Transient Electric Field Simulation to Understand Nanoparticle Injection Via Electroporation

Ryan Lagoy
EC 456: EM II
April 22, 2012

Introduction

There has been significant research in the past for transdermal drug delivery via electroporation; however, there has been very little success with breaking the surface of the skin and injecting medicinal nanoparticles. The purpose of this research is to model the electric field in the layers of skin to examine its behavior, discover the transient behavior of the surface charge density on the boundaries between the different layers of skin, and to determine the appropriate time scale in which it takes the system takes to reach its steady state condition.

Electroporation is the biological process by which pores are formed on the top layers of skin. When an electric field interacts with the skin it is hypothesized that the skin will undergo this process if the field has a minimum strength of 400 V/cm. One leading theory for this process is that the skin porates near the hair follicles, and the skin will form various charge densities for the nanoparticles to be injected. However, it is left to much debate about what exactly occurs at the microscopic level that creates the pores in the skin. Based on prior research, there are three key factors that cause electroporation: there has to be an electric field tangential to the boundary of the skin, the magnitude of the field has to be stronger than 400 V/cm, and the lower layers of the skin should not be affected strongly by the electric field. To create the electric field, an electrode with a specific contour shape will be applied to the surface of the first layer of the skin.

Several computer software packages were evaluated for their usefulness in achieving accurate models of the fields induced into the layers of skin. The first of which is a simulation package called *PSpice*, which could be useful with modeling the one-dimensional electric field. More specifically, the famous electromagnetism problem, *Maxwell's Capacitor*, could be accurately modeled and reproduced in the *PSpice* environment. The second software package that was evaluated for the model was MATLAB. This software is versatile and not made

specifically for electromagnetism research, however, with proper knowledge and expertise it is possible to write robust, customizable, specific, and expandable software. It was decided that MATLAB would be the software package that will be used to develop the proper simulations for the electric field problem.

A very brief discussion about the capabilities of MATLAB with electric field modeling will be discussed. MATLAB is a computer language and software package made specifically with engineering in mind. Control structures such as for loops can be used for summation and product notation, which will be beneficial for the calculation of the electric fields. MATLAB also contains various libraries for graphical user interfaces and figure windows, which can be used for displaying qualitative vector fields and quantitative plots. The written code for all of the models produced for this research can be found in the appendix of the report.

A proper and robust numerical method for calculating potential and electric field had to be integrated into the model. A method known as the Charge Simulation Method (CSM) was chosen to accurately calculate electric fields from discrete points positioned along any contour shape. This numerical method, first presented by Singer, Steinberger, and Weiss in IEEE Transactions in Power Apparatus and Systems, replaces older techniques that require complex integration and solutions to Laplace's equations (1660). This numerical technique is applicable to multidimensional structures in homogenous media (1660). With the benefits of elegance and simplicity of CSM, this technique was chosen as the basis for the research. A more in depth description will follow when the two-dimensional model is described.

A brief discussion about boundary conditions will ensue. There are three key electric field boundary conditions that have been applied in the research to calculate electric fields across

dielectric boundaries. First, the potential as seen from both sides of the dielectric has to be continuous. The equation takes the form of the following in the two-dimensional case:

$$\varphi_1(x,0) = \varphi_2(x,0), \qquad (1)$$

where φ_1 is the potential seen on the side of the first dielectric and φ_2 is the potential seen on the side of the second dielectric. Secondly, the electric field parallel to the surface is continuous. The equation takes the form as the following:

$$\hat{n}\times\left(\vec{E}_1 - \vec{E}_2\right) = 0, \qquad (2)$$

where $\hat{n}$ is the orthogonal unit vector that describes the boundary, $\vec{E}_1$ is the electric field on the side of the first dielectric at the boundary, and $\vec{E}_2$ is the electric field on the side of the second dielectric at the boundary. The last boundary condition states that the difference between the electric fields multiplied by their corresponding permittivities is equal to the surface charge density. The equation takes the form as the following:

$$\hat{n}\cdot\left(\epsilon_1\vec{E}_1 - \epsilon_2\vec{E}_2\right) = \rho_s, \qquad (3)$$

where $\hat{n}$ is the orthogonal unit vector that describes the boundary, $\vec{E}_1$ is the electric field on the side of the first dielectric at the boundary, $\vec{E}_2$ is the electric field on the side of the second dielectric at the boundary, ϵ_1 and ϵ_2 are the corresponding permittivities, and ρ_s is the charge density on the boundary. These comprise the three boundary conditions that exist for electric fields across a dielectric boundary.

The calculation of the electric field of *n* charges over a conducting plate or over a ground plane requires the use of image charge theory. With this theory, image charges are reflected over the boundary that is the conducting place or ground plane. The images charges have equal but opposite magnitude to the original charges. The electric field over the space is then calculated with both the original charges and image charges through superposition. This method works

because of the idea that the conducting plate cannot contain any internal electric field. So, this follows that the electric field has to be parallel to the orthogonal unit vector which defines the conducting plane, due to boundary condition (2) and (3). When the image charge is added to the model, the field in between the two charges is perfectly perpendicular to the conductive plate. The image charge theory is essential to the calculations done in the model.

Methods of Investigation

In this section, the theory behind each model will be explained in detail, so that the analysis could be reproduced or modified if need be. The code will be explained briefly, as there are comments that explain the functions if clarification is necessary. Before discussing the numerical processes involved with this research, an analytical approach to the problem will be explained.

Maxwell's Capacitor

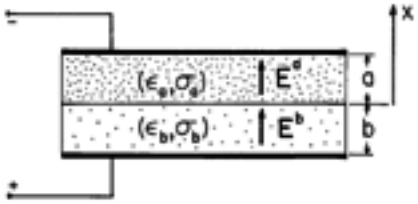

Figure 1. Maxwell's Capacitor (Haus & Melcher)

Maxwell's Capacitor is a well-know electromagnetics problem involving two capacitors in series, and each capacitor is filled with a medium defined by both permittivity and conductivity. A brief derivation of the transient electric fields, E_a and E_b, will be outlined below.

Using Faraday's Law, an equation was derived that relates the magnitudes of the field and the distances between the plates of the capacitor with the total voltage:

$$\int_{-b}^{a} E_x dx = v(t) = aE_a + bE_b \qquad (4)$$

Using (3) and $\nabla \cdot J = \frac{d\rho}{dt}$, the following equation was derived:

$$\left(\sigma_a \vec{E}_a - \sigma_b \vec{E}_b\right) = \frac{d}{dt}\hat{n} \cdot \left(\epsilon_a \vec{E}_a - \epsilon_b \vec{E}_b\right) \qquad (5)$$

When solving for E_b in (4) and substituting the result into (5), the final differential equation was derived:

$$(b\epsilon_a - a\epsilon_b)\frac{d\vec{E}_a}{dt} + (b\sigma_a + a\sigma_b)\vec{E}_a = \sigma_b v(t) + \epsilon_b \frac{dv(t)}{dt} \qquad (6)$$

This differential equation can be solved, and solutions for $\vec{E}_a$ and $\vec{E}_b$ will be derived. But first an initial condition has to be determined. At the instant when a voltage is applied, t=0, the conductivity does not play a role on the electric field distribution. Thus, only permittivites will be accounted for. The initial condition is found below:

$$\vec{E}_a(0) = \frac{\epsilon_b}{(b\epsilon_a - a\epsilon_b)}\, v(t) \qquad (7)$$

