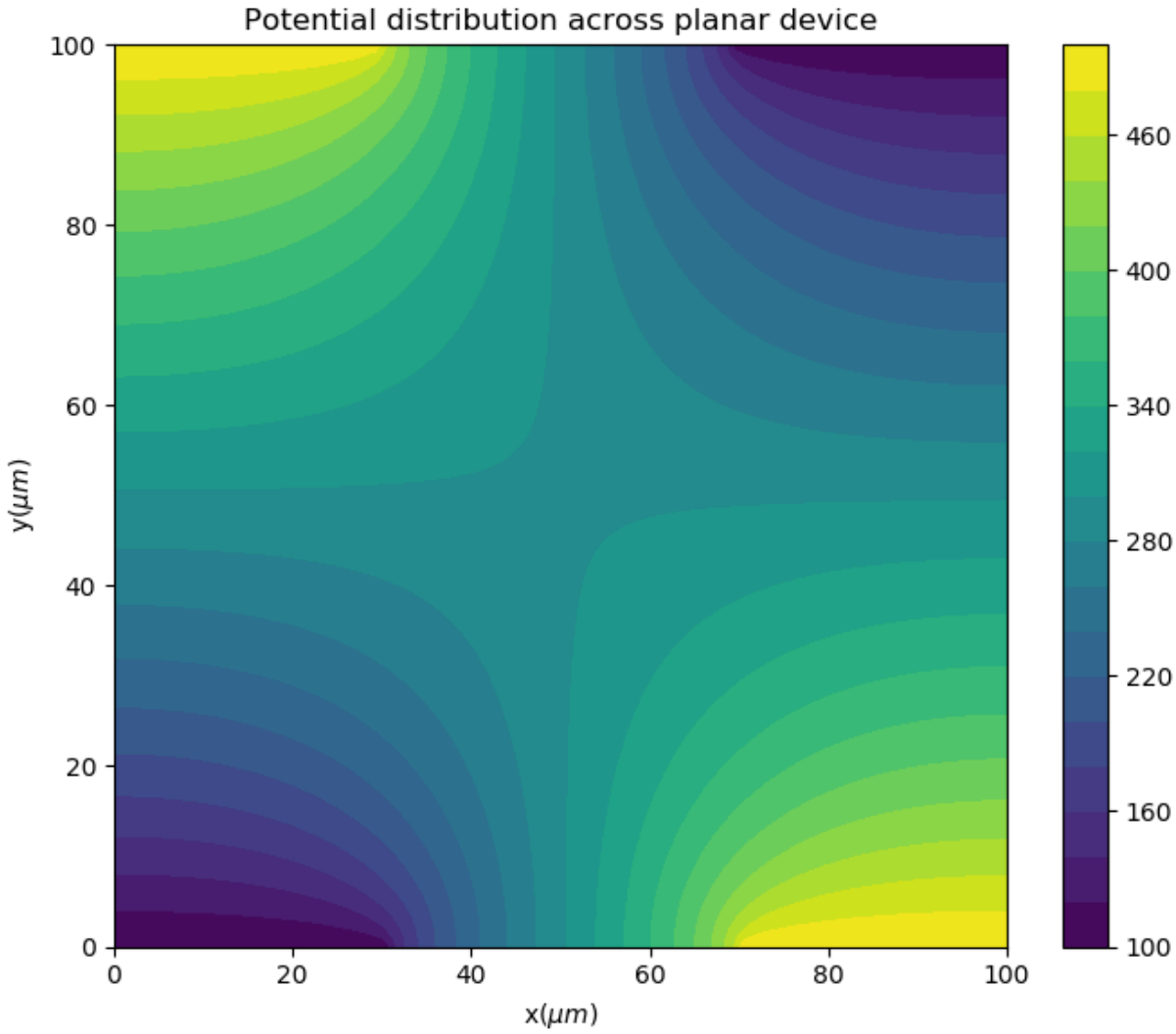
Finite Element Approximation of Poisson's Equation for Determination of Internal Voltages

Alan Burl

September 12, 2019



1 Problem Statement

The determination of the potential field in a detector leads by way of weighting potential to the determination of theoretical pulse output. The determination of this is done by solving Poisson's equation. This proves to be difficult, if not impossible, for geometries other than simple cases. To solve these geometries, a finite element approach can be taken and implemented to solve nearly any geometry. This work details the implementation of the finite approach to solve Poisson's equation to solve a three terminal device.

