# ANALYTICAL AND NUMERICAL OPTIMIZATION OF AN IMPLANTABLE VOLUME CONDUCTION ANTENNA

by

**Brian Langan Wessel**

BS, University of Pittsbugh, 2002

Submitted to the Graduate Faculty of

the School of Engineering in partial fulfillment

of the requirements for the degree of

**Master of Sciences**

University of Pittsburgh

2004

UNIVERSITY OF PITTSBURGH

SCHOOL OF ENGINEERING

This thesis was presented

by

Brian Langan Wessel

It was defended on

April 5, 2004

and approved by

# ANALYTICAL AND NUMERICAL OPTIMIZATION OF AN IMPLANTABLE VOLUME CONDUCTION ANTENNA

Brian Langan Wessel, M.S.

University of Pittsburgh, 2004

As implantable devices become increasingly sophisticated, a means of communication is required to transmit data to and from the device. A volume conduction antenna model has been developed that meets the size and power constraints of an *in vivo* environment. This thesis aims to optimize the shape, curvature, and orientation of these antennas. Analytical and numerical analysis shows that the performance is independent of the conic section used to simulate an antenna. Both analyses were also in agreement that highest curvatures achieve maximum surface potentials, and that the angle is dependent on the distance of the antenna from the surface of the head. Analytical analysis suggests that pointing the antenna elements directly at the surface may not be the optimum angle, but rather at a smaller angle. Too few data points were taken to make the same determination from the numerical case but the optimum angle does deviate from the hypothesized angle in the same way, suggesting a similar result. The numerical analysis was important as it facilitated the simulation of the epoxy between the antenna elements. Incorporating epoxy into the simulation showed 30-35% increases in surface potential. A reflective sheet was then added showing further increases in surface potential.

## PREFACE

I would like to extend my sincere thanks to Dr. Sclabassi and Dr. Sun for their guidance, positive support, and continual attention to my growth and development as a student and researcher. They were always available to talk and spend as much time as I needed to make things clear.

I also thank the students in the Laboratory for Computational Neuroscience for listening to my many and diverse questions and willingness to lend-a-hand.

Special thanks to my family: Mom, Dad, Frannie, Heather, Brenda and John, Heidi and Tommy, Grandma and Grandpa, and everyone else for believing in me and always having words of support.

Last, but not least, I would like to thank my thesis committee for spending their time to read and listen to my thesis defense and offer constructive criticism to help stengthen the research.

## TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# 1.0 INTRODUCTION

## 1.1 MOTIVATION

In 1990 there were more than half a million people suffering epileptic seizures despite appropriate pharmacotherapy [2], and it has been estimated that 100,000 to 200,000 of these are potential surgical candidates [3]. For many of these people, their quality of life is severely diminished by the afflictions of their disease. Surgery has become an effective tool in ameliorating severe symptoms expressed by patients [4], however it can be difficult to ascertain the exact foci, or region of cells responsible for the epileptic expression. Operating procedures normally involve destruction of the suspect epileptic region. However, it would be beneficial to be able to control the epileptic region without destroying cells and/or monitor the region to better locate the foci. An implantable device could accomplish both of these tasks. Thus the device could be used in both long- and short-term settings.

Establishing volume conduction as a communication link would also benefit those with spinal cord injuries. This could address the problem of the means by which activity in the motor cortex would be coupled to muscle and how the sensory cortex could perceive the surroundings. In general, the device would be capable of stimulating any part of the brain. This feature could lead to next-generation devices capable of deep brain stimulation for patients with Parkinson's disease.

Thus, a need exists for bidirectional data communication between implantable and external devices, unconstrained by wires traversing the skin. Wires pose a problem as they can create possibilities of infection or they may break. Computational advantages arise as there is a nearly infinite amount of computing power outside of the body versus a limited amount of such power inside due to the constraints of available space and energy. Peripheral comput-

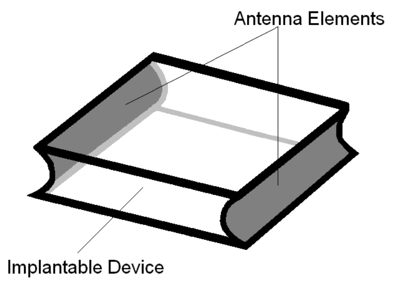


Figure 1: Antenna elements shown on the sides of the implantable device.