The solutions can now be found. The forced solution to the differential equation is as follows,

$$\vec{E}_{af}(t) = \frac{\sigma_b}{(b\sigma_a + a\sigma_b)}\, v(t)\,, \qquad (8)$$

and the natural solution is:

$$\vec{E}_{an}(t) = Ke^{-t/\tau}, \qquad (9)$$

where

$$\tau = \frac{(b\epsilon_a - a\epsilon_b)}{(b\sigma_a + a\sigma_b)}. \qquad (10)$$

Adding the natural and forced solutions together and solving for constant, K, based on the initial condition gives a final equation for $\vec{E}_a$ and $\vec{E}_b$ using (4):

$$\vec{E}_a(t) = \frac{\sigma_b}{(b\sigma_a + a\sigma_b)}\, v(t)\, \left(1 - e^{-\frac{t}{\tau}}\right) + \frac{\epsilon_b}{(b\epsilon_a - a\epsilon_b)} v(t)\, e^{-\frac{t}{\tau}} \qquad (11)$$

$$\vec{E}_a(t) = \frac{v(t)}{b} - \left(\frac{\sigma_b}{(b\sigma_a + a\sigma_b)}\, v(t)\, \left(1 - e^{-\frac{t}{\tau}}\right) + \frac{\epsilon_b}{(b\epsilon_a - a\epsilon_b)} v(t)\, e^{-\frac{t}{\tau}}\right)\frac{a}{b} \qquad (12)$$

These equations were then implemented into MATLAB, and plotted for various parameters. It is also significant to remember that this can be simply modeled with a resistor in parallel with a capacitor in series with another resistor in parallel with a capacitor. Figure 2 shows the output with the parameters of $\sigma_a > \sigma_b$ and $\epsilon_b > \epsilon_a$.

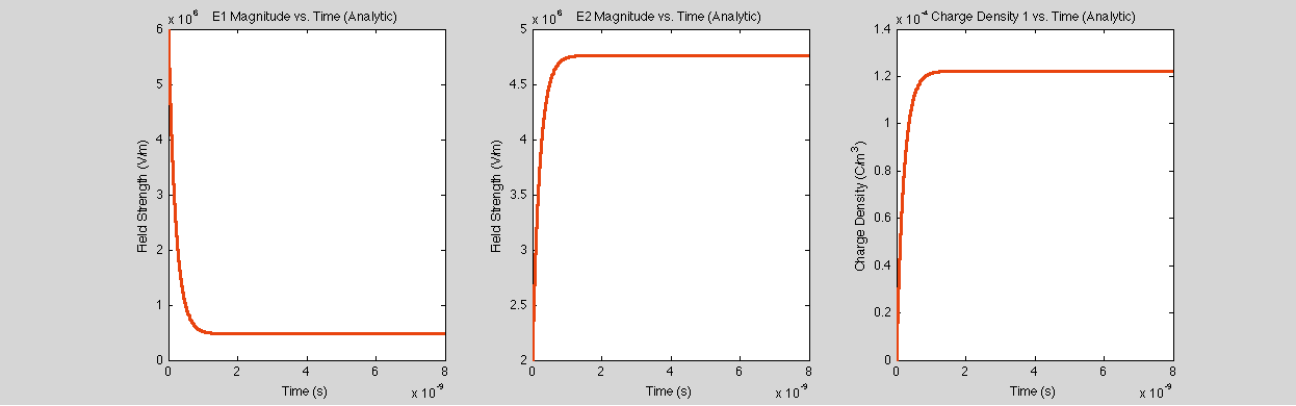

Figure 2. Analytical solution for Maxwell's Capacitor problem.

One Dimensional Modeling

The second phase of the process was to solve the Maxwell's Capacitor problem numerically using MATLAB. Also, instead of only solving for two capacitors, this model included three to represent the three different layers of skin. The purpose of this model was to produce accurate data for a one-dimensional electrode, and to show the transitory electric field and surface charge density evolution over several time constants.

To begin, a system of equations was formed to solve for the electric field in each capacitor. The system is shown below:

$$E_1 d_1 + E_2 d_2 + E_3 d_3 = v(t) \qquad (13)$$

$$-\epsilon_1 E_1 + \epsilon_2 E_2 = \rho_{s1} \qquad (14)$$

$$-\epsilon_2 E_2 + \epsilon_3 E_3 = \rho_{s2} \qquad (15)$$

This system employs the boundary conditions and the simple consequence of Faraday's Law as shown above. The base case can be calculated knowing that $\rho_{s1} = \rho_{s2} = 0$, but because the transient charge density is not known. Notice that only the base case can be calculated without employing numerical methods because the charge density over time is not known yet. Specifically, Newton's method has to be used over several time constants to determine the nature of the charge density.

Newton's method is a numerical technique used to approximate a differential equation, and this is key to solving for the surface charge on the boundary between the two layers of skin. Kirchoff's current law in the circuit sense, states that $\nabla \cdot J = 0$, meaning the current density entering a node is equal the current density leaving the same node. While this approximation holds true in numerous circuits, it does not hold true in this problem. The divergence of the current density is equal to the rate of change for the surface charge density, or $\nabla \cdot J = \frac{d\rho}{dt}$. With Newton's method, it is possible to approximate the charge density over time. Newton's method takes the form of the following:

$$\rho_s(t_{next}) = \rho_s(t_{old}) + \frac{d\rho_s}{dx}\,\Delta t \;\rightarrow \rho_{s1}(t_{next}) = \rho_{s1}(t_{old}) + \left(\sigma_2 \vec{E}_2 - \sigma_1 \vec{E}_1\right)\Delta t \qquad (16)$$

A similar equation can be derived for ρ_{s2}. To properly define this series, a base case is needed. At t=0, there is no surface charge density on the boundary. As time progresses, $\frac{d\rho_s}{dx} \rightarrow 0$ and the

charge on the boundary of the surface becomes constant. By using this method it is possible to solve the system shown by (13-15) at discrete time intervals. It is essential that $\Delta t \to 0$, for the method to be as accurate as possible.

The time constants for the system have to be calculated, which is simply ϵ/σ. In this model, the time increments by the smallest time constant divided by 100, and the method is repeated until the time reaches 5 times the largest time constant. At 5 time constants, the electric field and surface charge density will reach its steady state form, and the data can then be analyzed properly. A simple graphical user interface was created so that the user could alter the permittivities and conductivities of the skin layers. The figure window is shown below, and the MATLAB code can be found in the appendix.