2 Development of Matrices and solution

Using input from the user interface, a matrix of dimension *nxm* with *n* and *m* being the number specified in the *x* and *y* directions. The spaing and placement of the nodes is done using Python3.7 NumPy module. Nodes of the geometry are broken down into boundary conditions and field conditions.

2.1 Boundary Conditions

Boundary condition locations are taken form the user interface by means of the terminal geometry widget. In this widget, the geometric location and size of the terminal is entered as well as the potential applied across the terminal. At this point, the terminal is limited to having an uniform distribution across the entire surface. These inputs are passed as arguments to the boundary condition method of Voltage Calculation class in Calculation.py. The specified voltage is used as the boundary condition and is placed assigned to its respective location in the *b* matrix. The remainder of the boundary conditions are determined by reflecting the adjacent node normal to the boundary condition. For a terminal with known potential on the right side of the geometry, an example is shown in (1).

$$\frac{\partial^2 V}{\partial x^2}+\frac{\partial^2 V}{\partial y^2}\approx\frac{(V_{x+1}-V_{xy})-(V_x-V_{x+1})}{\Delta x^2}+\frac{\left(V_{y-1}-V_{xy}\right)-\left(V_{xy}-V_{y+1}\right)}{\Delta y^2}=V_{ap} \tag{1}$$

Where V_{xy} is the boundary node being evaluated, and Δx, Δy are the distance between nodes in the *x* and *y* directions respectively. (1) is reduced to have a coefficient of one for the node in question for matrix solving, seen in (2).

$$V_{xy}-\frac{V_{x+1}\Delta y^2}{\Delta x^2+\Delta y^2}-\frac{V_{y-1}\Delta x^2}{\Delta x^2+\Delta y^2}-\frac{V_{y+1}\Delta x^2}{\Delta x^2+\Delta y^2}=\frac{-\rho\Delta x^2\Delta y^2}{2\varepsilon\left(\Delta x^2+\Delta y^2\right)} \tag{2}$$

The remainder of the boundary conditions are solved using the same method with their respective nodes.

2.2 Field Conditions

After determination of the boundary conditions is complete, the field of the geometry is calculated. All nodes in the field are determined using (3). The implementation of a space charge term, ρ, can be done in this process.

$$V_{xy}-\frac{V_{x+1}\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}-\frac{V_{x-1}\Delta y^2}{2\left(\Delta x^2+\Delta y^2\right)}-\frac{V_{y-1}\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}-\frac{V_{y+1}\Delta x^2}{2\left(\Delta x^2+\Delta y^2\right)}=\frac{-\rho\Delta x^2\Delta y^2}{2\varepsilon\left(\Delta x^2+\Delta y^2\right)} \tag{3}$$

2.3 Calculation

After all nodes in the matrix have been accounted for, the A matrix will have ones on the diagonal with other constituents on the same row. Use of Python3.7 NumPy module is used to solve for the voltage vector. This vector is returned to the user interface by means of a contour plot.

3 Code Validation

To verify the functionality and accuracy of the implemented code a simple two terminal device was used. The geometry was set to be a $100\mu m$ square with 100 nodes on each axis. The terminals were placed across the top and bottom surfaces, with the top having an applied voltage of $1000V$ and the bottom being set to ground. Solving of Poisson's equation for this geometry results in a linear relationship.

The simulated results of the two terminal device is seen in Figure 1. As can be seen the simulated results show a linear relationship between voltage and distance in y.