ing power can be concerned with problems of signal processing and other computationally intensive procedures, whereas an *in vivo* device(s) can perform very specialized tasks thus minimizing its size, weight, and energy consumption while greatly enhancing performance

Researchers have previously been successful in modeling these tissues as a volume conductor for the study of bioelectric events. We apply these models to design an efficient volume conduction antenna to both send and receive signals. Preliminary experimental results [5] have shown that two concave-shaped antenna elements located on the sides of an implantable device (Figure 1) is an efficient design. Further, it is hypothesized that the shape of the antenna elements will significantly affect how the current is distributed in the near- and far-field, as is the case in RF designs. However, volume conduction does differ from RF in the following ways [5]:

1. The strong shielding effect of ions in biological tissue is no longer a problem, instead, these ions are now employed as information carriers.
2. The electronic circuit associated with the communication system is simple and does not involve bulky components, allowing an aggressive reduction in size and weight.
3. The system does not require signal conversions to/from RF, increasing energy-efficiency.

Thus, volume conduction design differs from normal antenna design as it does not use RF energy but rather, volume currents. In RF design, the size of the antenna is dependent on the frequency of the signal. Further, to work in the frequency range having low attenuation, the frequency should be below 10 kHz [23]. This results in an unreasonably large antenna size, on the order of $10^4$ m.

## 1.2 THEORY

### 1.2.1 Volume Conductor

Classical electrical engineering circuit design involves the analysis of networks of discrete components such as resistors, capacitors, inductors and sources. The investigation of these components led to constitutive relations thereby allowing the engineer to predict values in the circuit. However, the human body defies this discretization. The conducting medium can be thought of as a 3-dimensional distributed circuit. This circuit contains no inductance but rather distributed resistance, capacitance, and sources [7].

Although the volume conductor is a distributed circuit, we must still ascertain and verify important material properties. First, the assumption of linearity is imposed. As a consequence, the rules of superposition and multiplication must hold. This is important so that the combined effects of current sources and sinks can be scaled and superposed. Although in reality, the properties of human tissue change depending on direction, these anisotropies are small for the tissue in the head, furthermore, in [7] only muscle tissue is cited as having significant anisotropies. Table 1 lists resistivity values for certain tissues.

Table 1: Resistivities of relevant material properties ( [7] compiled biological material)

| **Material** | **Resistivity** $\Omega \cdot m$ | **References** |
|---|---|---|
| brain | 5.8 | average of [9] and [8] |
| cerebral spinal fluid | 0.7 | [9] |
| bone | 177 | [8] |
| scalp | 2.22 | [8] |
| epoxy (glass) | $1{\times}10^{12}$ | [1] |
| metal (copper) | $1{\times}10^{-8}$ | [1] |

There are also ratios defined by [22] that specify the size of the brain, CSF, skull, and scalp with respect to the radius of the head. The brain surface is 88.24%, CSF is 92.94%, skull is 95.29% and scalp is 100% of the radius. Note that a radius of 7.5 cm was implemented.

Table 2: Relevant Assumptions for Human Tissue [10]

| **Condition** | **Criteria** |
|---|---|
| Neglect Propagation Effects | $kR_{max} \ll 1$ |
| Neglect Capacitive Effects | $\omega\epsilon/\sigma \ll 1$ |
| Set $E_{1n} = 0$ | $\omega\epsilon_0/\sigma_1$ |

**1.2.1.1 Quasi-static Assumption** Unlike in a discrete circuit, Kirchoff's laws cannot be applied here to solve a distributed circuit. Thus, one must solve partial differential equations for the resultant fields. Taking into account capacitance and propagation of the signal, one must solve the inhomogeneous Helmholtz equation to arrive at

$$\Phi(\vec{r}) = \frac{1}{4\pi\sigma(1+\frac{j\omega\epsilon}{\sigma})}\int_{V'} \frac{\gamma'(\vec{r}')e^{-jkR}}{R} dV' \quad , \tag{1.1}$$

where $R = |\vec{r} - \vec{r}'|$, $\gamma'$ is the volume current source density, $\sigma$ is the conductivity, and $k$ is a complex dielectric constant. To simplify this equation, a quasi-static assumption can be made which assumes that at each instant in time the potential field satisfies Poisson's equation, and that the boundary conditions are those which would exist if the source condition were stationary [10]. Plonsey showed that the assumptions in Table 2 are valid for frequencies below 1 kHz for biological material. Using these conditions, taking the Laplacian of (1.1), and noting $\nabla^2(1/R) = -4\pi\delta(\vec{r} - \vec{r}')$, one arrives at Poisson's equation,