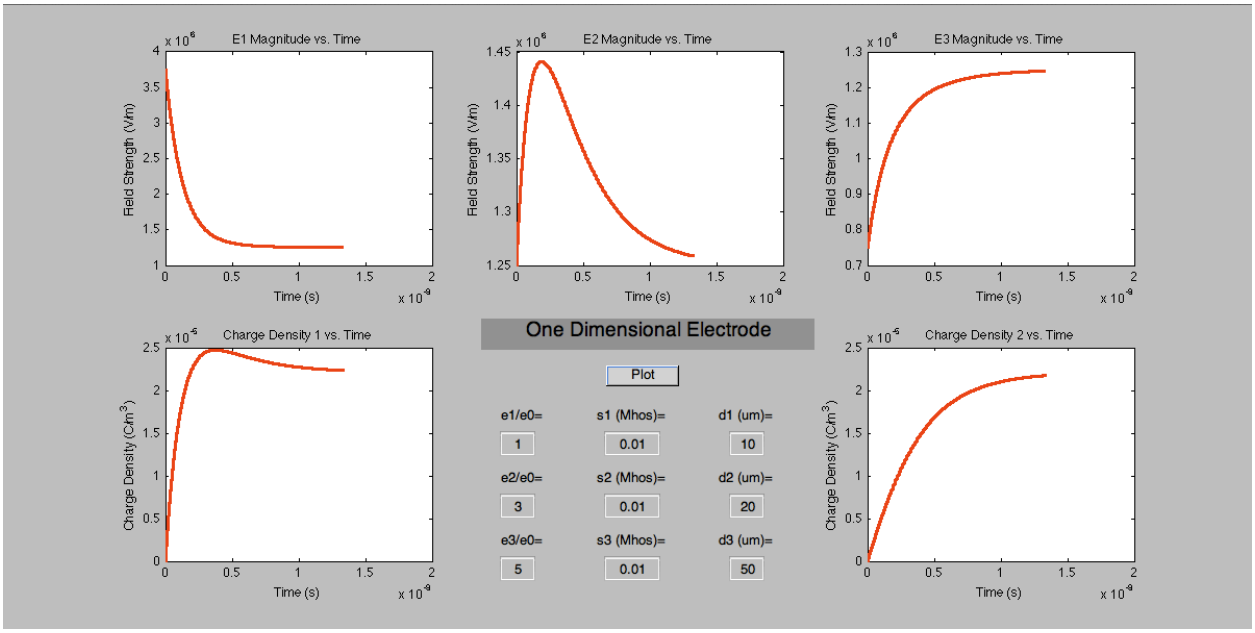

Figure 3. Graphical user interface for numerical computation of Maxwell's Capacitor with three layers.

Inaccuracies of this model include the assumption that the bottom of the third layer of skin is grounded, and the plate is infinite in size in comparison to thickness of the skin layers. The limitations of the method are that only one-dimensional electrodes can be tested. In numerous

ways, this method will not be incredible useful for the research of stimulating the skin for electroporation. Also, it is important to note that the transient nature of the electric field and surface charge density occurs at the nanosecond timescale. This method was tested for d3=0um (the thickness of the third layer), and the results match those of the analytical solution. To improve on the inaccuracies and limitations, a second simulation was created.

Two-Dimensional Model (no Conductivity)

This simulation produces a qualitative vector field based on a specific contour shape specified by the user. In this model, the top two layers of skin were modeled for simplicity and computational efficiency. In order to calculate the electric field due to the custom electrode shape a numerical method called the Charge Simulation Method had to be adopted. Because each layer has a different permittivity, another simulation technique using fictitious charges had to be integrated. CSM is first explained, and then the fictitious charge method.

CSM begins with *n* discrete charges of which are aligned along a conductor's contour. Each point then has a corresponding contour point, which are placed on the boundary between the conductor and the dielectric medium. The conductor is assumed to be perfect, and in this case, all of the charges, either positive or negative, exist on the surface. Also, there is also no electric field on the inside of a perfect conductor. The conductor is then charged to an arbitrary potential, and the following equation is formed.

$$\sum_{j=1}^{n} \frac{1}{4\pi\epsilon * r_j} * Q_j = \varphi_c, \qquad (17)$$

where r_j is the distance from the contour point, Q_j is the fictitious charge, and φ_c is the potential of the conductor. Since there are *n* contour points, and r_j and φ_c are known, a system of equations can be formed and each charge can be solved directly with the equation below.

$$[P] * [Q] = [\varphi] \rightarrow [Q] = inv[P] * [\varphi], \qquad (18)$$

where [P] is the potential coefficient matrix, [Q] is the column vector of all the charges, and $[\varphi]$ is the column of the potentials which should be the same if the conductor is perfect and continuous. Once the charges are solved, the electric field associated with *n* point charges can be calculated. The formulas that are used for the x and y components of the electric field are as follows based solely on the contour charges:

$$\vec{E}_x = \frac{dE}{dx} = \frac{1}{4\pi\epsilon}\sum_{j=1}^{n}\frac{Q_j\,\Delta x_j}{(\Delta {x_j}^2 + \Delta {y_j}^2)^{3/2}} \qquad (19)$$

$$\vec{E}_y = \frac{dE}{dy} = \frac{1}{4\pi\epsilon}\sum_{j=1}^{n}\frac{Q_j\,\Delta y_j}{(\Delta {x_j}^2 + \Delta {y_j}^2)^{3/2}} \qquad (20)$$

To see if this method is accurate, a point is chosen on the contour which does not have a contour point and checked to see how close its potential is to the potential defined in the problem. Given ~140 charges along the contour, the error is less than 1%. Next, a separate method is needed to derive the electric fields in the two layers of skin due to the different permittivities and requires the use of two sets of fictitious charges.

To account for the effects of the layers of skin with different permittivities and their effect on the electric field, fictitious charges have to be used. Shen and Kong, in *Applied Electromagnetism*, describe a solution technique for this problem (402). Before applying the concept to the current simulation, it is necessary to look at one real charge in a medium with permittivity ϵ_1 near a boundary to a separate permittivity ϵ_2. See figures below for the layout.

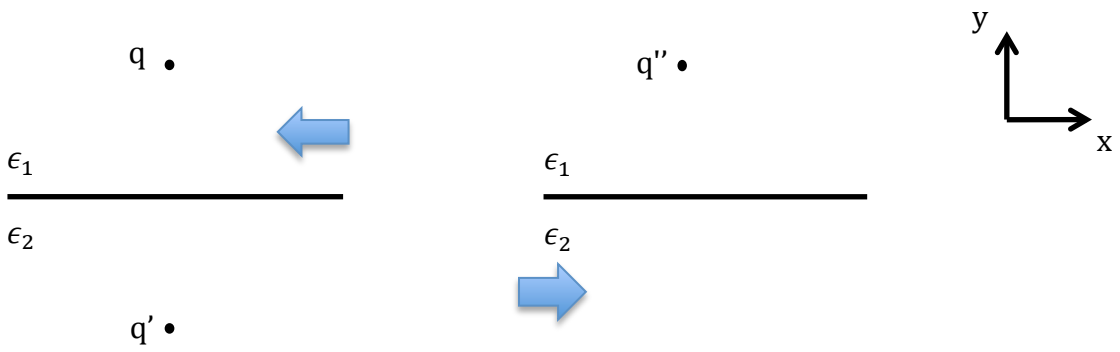

Figure 4. (left) q and q' are used to calculate the field in the medium characterized by ϵ_1. (right) q'' is used to calculate the field in the medium characterized by ϵ_2.

