Numerical Calculation of the Electric Potential and Electrostatic Field inside a Bounded and Symmetrical Box

Fernando Fernandez*
University of British Columbia, Physics 210
(Dated: December 20, 2007)

A program was written in Python that is capable of calculating the electric potential using the method of finite differences and of producing a graph showing the equipotentials, as well as the electrostatic field, using the simple method of “left” and “right” differences. This is accomplished in a simple situation: inside a 2-D square box whose sides are charged to any voltage.

INTRODUCTION

The main idea was to write a program in Python that would allow me to calculate the electric potential and electrostatic field, and showing a graph of both. This was done by using a simple algorithm from the finite differences numerical method, used to calculate Laplace’s equation, and then use this solution to calculate the electrostatic field using the numerical method of “left” and “right” differences.

The problems that can be calculated must are very rather simplistic unfortunately: the inside of a two dimensional box, whose sides must be equal. The only variables are the different voltages which are given to each of the sides, as Dirichlet Boundaries. As such, it is only an exercise of solving Laplace’s equation numerically, with the added bonus of a visualization of the abstract ideas of the fields.

MATH AND METHODS

Maxwell’s Equations

As this program will deal with electrostatic fields, meaning no time dependence, and no magnetic fields, only 2 of the 4 famous equations are needed:

\[ \nabla \times \vec{E} = 0 \quad (1) \]
\[ \nabla \vec{E} = 0 \quad (2) \]
\[ \vec{E} = \nabla V \quad (3) \]
\[ \nabla^2 V = 0 \quad (4) \]

From (Eq.1), we can say that \( \vec{E} \) is just the gradient of a scalar function (Eq.3). Since a scalar, is far easier to calculate than the a vector field, in order to calculate the electrostatic field, the potential is calculated first by solving Laplace’s equation (Eq.4).

Finite Difference Method

In order to calculate \( \nabla V \), the simplest method to use is called “The Method of Finite Difference”. It works by setting up a large grid, which will contain a value at each node. At first, the grid only contains the values of the boundary conditions at the boundaries of the grid itself, and every other node is 0. To fill in the rest of the values, what is done is calculate the value of a certain node by averaging the value of the four contiguous ones [1]. So this is done for each node, several times until a minimum error is reached. This error is calculated as the total sum of the difference of the first value of the node with the newly calculated one, squared (Eq.5).

\[ \text{error} = \sum (\text{original} - \text{new})^2 \quad (5) \]

This method comes from the following equations, which are obtained by expanding the Taylor Series around one of the nodes \( f(x, y) \) until the third argument. Here it is shown for two dimensions only, but it is easily extended to three:

\[ f(x + h, y) = f(h) + h \frac{df}{dx} + \frac{h^2}{2} \frac{d^2 f}{dx^2} \quad (6) \]
\[ f(x - h, y) = f(h) - h \frac{df}{dx} + \frac{h^2}{2} \frac{d^2 f}{dx^2} \quad (7) \]
\[ f(x, y + h) = f(h) + h \frac{df}{dy} + \frac{h^2}{2} \frac{d^2 f}{dy^2} \quad (8) \]
\[ f(x, y - h) = f(h) - h \frac{df}{dy} + \frac{h^2}{2} \frac{d^2 f}{dy^2} \quad (9) \]

Now, by adding this four equations together, and acknowledging from Laplace’s Equation that the sum of the second derivatives equals 0, we obtain the equation used [2]:

\[ f(x, y) = \frac{f(x + h, y) + f(x - h, y) + f(x, y + h) + f(x, y - h)}{4} \quad (10) \]
“Left” and “Right” Differences Method

This method is nothing more than the standard definition of a derivative, in which we calculate the slope between two points that are separated by ‘h’. As \( h \to \infty \) we find the derivative, in this case \( h \) is the number of nodes for each coordinate. “Left” (Eq.11) and “Right” (Eq.12) only refer to if we are adding or subtracting \( h \). The “Right” version is needed for the right and top boundaries. Mathematically:

\[
\frac{df(x)}{dx} \approx \frac{f(x) - f(x - h)}{h} \tag{11}
\]

\[
\frac{df(x)}{dx} \approx \frac{f(x + h) - f(x)}{h} \tag{12}
\]

THE PROGRAM

Explanatory Example

The best way to explain how the program works would be with a simple example of what it can do. Let’s start by defining a simple Boundary Value Problem:

\[ x_0 = 0, V = 50 \tag{13} \]

\[ x_1 = 1, V = 50 \tag{14} \]

\[ y_0 = 0, V = 0 \tag{15} \]

\[ y_1 = 1, V = 0 \tag{16} \]

At first there is a simple menu with few options. The first thing would be to have the user enter the boundaries, and/or set a different error limit and the step size if desired, the default is setup at \( 10^{-3} \) and 25 respectively. Next, the grid will be created using the boundaries given, its size will depend on them, and it will be divided by a certain number of steps, as well as populated with the given values of \( V \). After this, the user can now ask the program to calculate the potential and electric fields, which will do so and present three plots (Figs. 1-3). And that’s it.

DISCUSSION

There is a lot of missing things from the project from what I thought I would be able to do. For instance, I could not find a way to work with polar coordinates, boundaries to infinity, setup more than one box, not even to work with non-square boxes. Something was wrong with the logic for those last two. Specially when trying to setup boundaries that were not at the corners of the grid, it would simply calculate near where the numbers were, but not the entire grid.

Also, I could not make it possible to allow the user to replot in case the boundaries had changed, since matplotlib crashes if one tries to plot a second time. I took out the plotting of the boundaries, since I thought it was superfluous since the only thing one can get is a box anyways. Let the graph be the boundaries themselves, there was no point in adding more lines to them.

* Electronic address: fxfernandez@gmail.com


FIG. 3: Both $V$ and $\vec{E}$ together