$$\nabla^2\Phi(\vec{r}) = -\frac{\gamma(\vec{r})}{\sigma} \quad . \tag{1.2}$$

The boundary condition at the surface of the head must also be the same as that for the static case,

$$\sigma_1(1+\frac{j\omega\epsilon_1}{\sigma_1})E_{1n} = \sigma_2(1+\frac{j\omega\epsilon_2}{\sigma_2})E_{2n} \quad , \tag{1.3}$$

where $\sigma_1$ and $\sigma_2$ are the inner and outer conductivites, respectively, and $E_{1n}$ and $E_{2n}$ are the inner and outer electric fields, respectively. Thus, for $\sigma_2 = 0$ and $\epsilon_2 = \epsilon_0$ the equation reduces to (1.4), where from the third entry in Table 2, one can arrive at the conclusion that $E_{1n} = 0$, which is exactly the same as the static case.

$$\sigma_1(1 + \frac{j\omega\epsilon_1}{\sigma_1})E_{1n} = j\omega\epsilon_0 E_{2n} \tag{1.4}$$

#### 1.2.1.2 Attenuation

In order to maintain a suitable SNR, the frequency range should have low attenuation. Lindsey has shown that the frequency range below 10 kHz has constant attenuation of about 54 dB for a current of 1 mA. Frequencies above 10 kHz exhibit an increase of 2 dB/decade to 100 kHz [23]. These experiments were performed in the knee so more work must be done to characterize attenuation in the head.

#### 1.2.1.3 Half-cell Assumptions

Unlike a classical circuit, the conductance in a volume conductor arises due to ionic currents rather than electronic ones. Therefore, the antenna elements also play the role of transduction because the current in the antenna is carried by electrons whereas the current carried by the tissue is ions. In the case of the antenna, there will be currents flowing to and from the surface of the electrode. However, the half-cell potential, describing the transduction of electrons to ions, is calculated for a zero-current situation. Thus since the antenna violates the zero-current assumption, the half-cell potential must be modified by three terms to properly predict the behavior of the antenna element. These correction terms are called the ohmic, concentration, and activation overpotentials and describe the differences seen between the equilibrium zero-current and the observed half-cell potential when the current is nonzero [11]. It will be assumed that these effects cancel as a negative effect on one electrode will be a positive effect on the other [6].

### 1.2.2 The Finite Element Method (FEM)

The FEM can be summarized in the following statement: Project the weak (variational) form of the differential equation onto a finite-dimensional function space [18]. The FEM is useful for solving partial differential equations (PDE). A PDE is an equation involving a function and its partial derivatives.

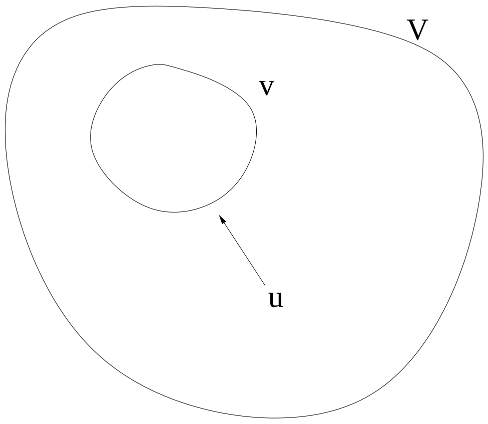


Figure 2: Set theory description

The derivation will be shown for three-dimensions but is appropriate for any number of dimensions. It is easiest to understand this statement by working through an example using the elliptic PDE, shown in the following equation:

$$-\vec{\nabla} \cdot (\sigma \vec{\nabla} u) + au = f \qquad in \ \Omega \qquad , \tag{1.5}$$

where $u$ is a scalar field,$a$ is a constant, $\sigma$ is the conductivity, which in general can be a matrix, and $f$ is a source term. Note that when $a$ is set equal to 0, (1.5) becomes the Poisson equation, (1.2). The first step is to project (1.5) onto a subspace of $V$, where it is contained. This subspace, of dimension $N$, also lies in $V$. This procedure is illustrated in Figure 2. Mathematically, this is shown in the following equation (please note that $d^3x$ is a volume integral, $d^2x$ is a surface integral, and $dx$ is a line integral):

$$\int_\Omega (-\sigma \nabla^2 \hat{u}) v + a\hat{u}v \ d^3x = \int_\Omega fvd^3x \qquad , \tag{1.6}$$

where $\hat{u}$ is an approximation to the solution, $u$. Green's first identity, shown in the following equation, is then applied:

$$\int_\Omega (v\nabla^2 \hat{u} + \vec{\nabla} v \vec{\nabla} \hat{u}) d^3x = \oint_{d\Omega} v \frac{\partial \hat{u}}{\partial n} d^2x \qquad . \tag{1.7}$$