The potential can be calculated in the medium characterized by ϵ_1 with the real charge, q, and q' (which can be found by reflecting the position of the original charge over the boundary). The potential in the top layer is defined as:

$$\varphi_1(x,y) = \frac{1}{4\pi\epsilon_1}\left(\frac{q}{\sqrt{\Delta x^2 + (y-d)^2}} + \frac{q'}{\sqrt{\Delta x^2 + (y+d)^2}}\right), \qquad (21)$$

where d is the vertical distance between the charge and the boundary. The potential in the second layer is given with q'' which is in the same position of q, and is calculated with the following equation:

$$\varphi_2(x,y) = \frac{1}{4\pi\epsilon_2}\left(\frac{q''}{\sqrt{\Delta x^2 + (y-d)^2}}\right) \qquad (22)$$

Two boundary conditions (1) and (2) have to be satisfied at the boundary, and the equations are shown below.

$$\varphi_1(x,0) = \varphi_2(x,0) \rightarrow \frac{1}{4\pi\epsilon_1}\left(\frac{q}{\sqrt{\Delta x^2 + d^2}} + \frac{q'}{\sqrt{\Delta x^2 + d^2}}\right) = \frac{1}{4\pi\epsilon_2}\left(\frac{q''}{\sqrt{\Delta x^2 + d^2}}\right) \qquad (23)$$

$$\epsilon_2 \frac{\partial\varphi_2}{\partial y}(y=0) = \epsilon_1 \frac{\partial\varphi_1}{\partial y}(y=0) \rightarrow$$

$$\left(\frac{qd}{(\Delta x^2 + d^2)^{3/2}} + \frac{q'd}{(\Delta x^2 + d^2)^{3/2}}\right) = \left(\frac{q''d}{(\Delta x^2 + d^2)^{3/2}}\right) \qquad (24)$$

Solving for q' and q'' results in the following:

$$q' = q\left(\frac{\epsilon_1 - \epsilon_2}{\epsilon_1 + \epsilon_2}\right) \quad (25)$$

$$q'' = q\left(\frac{2\epsilon_2}{\epsilon_1 + \epsilon_2}\right) \quad (26)$$

It is possible to extrapolate and apply these results to a model with *n* points. To do so, there will be *n* q' and *n* q'' charges, all of which abide by equations (25) and (26). This is possible because of the theory of superposition. Once all image charges are defined, it is then possible to calculate the electric field in the top layer and electric field in the bottom layer by finding the negative gradient of (21) and (22). The electric field in the top layer is then defined as the following:

$$\vec{E}_x = \frac{1}{4\pi\epsilon_1}\sum_{j=1}^{n}\left(\frac{q\Delta x_n}{(\Delta {x_n}^2 + \Delta {y_n}^2)^{3/2}} + \frac{q'\Delta x_n}{(\Delta {x_n}^2 + \Delta {y_n}^2)^{3/2}}\right) \quad (27)$$

Lastly, the electric field in the bottom layer is defined as the following:

$$\vec{E}_y = \frac{1}{4\pi\epsilon_2}\sum_{j=1}^{n}\left(\frac{q''\Delta y_n}{(\Delta {x_n}^2 + \Delta {y_n}^2)^{3/2}}\right) \quad (28)$$

One output of this simulation is shown below. The parameters for this simulation are as follows: the contour is parabolic, $\epsilon_1 > \epsilon_2$, and $d_2 > d_1$. Note that the electric field lines in the top later begin to point outward as the boundary approaches, which is a consequence of $\epsilon_1 > \epsilon_2$.

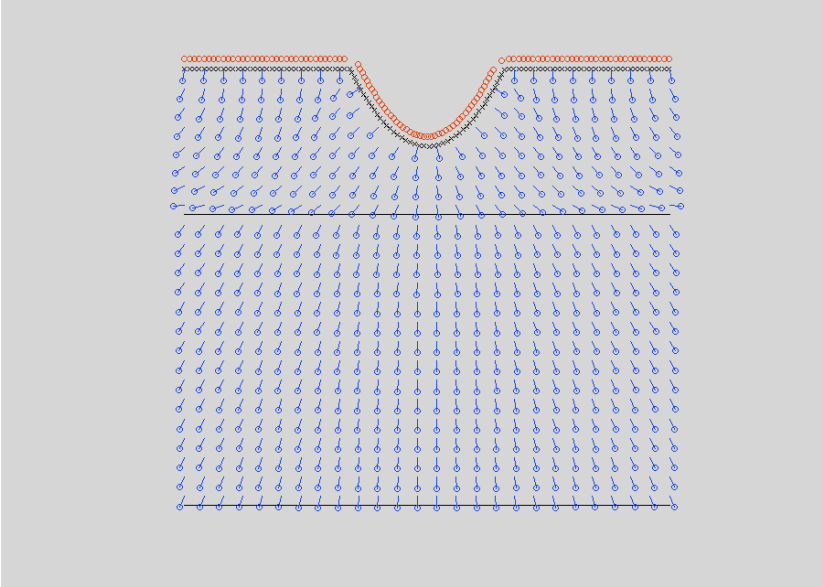

Figure 5. Electric field model for a parabolic contour.

The MATLAB program is robust in the sense that any contour shape can be modeled given its mathematical function. Note that while only the electric field lines are shown, data for each point represented is contained in a master array, which can be accessed outside of the program. Also, the MATLAB script can receive a variety of parameters, such as the skin layer depths, permittivities, and vector and charge spacing. However, this simulation does not have the capabilities for the calculation of transient electric fields. Also, since it is only a two-dimensional model the electric field magnitudes will not be as accurate as a three-dimensional simulation. To accomplish a three-dimensional simulation, ring charges and elliptical integrals will have to be calculated. Although there are inaccuracies and some limits in the model, it simulations qualitative electric field behavior.

Two-Dimensional Modeling (Conductivity)

In order to understand how to calculate the transient electric fields in this system, a new numerical technique has to be described. The time evolving electric field occurs when the medium has conductivity, and charge will build up on the boundary between the two layers of

skin. The new method is based on the theory of superposition. Its concept and theory will be discussed in this section; however the code will be omitted from the appendix.

At the initial condition when t=0, as the voltage is applied to the electrode, there will be no charge build up and the electric field solution will be that of model with no conductivity. However, the solution at the next time step requires the calculation of the surface charge density on the boundary.

The basic principle of charge accumulation on the boundary is discussed in the section of one-dimensional modeling, but the pertinent equations and boundary conditions will be covered briefly here. The charge can be approximated using Newton's method and the base case, $\rho_s(0) = 0$, as shown in (29).

$$\rho_s(t_{next}) = \rho_s(t_{old}) + \frac{d\rho_s}{dx}\,\Delta t \;\rightarrow \rho_s(t_{next}) = \rho_s(t_{old}) + \left(\sigma_2\vec{E}_{2\perp} - \sigma_1\vec{E}_{1\perp}\right)\Delta t \qquad (29)$$

Notice that the electric fields are the perpendicular components because the currents passing into and out of the boundary are of interest.

Once the surface charge is calculated, the electrode is grounded and the electric field due solely to the charges on the boundary is calculated. Image charges were included to account for the grounded electrode. After the field due to the surface charges was calculated, it was added to the original field (initial solution). This is then the electric field solution after a time step of Δt, and this is possible because of superposition. The process of obtaining the surface charges and new fields is then repeated until the time surpasses several time constants and reaches its steady state.

Below is an example of the simulation with the following parameters: the contour contains two half circles (left with positive voltage and right with negative voltage), $\epsilon_1 < \epsilon_2$,

$\sigma_1 < \sigma_2$, and $d_2 > d_1$. The time after the voltage is applied is given in the title. The simulation ran after several iterations, and two outputs are shown.

