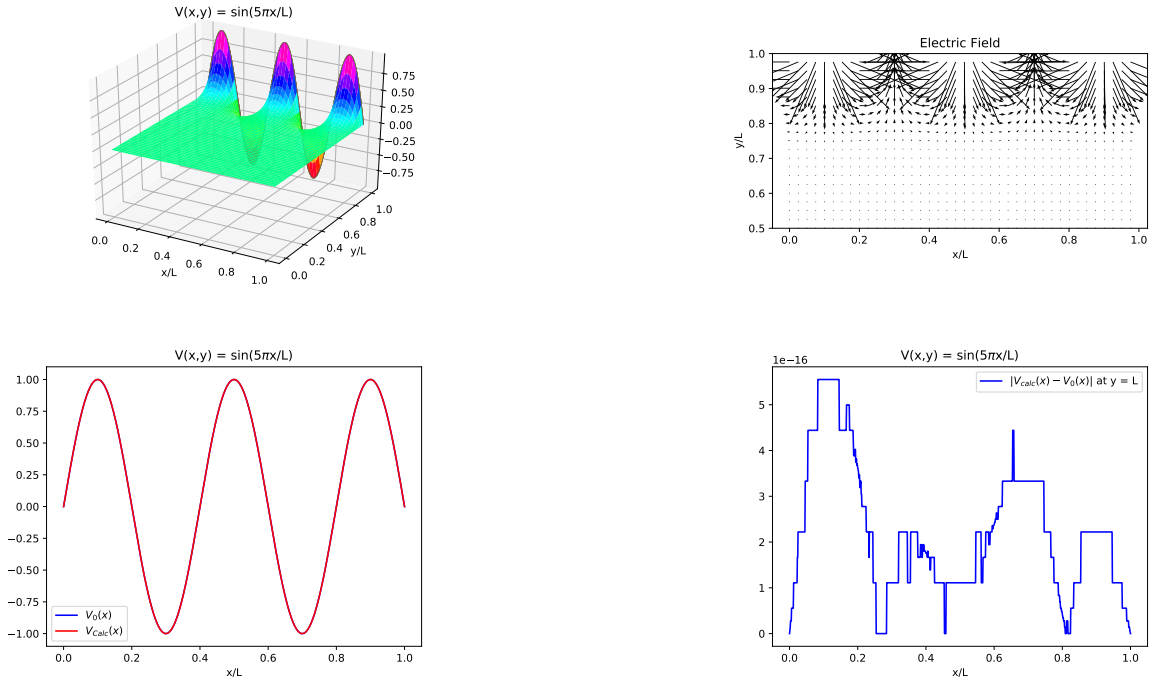

Figure 2: The upper left figure shows the calculated potential due to $V_0 = \sin(5\pi x/L)$. The upper right figure shows the corresponding calculated electric field. The lower left figure shows both the calculated potential, $V_{calc}(x, y = L)$, and the boundary potential $V_0(x, y = L)$ at y = L. The lower right figure plots the difference $|V_{calc}(x, y = L) - V_0(x, y = L)|$.