Equation (1.6) becomes

$$\int_\Omega (\sigma \vec{\nabla} \hat{u}) \cdot \vec{\nabla} v + a\hat{u}v \; d^3x - \oint_{d\Omega} \sigma \frac{\partial \hat{u}}{\partial n} v \; d^2x = \int_\Omega f v d^3x \qquad . \tag{1.8}$$

The following boundary condition is then applied:

$$\sigma \frac{\partial \hat{u}}{\partial n} + q\hat{u} = g \qquad , \tag{1.9}$$

where when q and g are equal to zero is equivalent to writing:

$$\frac{\partial u}{\partial r} = 0 \Bigg|_{r=R} \qquad . \tag{1.10}$$

In this case, Neumann boundary conditions were used on the outer boundary of the geometry. In general, Neumann conditions are always used on the inner surfaces. The result of substituting (1.9) into (1.8) is the following:

$$\int_\Omega (\sigma \vec{\nabla} \hat{u}) \cdot \vec{\nabla} v + a\hat{u}v \; d^3x + \oint_{d\Omega} (q\hat{u} - g) v \; d^2x = \int_\Omega f v d^3x \qquad . \tag{1.11}$$

The original problem can thus be rephrased as the following: Find $u$ such that

$$\left( \int_\Omega (c \vec{\nabla} \hat{u}) \cdot \vec{\nabla} v + a\hat{u}v - f v \; dx - \oint_{d\Omega} (-q\hat{u} + g) v \; ds \right) = 0 \qquad \forall v \qquad . \tag{1.12}$$

Equation (1.12) is called the variational or weak form of the differential equation. As previously mentioned, $u$ and $v$ belong to some function space $V$. The next step is to determine a finite-dimensional subspace $v_N \subset V$. To project the weak form of the differential equation onto a finite-dimensional function space means requesting $u$ and $v$ to lie in $v_N$. Since (1.12) is true for all $v$, it is useful to define $N$ basis functions that span $v$, thus $\phi_i \in v_N$. One can also expand $u$ in the same basis:

$$\hat{u}(x) = \sum_{j=1}^{N} U_j \phi_j(x) \qquad . \tag{1.13}$$

This is known as Galerkin's method. Substituting for $\hat{u}$ and $v$ one gets

$$\sum_{j=1}^{N}\left(\int_{\Omega}(\sigma\vec{\nabla}\phi_j)\cdot\vec{\nabla}\phi_i+a\phi_j\phi_i\ d^3x+\oint_{d\Omega}q\phi_j\phi_i\ d^2x\right)U_j=\int_{\Omega}f\phi_i\ d^3x+\oint_{d\Omega}g\phi_i\ d^2x \qquad i,...,N \qquad . \tag{1.14}$$

Using the following notations:

$$K_{i,j}=\int_{\Omega}(\sigma\vec{\nabla}\phi_j)\cdot\vec{\nabla}\phi_j\ d^3x \qquad , \tag{1.15}$$

$$M_{i,j}=\int_{\Omega}a\phi_j\phi_i\ d^3x \qquad , \tag{1.16}$$

$$Q_{i,j}=\int_{d\Omega}q\phi_j\phi_i\ d^2x \qquad , \tag{1.17}$$

$$F_i=\int_{\Omega}f\phi_i\ d^3x \qquad , and \tag{1.18}$$

$$G_i=\int_{d\Omega}g\phi_i\ d^2x \qquad . \tag{1.19}$$

Equation (1.14) can be written in a compact matrix form:

$$(\overset{\leftrightarrow}{K}+\overset{\leftrightarrow}{M}+\overset{\leftrightarrow}{Q})\vec{U}=\vec{F}+\vec{G} \qquad . \tag{1.20}$$

Usually it is not necessary to distinguish between $\overset{\leftrightarrow}{K}$, $\overset{\leftrightarrow}{M}$, and $\overset{\leftrightarrow}{Q}$ or $\vec{F}$ and $\vec{G}$ so (1.20) can be written as:

$$\overset{\leftrightarrow}{K}\vec{U}=\vec{F} \qquad . \tag{1.21}$$

### 1.2.3 The Boundary Element Method (BEM)

The boundary element method is similar to the FEM, however it seeks to find a relation involving only the surface distributions of the unknown function $u$, $\frac{\partial u}{\partial n}$ and $u$ evaluated at a point P on the surface. Volume points can then be reconstructed if needed as a postprocessing step. Much of the material describing the BEM was adapted from [19].