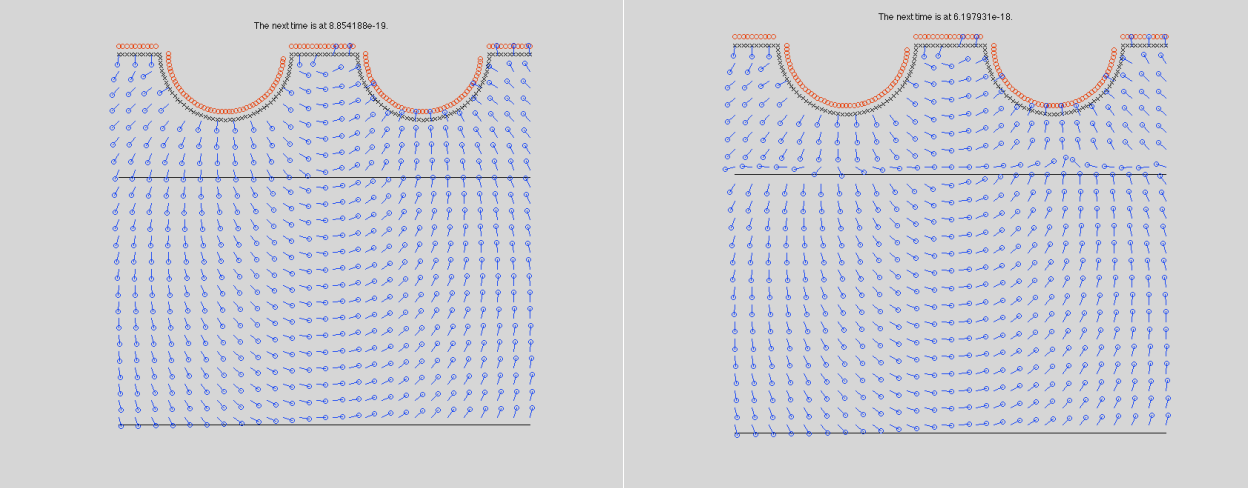

Figure 6. Time evolution of electric field. (left) The initial condition. (right) Output after several iterations.

It is also of great interest to see the distribution of surface charge density along the boundary. The plot is shown below after one iteration of the model:

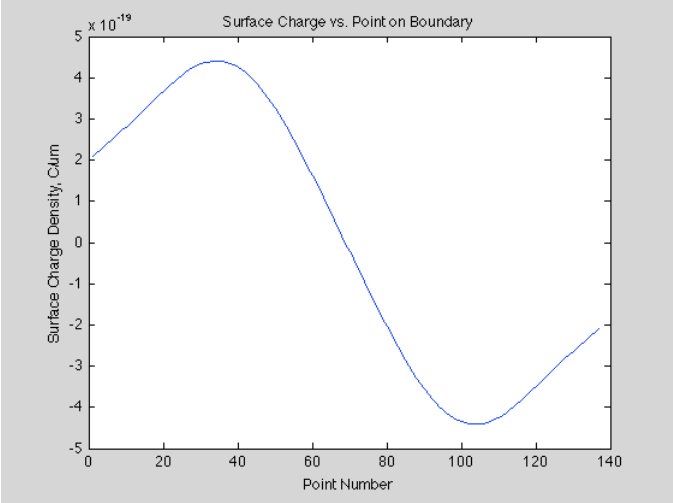

Figure 7. Surface charge density along boundary after second iteration.

It is possible to see that positive charges begin to accumulate where the positive half circle is and negative charges begin to accumulate where the negative half circle is. This model was not tested

for accuracy, but will be soon verified. One major inaccuracy with this model is the fact that it is only two-dimensional, but a fourth model will be developed to account for this.

Conclusions:

Electroporation has been studied intensively, however, there has been very little progress with understanding what actually causes this biological phenomenon. As predicted by the doctoral student, Cassandra Browning, electroporation may occur where the field is strongest and where there is a large horizontal component to the field. Based on this criterion, a design was developed making use of inter-digitated electrodes with opposite polarities. The opposite polarities, as shown in Figure 6, will form strong, horizontal electric fields. The field is strongest between the two nodes, and it is believed that there will be a high probability that electroporation will occur at that location. This is where nanoparticles with medicinal properties can be injected according to the theory.

The main reason why MATLAB was used to model this system was because of its customizability and its ability to expand. The first step forward will be to verify the transient two-dimensional electric field model in terms of behavior and units. Future models, as mentioned above will include three dimensional electrode shapes. Also, three layers will be modeled so that the electric fields can be calculated at the deeper levels of the skin. An optimal electrode design will be produced that incorporates the entire criterion necessary for electroporation.

Nanoparticle injection via electroporation has the potential of being a popular transdermal drug delivery method of the future. Many benefits include lessening the risk for infection, rapid bodily activation of the medicine, and rapid delivery. This method may play a

significant role on the battlefield to fighting full-scale epidemics in record times. It is with electromagnetic theory combined with biological understanding that could revolutionize medicinal delivery in the near future.

Works Cited

Haus, H. A., & Melcher, J. R. (1998). *Electromagnetic fields and energy*. Boston, MA: Massachusetts Institute of Technology. Retrieved from http://web.mit.edu/6.013_ book/www/

Malik, N. H. (1989). Areview of the charge simulation method and its applications. *IEEE Transactions on Electrical Insulation*, *24*(1), 3-20.

Shen, L. C., & Kong, J. A. K. (2009). *Applied electromagnetism*. (3rd Ed. ed., Vol. 1). Stamford, CT: Cengage Learning.

Singer, H., Steinbigler, H., & Weiss, P. (1973). A charge simulation method for the calculation of high voltage fields. *IEEE Transactions on Power Apparatus and Systems*, *PAS-93*(5), 1660-1668. Retrieved from http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber =4075532&tag=1

Appendix

Below is all of the code used for the MATLAB simulation software.