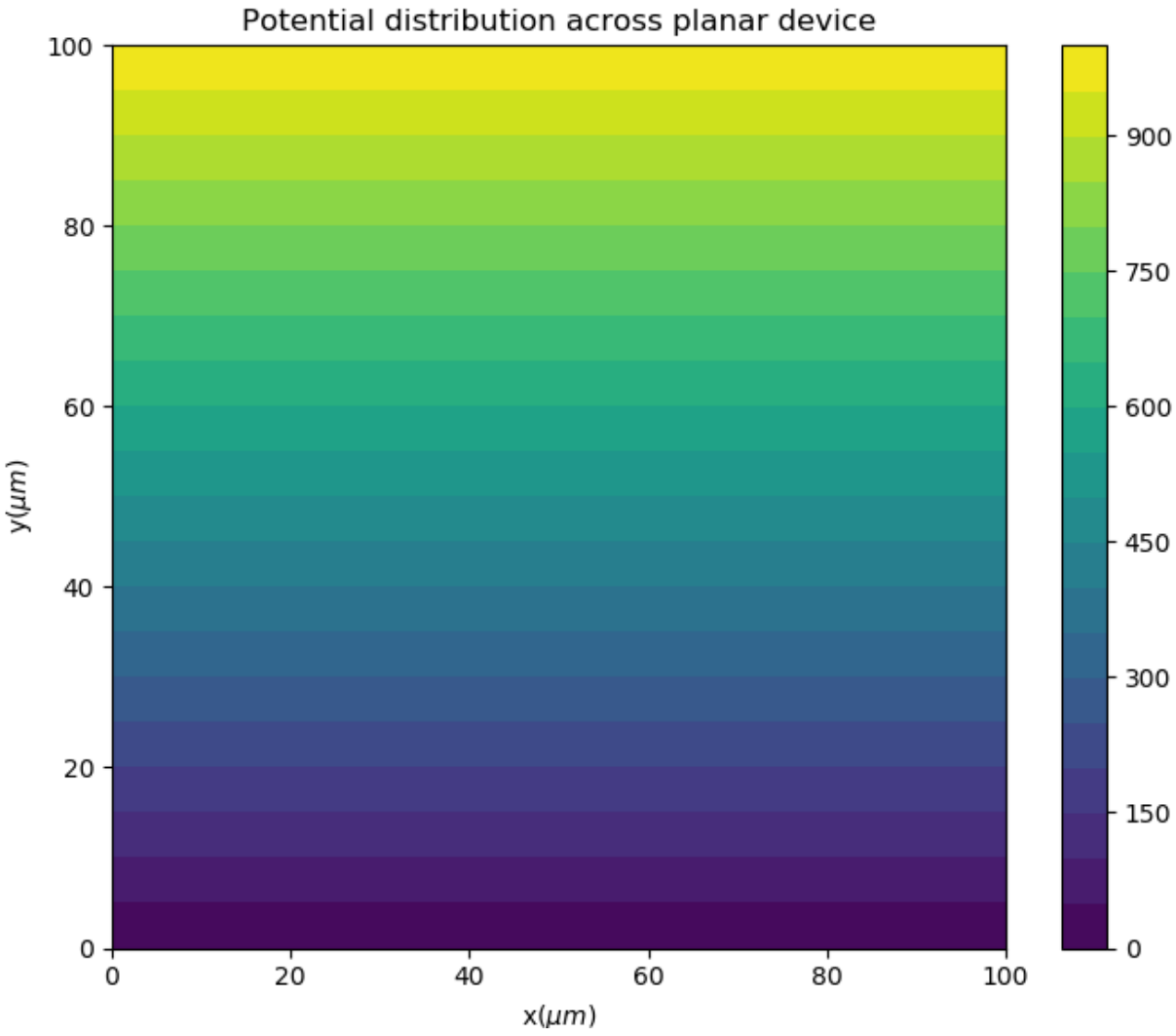


Figure 1: Simulated voltage distribution of two terminal device.

4 Results

Having verified the functionality of the code, a three terminal device was simulated. The device was $100\mu m$ square with 100 nodes an each axis. The first terminal was placed on the bottom of the plate and spanned $100\mu m$ at a potential of 30V. The second terminal was located a $y = 100\mu m$ and spanned $30\mu m$ in the postive x-direction at a potential of 100V. The last terminal was located at $y = 100\mu m$ and $x = 70\mu m$ and spanned $30\mu m$ at a potential of 0V. The resulting contour plot is seen in Figure 2.