Before one applies the BEM to a problem they must first find a fundamental solution, which becomes the weighting function, and plays a similar role as a particular solution in differential equations. It is also commonly called the freespace Green's function. This function can also be thought of as the response a system has to a $\delta$-function input.

Equation (1.2) will be solved with $\gamma$ set equal to zero *i.e.*, a fundamental solution of Laplace's equation will be found. The solution will be valid in $\Re^3$. The fundamental solution of the Laplace equation is a solution of:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2} + \delta(\xi - x, \eta - y, \gamma - z) = 0 \qquad . \tag{1.22}$$

The method is to try and find a solution to $\nabla^2 u = 0$ in $\Re^3$ which contains a singularity at the point $(\xi, \eta, \gamma)$. It is expected that the solution is symmetric around the point $(\xi, \eta, \gamma)$ since $\delta(\xi - x, \eta - y, \gamma - z)$ is symmetric about this point. A local spherical coordinate system is adopted about the singular point $(\xi, \eta, \gamma)$. Letting $r = \sqrt{(\xi - x)^2 + (\eta - y)^2 + (\gamma - z)^2}$, we have

$$\nabla^2 u = \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 \frac{\partial u}{\partial r}) + \frac{1}{r^2 sin\theta}\frac{\partial}{\partial \theta}(sin\theta \frac{\partial u}{\partial \theta}) + \frac{1}{r^2 sin\theta}\frac{\partial^2 \psi}{\partial \phi^2} \qquad . \tag{1.23}$$

Note that the second and third terms are equal to zero due to symmetry, therefore $\frac{1}{r^2}\frac{\partial}{\partial r}(r^2 \frac{\partial u}{\partial r}) = 0$. This equation can be solved by simple integration:

$$\int \frac{1}{r^2}\frac{\partial}{\partial r}(r^2 \frac{\partial u}{\partial r}) r^2 dr = 0 \qquad . \tag{1.24}$$

Canceling the $r^2$ variables, and applying the Fundamental Theorem of Calculus i.e., $(\int g'(x)dx = g(x) + D)$ with an arbitrary constant $D$:

$$r^2 \frac{\partial u}{\partial r} + D = 0 \qquad , \tag{1.25}$$

and redefining $-D$ as $A$ and integrating:

$$\int \partial u = \int \frac{A}{r^2} \partial r \qquad , \tag{1.26}$$

where an arbitrary constant $B$ is used:

$$u = \frac{A}{r} + B \qquad . \tag{1.27}$$

We must now find the constants A and B. To do this we can make use of the integral property of the $\delta$-function. From (1.22) we must have

$$\int_\Omega \nabla^2 u = -\int_\Omega \delta d\Omega = -1 \text{ where } \Omega \text{ is any domain containing } r{=}0 \qquad . \tag{1.28}$$

It is wise to choose a simple domain to allow us to evaluate the integrals. If $\Omega$ is a small sphere of radius $\epsilon > 0$ centered at $r$=0, then according to (1.7), with $\Phi$=1

$$\int_\Omega \nabla^2 u = \int_{\partial\Omega} \frac{\partial u}{\partial n} d^2x \qquad , \tag{1.29}$$

where $\partial\Omega$ is a surface integral. Furthermore, $n$ and $r$ are in the same direction, so

$$\int_\Omega \nabla^2 u = \int_{\partial\Omega} \frac{\partial u}{\partial r} d^2x \qquad . \tag{1.30}$$

Integrating around the perimeter, where $\frac{\partial u}{\partial r} = -\frac{A}{r^2}$, we get

$$\int_0^{2\pi}\int_0^{\pi} -\frac{A}{r^2}(r^2 sin\theta d\theta d\phi) = -4\pi A \qquad . \tag{1.31}$$

Using (1.28), $A = \frac{1}{4\pi}$. So we have

$$u = \frac{1}{4\pi r} + B \qquad . \tag{1.32}$$

where $B$ is arbitrary but usually set equal to zero. The foundation is now laid to develop the boundary element equation. The basic steps are quite similar to those used in the FEM. We begin with Green's second theorem:

$$\int_\Omega (v\nabla^2 u - u\nabla^2 v) d^3x = \oint_{d\Omega} [v\frac{\partial u}{\partial n} - u\frac{\partial v}{\partial n}] d^2x \qquad , \tag{1.33}$$

where $\nabla^2 u = 0$. For the Galerkin FEM we chose $v$, the weighting function, to be $\phi_j$, one of the basis functions used to approximate $u$. For the BEM we choose $v$ to be the fundamental solution of Laplace's equation derived in (1.32) i.e., $v = \frac{1}{4\pi r}$.