One-Dimensional Model

electricfield_onedimension.m

```
function electricfield_onedimension
%This function calculates and plots the transient electric field and
%charges on the boundaries between the skin. To use the program, enter the
%information in the fields and press plot.

close all

%These are the initial values and constants.
E1=[];
E2=[];
E3=[];
ps1=[0];
ps2=[0];

e0= 8.85418782*10^-12;
v=100;

%This formats the figure window.

fig = figure('Position',[130 640 1200 600],...
    'Name','One Dimensional Electrode',...
    'Menubar','none',...
    'Color',get(0,'DefaultUIControlBackgroundColor'));

%This adds a title to the figure window.
uicontrol(gcf,'Style','text', ...
    'String','One Dimensional Electrode',...
    'fontsize',15, ...
    'Position',[460,270,320,30],...
    'BackgroundColor',[.5 .5 0.5]);

%This moves the figure to the center of the window.
movegui(fig, 'center')
set(fig, 'visible', 'on')

%The code below is for the Graphical User Interface. eg. This allows you to
%type values in for the dielectric constant (en/e0), conductivity (sn), and
%thickness of the layer (dn).

%==================Layer 1================================

%====================e1/e0================================

s1han = uicontrol('Style', 'edit', 'Position', ...
    [475, 165, 40, 30], 'String', 1,...
    'Callback', @gete1);
```

```
hsttext=uicontrol('Style', 'text', ...
    'Position', [465, 195, 70, 20], 'String', 'e1/e0= ','Visible', 'on');

    function gete1(source,eventdata)
        a1=get(s1han, 'String');
        a2=str2num(a1);
        set(hsttext, 'Visible', 'on', 'String', ['e1/e0= ' num2str(a1)])
    end

%====================s1================================

s2han = uicontrol('Style', 'edit', 'Position', ...
    [575, 165, 60, 30],'String', 1/100,...
    'Callback', @gets1);

hsttext2=uicontrol('Style', 'text', ...
    'Position', [530, 195, 150, 20], 'String', 's1 (Mhos)= ','Visible',
'on');

    function gets1(source,eventdata)
        b1=get(s2han, 'String');
        b1=str2num(b1);
        set(hsttext2, 'Visible', 'on', 'String', ['s1 (Mhos)= ' num2str(b1)])
    end

%====================d1================================

s3han = uicontrol('Style', 'edit', 'Position', ...
    [695, 165, 40, 30],'String', 10,...
    'Callback', @getd1);

hsttext3=uicontrol('Style', 'text', ...
    'Position', [670, 195, 90, 20], 'String', 'd1 (um)= ','Visible', 'on');

    function getd1(source,eventdata)
        c1=get(s3han, 'String');
        c1=str2num(c1);
        set(hsttext3, 'Visible', 'on', 'String', ['d1 (um)= ' num2str(c1)])
    end

%==================Layer 2================================

%====================e2/e0================================

s4han = uicontrol('Style', 'edit', 'Position', ...
    [475, 105, 40, 30],'String', 3,...
    'Callback', @gete2);

hsttext4=uicontrol('Style', 'text', ...
    'Position', [465, 135, 70, 20], 'String', 'e2/e0= ','Visible', 'on');

    function gete2(source,eventdata)
        a2=get(s4han, 'String');
        a2=str2num(a2);
```

```
        set(hsttext4, 'Visible', 'on', 'String', ['e2/e0= ' num2str(a2)])
    end

%====================s2================================

s5han = uicontrol('Style', 'edit', 'Position', ...
    [575, 105, 60, 30],'String', 1/100,...
    'Callback', @gets2);

hsttext5=uicontrol('Style', 'text', ...
    'Position', [530, 135, 150, 20], 'String', 's2 (Mhos)= ','Visible',
'on');

    function gets2(source,eventdata)
        b2=get(s5han, 'String');
        b2=str2num(b2);
        set(hsttext5, 'Visible', 'on', 'String', ['s2 (Mhos)= ' num2str(b2)])
    end

%====================d2================================

s6han = uicontrol('Style', 'edit', 'Position', ...
    [695, 105, 40, 30],'String', 20,...
    'Callback', @getd2);

hsttext6=uicontrol('Style', 'text', ...
    'Position', [670, 135, 90, 20], 'String', 'd2 (um)= ','Visible', 'on');

    function getd2(source,eventdata)
        c2=get(s6han, 'String');
        c2=str2num(c2);
        set(hsttext6, 'Visible', 'on', 'String', ['d2 (um)= ' num2str(c2)])
    end

%===================Layer 3================================

%====================e3/e0================================

s7han = uicontrol('Style', 'edit', 'Position', ...
    [475, 45, 40, 30],'String', 5,...
    'Callback', @gete3);

hsttext7=uicontrol('Style', 'text', ...
    'Position', [465, 75, 70, 20], 'String', 'e3/e0= ','Visible', 'on');

    function gete3(source,eventdata)
        a3=get(s7han, 'String');
        a3=str2num(a3);
        set(hsttext7, 'Visible', 'on', 'String', ['e3/e0= ' num2str(a3)])
    end

%====================s3================================

s8han = uicontrol('Style', 'edit', 'Position', ...
    [575, 45, 60, 30],'String', 1/100,...
```

```
    'Callback', @gets3);

hsttext8=uicontrol('Style', 'text', ...
    'Position', [530, 75, 150, 20], 'String', 's3 (Mhos)= ','Visible', 'on');

    function gets3(source,eventdata)
        b3=get(s8han, 'String');
        b3=str2num(b3);
        set(hsttext8, 'Visible', 'on', 'String', ['s3 (Mhos)= ' num2str(b3)])
    end

%====================d3================================

s9han = uicontrol('Style', 'edit', 'Position', ...
    [695, 45, 40, 30],'String', 50,...
    'Callback', @getd3);

hsttext9=uicontrol('Style', 'text', ...
    'Position', [670, 75, 90, 20], 'String', 'd3 (um)= ','Visible', 'on');

    function getd3(source,eventdata)
        c3=get(s9han, 'String');
        c3=str2num(c3);
        set(hsttext9, 'Visible', 'on', 'String', ['d3 (um)= ' num2str(c3)])
    end
%==================Calculate Button====================

uicontrol(gcf,'Style','Pushbutton', ...
    'String','Plot',...
    'Position',[580,235,70,20],...
    'BackgroundColor',[0.8,0.8,0.8], ...
    'CallBack',@plotdata);


    function plotdata (source, eventdata)

       %This gets the coefficients from the text fields. cn is distance, bn
       %is conductivity, and an is dielectric constant.
        c3=get(s9han, 'String');
        c3=str2num(c3);
        b3=get(s8han, 'String');
        b3=str2num(b3);
        a3=get(s7han, 'String');
        a3=str2num(a3);

        c2=get(s6han, 'String');
        c2=str2num(c2);
        b2=get(s5han, 'String');
        b2=str2num(b2);
        a2=get(s4han, 'String');
        a2=str2num(a2);

        c1=get(s3han, 'String');
        c1=str2num(c1);
        b1=get(s2han, 'String');
        b1=str2num(b1);
```

```
a1=get(s1han, 'String');
a1=str2num(a1);


%Permitivities/conductivities/thickness of layer 1
e1=a1*e0;
s1=b1;
d1=c1*10^-6;

%Permitivities/conductivities/thickness of layer 2
e2=a2*e0;
s2=b2;
d2=c2*10^-6;

%Permitivities/conductivities/thickness of layer 3
e3=a3*e0;
s3=b3;
d3=c3*10^-6;



%Determine the smallest e/s ratio:

tc1=e1/s1;
tc2=e2/s2;
tc3=e3/s3;

%This is the smallest time constant divided by 100
tcsmall=min([tc1 tc2 tc3])/100;
tclarge=max([tc1 tc2 tc3]);

%Set up a time vector
timevect=0:tcsmall:tclarge*3;


%Write a loop that calculates the E field and charge densities over
%time.
length(timevect)
 for i=1:1:length(timevect)-1

     %Linear algebra, to solve 3 eqs, 3 unknowns
     emat=[[d1 d2 d3]; [-e1 e2 0]; [0 -e2 e3];];
     equal=[v; ps1(i); ps2(i);];
     ematinv=inv(emat);
     ematsol=ematinv*equal;

     %Assign values for electric field.
     E1(i)=ematsol(1);
     E2(i)=ematsol(2);
     E3(i)=ematsol(3);

     %determine the rate of charge being accumulated on the
     %boaundaries.
     dps1dt=s1*E1(i)-s2*E2(i);
     dps2dt=s2*E2(i)-s3*E3(i);
```

```
    %Use Euler's method to get a new value for the charge density
    %on the boundary.
    ps1(i+1)=ps1(i)+dps1dt*tcsmall;
    ps2(i+1)=ps2(i)+dps2dt*tcsmall;

end

%The code below plots the data that accumulated in the loop above.
subplot(2,3,1);

p1=plot(timevect(1:end-1), E1);

xlabel('Time (s)')
ylabel('Field Strength (V/m)')
title('E1 Magnitude vs. Time')
set(p1,'Color','red','LineWidth',2)

subplot(2,3,2);
p2=plot(timevect(1:end-1), E2);

xlabel('Time (s)')
ylabel('Field Strength (V/m)')
title('E2 Magnitude vs. Time')
set(p2,'Color','red','LineWidth',2)

subplot(2,3,3);
p3=plot(timevect(1:end-1), E3);

xlabel('Time (s)')
ylabel('Field Strength (V/m)')
title('E3 Magnitude vs. Time')
set(p3,'Color','red','LineWidth',2)

subplot(2,3,4);
p4=plot(timevect,ps1);

xlabel('Time (s)')
ylabel('Charge Density (C/m^3)')
title('Charge Density 1 vs. Time')
set(p4,'Color','red','LineWidth',2)

subplot(2,3,6);
p5=plot(timevect,ps2);

xlabel('Time (s)')
ylabel('Charge Density (C/m^3)')
title('Charge Density 2 vs. Time')
set(p5,'Color','red','LineWidth',2)

%Analytical Solution

tau=(d2*e1+d1*e2)/(d2*s1+d1*s2);
ea1=ones(timevect);
ea2=ones(timevect);
```

```
        charge=ones(timevect);
        for i=1:length(timevect)

            ea1(i)=s2*v/(d2*s1+d1*s2)*(1-exp(-
timevect(i)/tau))+e2*v/(d2*e1+d1*e2)*exp(-timevect(i)/tau);
            ea2(i)=v/d2-d1/d2*ea1(i);
            charge(i)=-e1*ea1(i)+e2*ea2(i);

        end

        fig=figure;

        subplot(1,3,1)
        p2=plot(timevect,ea1);
        xlabel('Time (s)')
        ylabel('Field Strength (V/m)')
        title('E1 Magnitude vs. Time (Analytic)')
        set(p2,'Color','red','LineWidth',2)

        subplot(1,3,2)
        p2=plot(timevect,ea2);
        xlabel('Time (s)')
        ylabel('Field Strength (V/m)')
        title('E2 Magnitude vs. Time (Analytic)')
        set(p2,'Color','red','LineWidth',2)

        subplot(1,3,3)
        p2=plot(timevect,charge);
        xlabel('Time (s)')
        ylabel('Charge Density (C/m^3)')
        title('Charge Density 1 vs. Time (Analytic)')
        set(p2,'Color','red','LineWidth',2)


    end

end
```