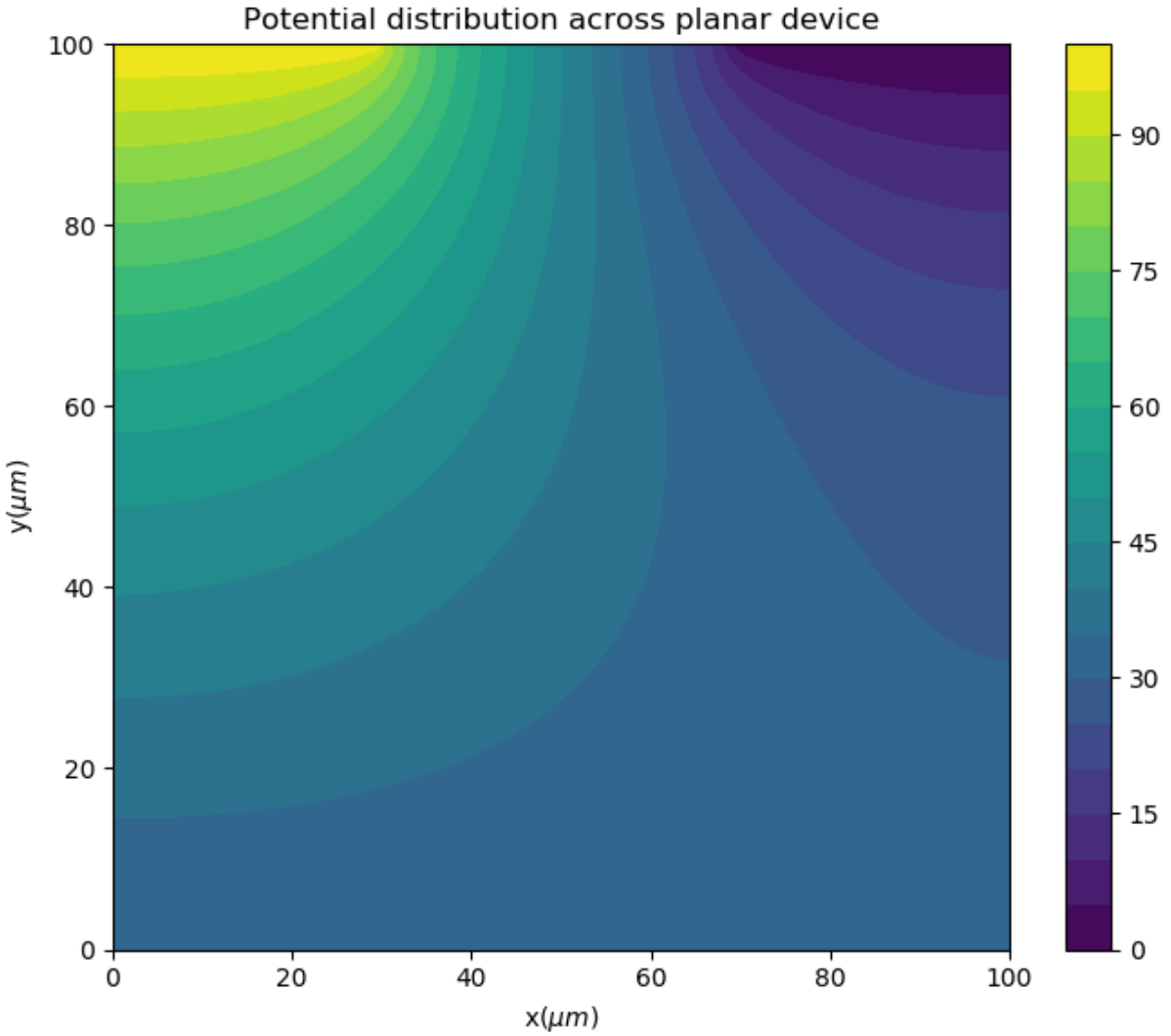


Figure 2: Simulated applied voltage of three terminal device detector geometry.