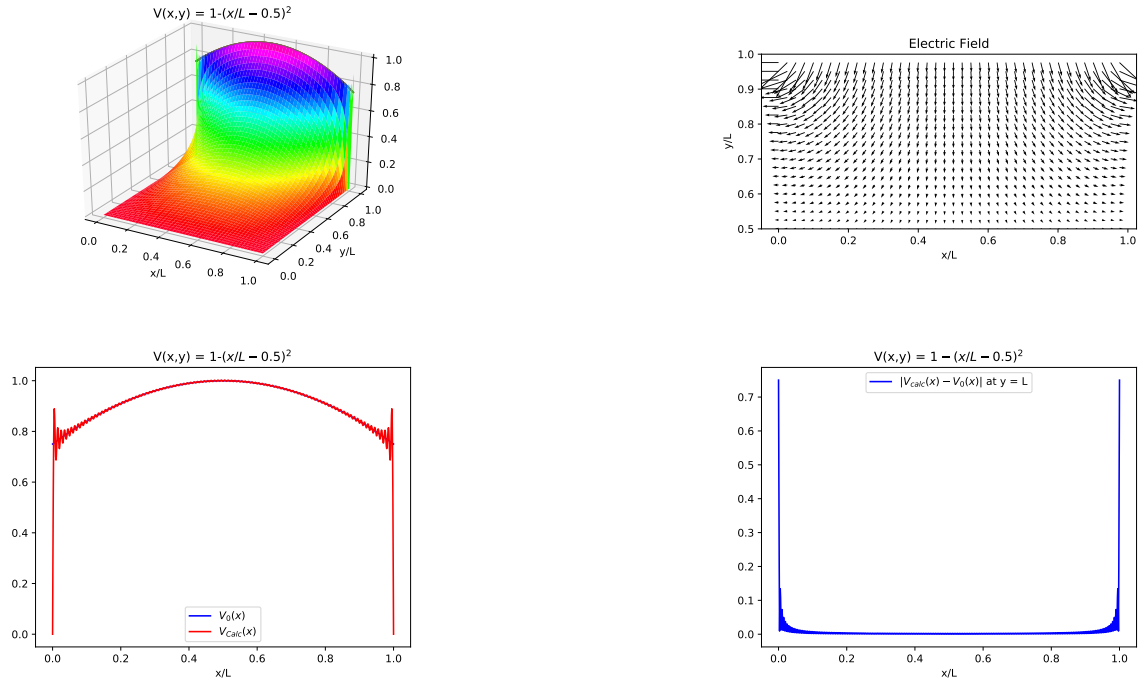

Figure 3: The upper left figure shows the calculated potential due to $V_0 = 1 - (x/L - 0.5)^2$. The upper right figure shows the corresponding calculated electric field. The lower left figure shows both the calculated potential, $V_{calc}(x, y = L)$, and the boundary potential $V_0(x, y = L)$ at y = L. The lower right figure plots the difference $|V_{calc}(x, y = L) - V_0(x, y = L)|$.

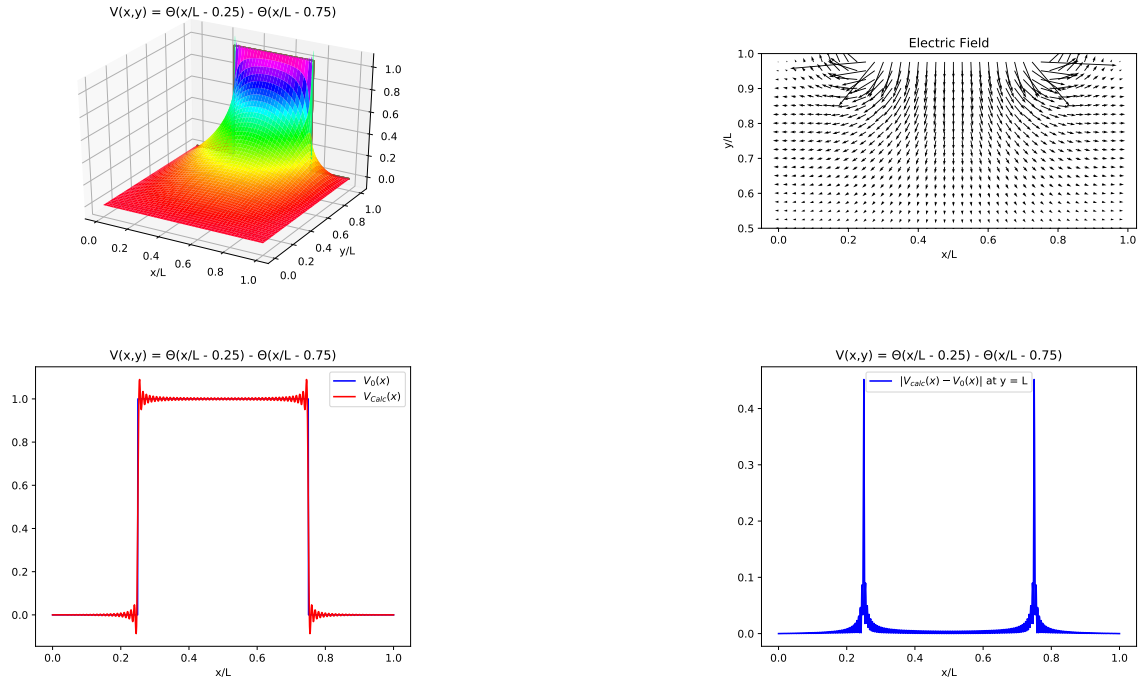

Figure 4: The upper left figure shows the calculated potential due to $V_0 = \Theta(x/L - 0.25) - \Theta(x/L - 0.25)$. The upper right figure shows the corresponding calculated electric field. The lower left figure shows both the calculated potential, $V_{calc}(x, y = L)$, and the boundary potential $V_0(x, y = L)$ at y = L. The lower right figure plots the difference $|V_{calc}(x, y = L) - V_0(x, y = L)|$.

approximate arbitrary potentials as evident by figures 3 and 4. Inspection of the lower sub-plots of figures 3 and 4, one sees that there are particular deviance further away from the center, and particularly when there is a sudden change in the potential. This is a phenomenon called the Gibbs phenomenon and it says that the Fourier series tend to overshoot at simple discontinuities. In figure 3 we see discontinuity at x = 0 while in figure 4 we see discontinuities at x = 0.25 and x = 0.75.