Two-Dimensional Model (No Conductivity)

EField_NonUniform_units.m

```
%Unlike the first model, this one is a script and not a function. THIS IS
%THE MAIN SCRIPT. Using the charge stimulation method, this model shows the
%field based on the eletrode configuration. The electric field is stored in
%the matrix eMat.


%These are the constants. Width is the width of the electrode, radius is
%the parameter substituted into the mathFunc (see mathFunc), spacing is the
%spacing of the contour points, efield spacing is the spacing of the
%E-field vectors. d1, d2, and d3 are the layers of the skin. Each of these
%parameters can be changed and the program will still function.
```

```
%=========================================================================
width=50;
%The radius is used with the contour, and usually adjusted based on
%aesthetics and/or criterion.
radius=8;

%Choose this radius when using the sin function in mathFunc
spacing=.5;
efieldspacing=2;
voltage=10;
displacement=1;

e0=8.85418782*10^-12/10^6;
e1=5*e0;
e2=e0;
s1=0.1;
s2=0.1;
k=1/(4*pi*e0);

d1=15;
d2=30;
%=========================================================================

%This makes the contour and charge points.
%matPoints(1,:) are the x points of the contours.
%matPoints(2,:) are the y points of the contours.
%matPoints(3,:) are the x points of the charges.
%matPoints(4,:) are the y points of the charges.
matPoints=makePoints(width,spacing,displacement,radius);


%This plots the contour and charge points.
clf
hold on
plot(matPoints(1,:),matPoints(2,:), 'xk')
plot(matPoints(3,:), matPoints(4,:), 'or')
axis equal
axis off


%This pre-allocates the matrix of potential coefficients (pMat, k/r).
%vCol is the column used for the know potential on the surface of the
%electrode.
len=length(matPoints);
pMat=zeros(len,len);
vCol1=ones(len/2,1)*voltage;
vCol2=ones(len/2+1,1)*-voltage;
vCol=[vCol1; vCol2];
vCol=ones(len,1)*voltage;

%This calculates the radius of each contour point to each charge point.
%Each row is for one contour point and the potential coefficient is
%calculated for that contour point in relation to each test charge point.
for i=1:len
    for j=1:len
```

```
        pMat(i,j)=k/(radiusFunc(matPoints(1,i), matPoints(2,i),
matPoints(3,j), matPoints(4,j)));
    end
end

%This calculates the charge of each test charge point. (CSM)
qCol=pMat\vCol;

%This tests to see if the potential at a contour point adds up to the total
%potential.
contourPointNum=1;
vtot=0;
for i=contourPointNum
    for j=1:len
        vtot=vtot+pMat(i,j)*qCol(j);
    end
end

%This preallocates the vector for the image charges
qPrimeCol=ones(len,1);
qDoublePrimeCol=ones(len,1);


%Create the image charges. This is a perfect reflection of the charges over
the
%ground plane, and the charges are negated.
%matPoints(5,:) are the x values for the image charges.
%matPoints(6,:) are the y values for the image charges.
for i=1:len
    matPoints(5,i)=matPoints(3,i);
    matPoints(6,i)=matPoints(4,i)-2*abs(-(d1)-matPoints(4,i));
    qCol(i,2)=-qCol(i,1);
    qPrimeCol(i,1)=qCol(i)*(e1-e2)/(e1+e2);
    qDoublePrimeCol(i,1)=qCol(i)*(2*e2/(e1+e2));

end


%Plots the boundaries of the skin.
plot([-width/2 width/2], [-d1 -d1],'-k')
plot([-width/2 width/2], [-d1-d2 -d1-d2], '-k')


%Make the electric field array.
%This is the preallocated matrix of electric field vectors.
eMat=zeros(round((d1+d2)/efieldspacing+1), round(width/efieldspacing+1), 6);

%These are the corresponding values for each element in the array.
%Basically, what I did was made a three dimensional array with
%corresponding values.
%eMat(:,:,1) are the x values for the efield points.
%eMat(:,:,2) are the y values for the efield points.
%eMat(:,:,3) are the x vectors for the efield.
%eMat(:,:,4) are the y vectors for the efield.
%eMat(:,:,5) are the scaled x vectors for the efield.
%eMat(:,:,6) are the scaled y vectors for the efield.
```

```
matCol=-width/2:efieldspacing:width/2;
matRow=-.1:-efieldspacing:(-(d1+d2));
maxField1=0;
maxField2=0;

%col, x
for i=1:length(matCol)
   %row, y
        for j=1:length(matRow)

            %make the x and y values of the grid correspond to the index of
            %the array.
            eMat(j,i,1)=matCol(i);
            eMat(j,i,2)=matRow(j);

            %This conditional is used so that no electric field points are
            %ploted above the electrode surface.
            if eMat(j,i,2)<mathFunc(eMat(j,i,1),radius)

                %This calculates the electric field based on the charges
                %calculated through CSM using both the charges above the
                %boundary and the images charges.
                if eMat(j,i,2)>=-d1
                [Ex Ey]=eFieldPoints(eMat(j,i,1),eMat(j,i,2),matPoints,qCol,
qPrimeCol, qDoublePrimeCol, e1, e2, 1);
                eMat(j,i,3)=Ex;
                eMat(j,i,4)=Ey;
                else
                [Ex Ey]=eFieldPoints(eMat(j,i,1),eMat(j,i,2),matPoints,qCol,
qPrimeCol, qDoublePrimeCol, e1, e2, 2);
                eMat(j,i,3)=Ex;
                eMat(j,i,4)=Ey;
                end

                %This finds the max field strength of the electric field.
                %This is not currently used in the model, but was when the
                %vectors in the vector field were of relative length.
                if eMat(j,i,2)>=-d1
                   if
(sqrt(max(max(eMat(j,i,3)))^2+max(max(eMat(j,i,4)))^2)>maxField1)

maxField1=sqrt(max(max(eMat(j,i,3)))^2+max(max(eMat(j,i,4)))^2);
                   end

                else
                   if
(sqrt(max(max(eMat(j,i,3)))^2+max(max(eMat(j,i,4)))^2)>maxField2)

maxField2=sqrt(max(max(eMat(j,i,3)))^2+max(max(eMat(j,i,4)))^2);
                   end
                end


            end

        end
```

```
end



%This loop plots the scaled vectors on the figure window for the electric
%field.

%col, x
for i=1:length(matCol)
   %row, y

        for j=1:length(matRow)

            if eMat(j,i,2)<mathFunc(eMat(j,i,1),radius)

            [xn yn]=unitVec(eMat(j,i,3),eMat(j,i,4));

            %This conditional is used for scaling and creating normalized
vectors.
            %There are different if statements if the vectors had relative
sizes.
            if eMat(j,i,2)>=-d1
                if (sqrt(eMat(j,i,3)^2+eMat(j,i,4)^2)<maxField1*0.08)

%scaleCoeff=(sqrt(eMat(j,i,3)^2+eMat(j,i,4)^2)/(maxField1*.08));
                   scaleCoeff=3.5;
                   eMat(j,i,5)=eMat(j,i,1)+scaleCoeff*xn/3;
                   eMat(j,i,6)=eMat(j,i,2)+scaleCoeff*yn/3;
                else
                   scaleCoeff=3.5;
                   eMat(j,i,5)=eMat(j,i,1)+scaleCoeff*xn/3;
                   eMat(j,i,6)=eMat(j,i,2)+scaleCoeff*yn/3;
                end

            else

                if (sqrt(eMat(j,i,3)^2+eMat(j,i,4)^2)<maxField2)

%scaleCoeff=(sqrt(eMat(j,i,3)^2+eMat(j,i,4)^2)/(maxField2));
                   scaleCoeff=3.5;
                   eMat(j,i,5)=eMat(j,i,1)+scaleCoeff*xn/3;
                   eMat(j,i,6)=eMat(j,i,2)+scaleCoeff*yn/3;
                 else
                    scaleCoeff=3.5;
                    eMat(j,i,5)=eMat(j,i,1)+scaleCoeff*xn/3;
                    eMat(j,i,6)=eMat(j,i,2)+scaleCoeff*yn/3;
                 end

            end
            %this actually plots the lines and an o at the tip.
            line([eMat(j,i,1) eMat(j,i,5)],[eMat(j,i,2) eMat(j,i,6)]);
            plot(eMat(j,i,5),eMat(j,i,6),'o')
            end

        end
```

```
end
```