5 Code Usage

The code used in the development of this can be found on GitHub at `https://github.com/alangburl/Finite_Element_Votlage.git`. It is also included in the included zip file. The code was developed using Python3.7.1 and PyQt5. Functionality of this code is not guaranteed if deployed using older/newer software. The package needed for calculation, NumPy, is included with many popular scientific distributions of Python, including Anaconda. Launching

and execution of the code should be done via command line. Execution of this code in Spyder will result in erratic behavior.

5.1 GUI Interface

To mitigate the possibility of errant arguments into code, human interfacing is done via a graphical user interface. This interface draws the base geometry for visual verification of correct geometric entry. The geometry is draw with a scaling of $1\mu m$ is represented by $1 pixel$. The graphical user interface is seen in Figure 3. The interfacing code does allow for any value to be entered into any field, but entry of none numeric values will result in a fatal error causing the program to crash.

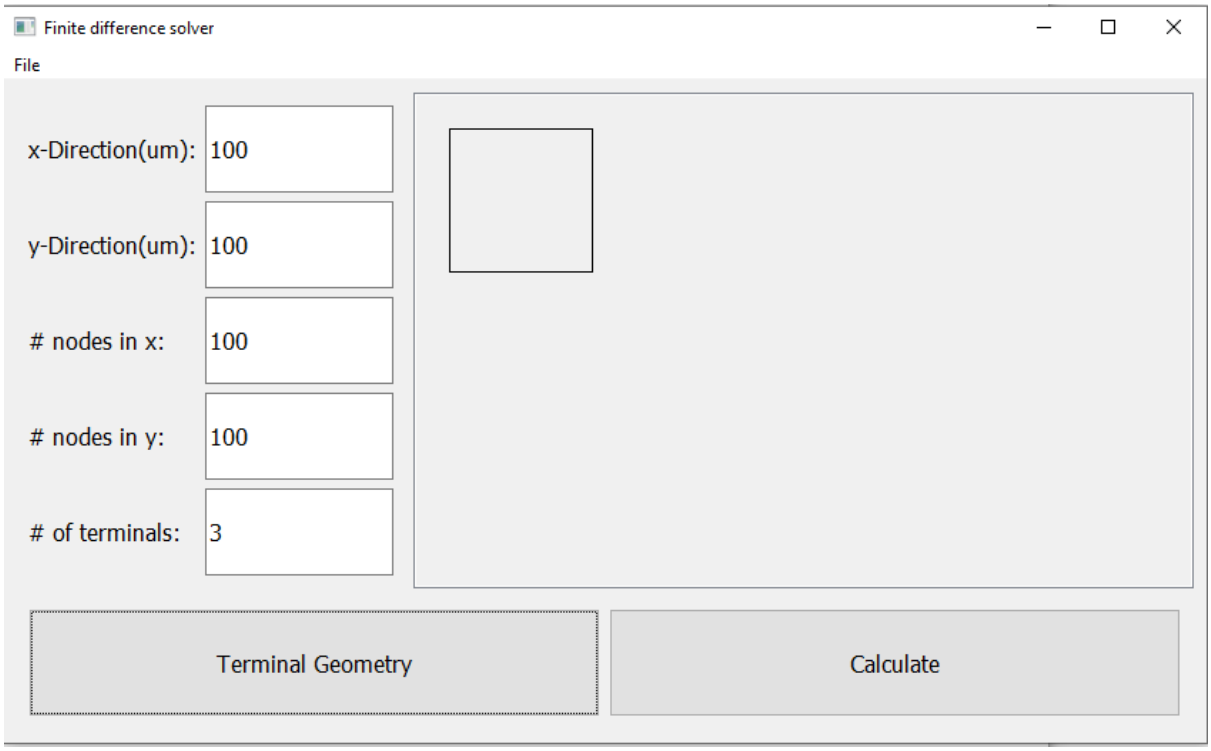


Figure 3: Graphical User Interface for human interfacing with solution