Due to the nature of the sinh(x) function, there are some limitations set by the libraries used to evaluate the potential. The expression of the potential at arbitrary points is proportional to $\frac{\sinh(\pi ny/L)}{\sinh(\pi n)}$ (see equation (8) and (10)). Analytically, this expresion cannot exceed 1, since the maximum value of y/L is 1. However, numerical implementations evaluate each terms separately, and thus for large n, the $\sinh(\pi ny/L)$ blows up. Therefore the code has a termination criteria in the amount of Fourier terms allowed to be evaluated. From experience, the $\sinh(\pi ny/L)$ experiences round-off errors at around n = 267 terms, and thus to limit the running time, this implementation includes the terms up to only n = 200.

As explained in the numerical solution section of this code, a tolerance test using the infinity norm of the difference vector between $V_0(x) - V_{calc}$ was used. This checks that the biggest difference between calculations and actual potential is within a certain tolerance. The infinity norm criteria works well with the sinusoidal boundary potentials, where the $V_0 = \sin(2\pi x/L)$ and $V_0 = \sin(5\pi x/L)$ requiring the expected n = 2 and n = 5 Fourier terms to converge within the tolerance 10^{-15}. However the remaining two potentials both exceeded the run-time criteria for the same tolerance, i.e it needed at least n = 200 terms to converge. If the vector norm didn't consider the discontinuous-contributions in the norm-calculations, the amount of Fourier coefficients necessary to approximate the boundary potential would be significantly lowered as evident by the lower right figures.

Potential solutions to the problem of the boundary potential $V_0 = 1 - (x/L - 0.5)^2$ could be to either raise the whole potential by adding a constant to equation (8) or expanding by using cosines instead of sines, but both corrections would break boundary conditions (4) and (5). To verify that Gibbs phenomenon is what is limiting the accuracy of the simulation, a comparison was ran using the boundary potential $V_0 = 4x(x - 0.5)$. This potential is both quadratic and takes on a similar form as the boundary potential that was attempted to be compared. Using a tolerance of 10^{-4}, the number of necessary runs were only 45, and yielded the results shown in figure 6. The crucial difference between these two quadratic potentials is that the first potential has a singularity at the edges. This comes form the fact that the boundary conditions (4) and (5) determines that the potential is zero at the edges x = 0 and x = L, however the boundary potential V_0 is nonzero here. Since sinusoidal functions are always zero at these edges, the Fourier terms tend to overshoot at these points, creating Gibbs phenomenon.

The relation between electric field and potential is $\vec{E} = -\nabla V$. Calculating the electric field was done numerically by implementing the Numpy gradient function. The resulting electric fields are shown together with each simulation. The electric field takes expected form, namely that when the potential is positive, the electric field spreads outward away from the potential, while if the potential is negative, the electric field folds inwards.

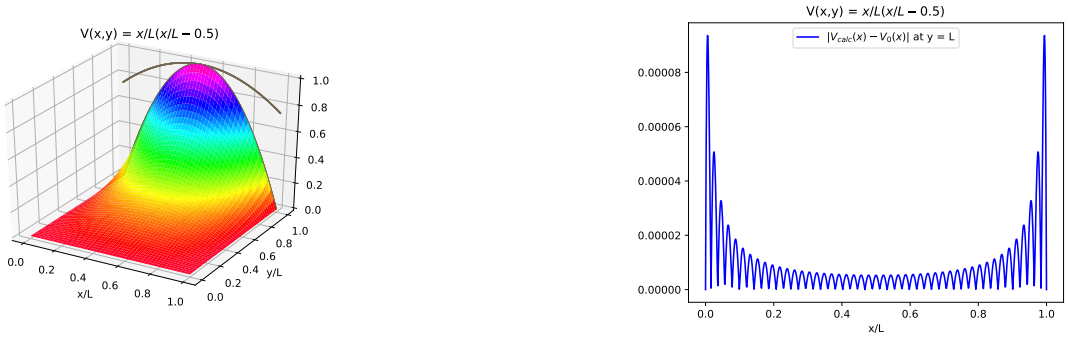

Figure 5: The left plot shows the resulting potential due to boundary potential $V_0 = 4x(x - 0.5)$. The boundary potential $V_0 = 1 - (x/L - 0.5)^2$ is also shown in the same figure. The right figure shows the error at each point between the calculated potential and the actual boundary potential.

Conclusion

The conclusion of this report is that a numerical solution to the Laplace equation is a good approximation depending on the potential that is being used. In the case in which the boundary potential is sinusoidal, one would only expect contribution from one Fourier term, and the calculated potential would match the boundary potential well. It also matches well in the case of continous potentials that have zero potential at the edges of the box $x = 0$ and $x = L$. However in the case of singularities in the boundary potential, Gibbs phenomenon requires several more terms to converge within certain tolerances. One thus experiences convergence issues due to numerical instability due to the $\sinh(x)$ function.

References

Griffiths, David J. (2017) Introduction to Electrodynamics (4th ed.). Cambdrige University Press