eFieldPoints.m

```
%This calculates the electric field for a given point.


 function [Ex Ey]=eFieldPoints(x2,y2,matPoints,qCharge, qPrime, qPrimePrime,
e1, e2, region)

k1=1/(4*pi*e1);
k2=1/(4*pi*e2);
len=length(qCharge);
Ex=0;
Ey=0;
%scale factor for V/cm Electric Field
sf2=10^4;

for i=1:len
    x1=matPoints(3,i);  %xPoints of original charges and q''
    y1=matPoints(4,i);  %yPoints of original charges and q''
    x1p=matPoints(5,i); %xPoints of q'
    y1p=matPoints(6,i); %yPoints of q'
    q=qCharge(i,1);
    qp=qPrime(i,1);
    qpp=qPrimePrime(i,1);

    %E-field=lastE-Field+ Efield due to the test charges + Efield due to
    %the image charges for the different permittivities.
    if region==1
    Ex=Ex+k1*q*(x2-x1)/((x2-x1)^2+(y2-y1)^2)^(3/2)+k1*qp*(x2-x1p)/((x2-
x1p)^2+(y2-y1p)^2)^(3/2);
    Ey=Ey+k1*q*(y2-y1)/((x2-x1)^2+(y2-y1)^2)^(3/2)+k1*qp*(y2-y1p)/((y2-
y1p)^2+(y2-y1p)^2)^(3/2);
    else
    Ex=Ex+k2*qpp*(x2-x1)/((x2-x1)^2+(y2-y1)^2)^(3/2);
    Ey=Ey+k2*qpp*(y2-y1)/((x2-x1)^2+(y2-y1)^2)^(3/2);
    end
end


end
```

makePoints.m

```
%This function produces the contour and charge points.
function array=makePoints(width,spacing,dif,rad)

%These are the intial values
%x(1,2)
array(1,1:2)=-width/2;
%y(1,2)
array(2,1:2)=0;

%x(1,2)
array(3,1:2)=-width/2;
%y(1,2)
array(4,1:2)=dif;

%Hardcode a formula.
i=2;
for n=-width/2:.001:width/2
    %This conditional checks to see if the function in mathFunc produces a
    %value less than zero; if so, use that value instead of 0 for the y
    %value. This also calculates the charge points based on the slope of
    %the current point and the previous point.
    if (real(mathFunc(n,rad))<0 && radiusFunc(array(1,i-1), array(2, i-1), n,
mathFunc(n,rad))>=spacing)
        array(1,i)=n;
        array(2,i)=mathFunc(n,rad);
        [array(3,i-1) array(4,i-1)]=chargePoints(array(1,i-1), array(2, i-1),
array(1,i), array(2, i), dif);
        i=i+1;

    elseif (real(mathFunc(n,rad))>=0 && radiusFunc(array(1,i-1), array(2, i-
1), n, 0)>=spacing)
        array(1,i)=n;
        array(2,i)=0;
        [array(3,i-1) array(4,i-1)]=chargePoints(array(1,i-1), array(2, i-1),
array(1,i), array(2, i), dif);
        i=i+1;
    end
end

%This makes a charge point for the last contour point.
array(3,end)=array(3,end-1)+spacing;
array(4,end)=dif;

end
```

chargePoints.m

```
%This calculates the slope, and then creates a point that is perpendicular
%to the slope and offset from the first point by a distance of diff.

function [xnew ynew]=chargePoints(x1,y1,x2,y2,diff)

m=(y2-y1)/(x2-x1);
if m>0
    xnew=(x1)-diff/sqrt(1+1/m^2);
else
    xnew=(x1)+diff/sqrt(1+1/m^2);
end
ynew=(y1)+diff/sqrt(m^2+1);

end
```

mathFunc.m

```
%Enter any function using the arguments as parameters. The rad parameter is
%used to chage the graph for the boundary and contour points.
function val=mathFunc(arg, rad)

%this is a function for a round electrode shape.
%val=-sqrt(rad^2-arg^2);
val=-real(sqrt(rad^2-(arg-12)^2))-real(sqrt(rad^2-(arg+12)^2));

%This is a function for a parabolic electrode shape.
%val=(1/rad)*arg^2-8;

%This is a value for a sinusoidal electrode shape.
%val=10*sin(arg*(1/rad)-pi+1.4);

end
```

mathFunc.m

```
%This calculates the distance between two points.
function radius=radiusFunc(x1, y1, x2, y2)

radius=abs(sqrt((x1-x2)^2+(y1-y2)^2));

end
```

untiVec.m

```
%This calculates a unit vector.
function [xn yn]= unitVec(x,y)

xn=x/sqrt(x^2+y^2);
yn=y/sqrt(x^2+y^2);

end
```