Then using the property of the Dirac delta:

$$\int_{\Omega}(u\nabla^2 v)d^3x = -\int_{\Omega} u\delta(\xi - x, \eta - y, \gamma - z)d^3x = -u(\xi,\eta,\gamma) \qquad (\xi,\eta,\gamma)\in\Omega \tag{1.34}$$

i.e., the domain integral has been replaced by a point value. Thus (1.33) becomes

$$u(\xi,\eta,\gamma) + \oint_{d\Omega} u\frac{\partial v}{\partial n}d^2x = \oint_{d\Omega} v\frac{\partial u}{\partial n}d^2x \qquad (\xi,\eta,\gamma)\in\Omega \qquad . \tag{1.35}$$

This equation contains only boundary integrals (and no domain integrals as in the FEM). It relates the value of $u$ at some point inside the solution domain to integral expressions involving $u$ and $\frac{\partial u}{\partial n}$ over the boundary of the solution domain. It is usually more helpful to have an expression relating the value of $u$ at some point on the boundary to boundary integrals. For brevity, the result will not be derived but is as follows:

$$c(P)u(P) + \oint_{d\Omega} u\frac{\partial v}{\partial n}d^2x = \oint_{d\Omega} v\frac{\partial u}{\partial n}d^2x \qquad (\xi,\eta,\gamma)\in\Omega \tag{1.36}$$

where:

$$c(P) = \begin{cases} 1 & \text{if P} \in \Omega, \\ \frac{1}{2} & \text{if P} \in d\Omega \text{ and } d\Omega \text{ smooth at P}, \\ \frac{\textit{inner solid angle}}{4\pi} & \text{if P} \in d\Omega \text{ and } d\Omega \text{ not smooth at P}. \end{cases} \tag{1.37}$$

Once the surface distributions of $u$ and $\frac{\partial u}{\partial n}$ are known, the value of $u$ at any point P inside $\Omega$ can be found since all surface integrals in (1.36) are known. Thus we solve for the boundary data first, and find the volume data as a separate step.

Since (1.36) contains only surface integrals, as opposed to volume integrals in a finite element formulation, the overall size of the problem has been reduced by one dimension. This can result in huge savings for a problem with a large volume to surface ratio. Also the effort required to produce a volume mesh of a complex three-dimensional object is far greater than that required to produce a mesh of the surface. Thus the BEM offers distinct advantages over the FEM in certain situations. There are also disadvantages, and both will be discussed in the next section.

### 1.2.4 Comparison of the FEM and BEM

For complex geometries, the forward problem is typically solved by the BEM or FEM which solve for the potential on a 3D grid of data points. It is useful to compare the advantages and disadvantages of each method [16], [19].

**Meshing**

- FEM: An entire domain mesh is required.
- BEM: A mesh of the boundary only is required.
- Comment: The reduction in the size of the mesh implies that the problem complexity has been reduced by one dimension. This is advantageous as the creation of complex 3-dimensional meshes is time consuming. However, the FEM allows for the modeling of complex geometries.

**Solution Domain**

- FEM: The entire domain solution is calculated as part of the solution.
- BEM: The solution on the boundary is calculated first, and then the solution at domain points (if required) are found as a separate step.
- Comment: There are many problems where the details of interest occur on the boundary or are localized to a particular part of the domain, and hence an entire domain solution is not required. For problems involving infinite or semi-infinite domains, the BEM is favored as solving the whole domain is intractable.

**Approximations**

- FEM: The differential equation is being approximated.
- BEM: Boundary conditions are being approximated.
- Comment: The use of Green's second identity and a fundamential solution in the formulation means that the BEM involves no approximations of the differential equation in the domain; only in its approximations of the boundary conditions.

**Matrix Form**

- FEM: Sparse, banded, positive-definite, symmetric matrices are generated.

- BEM: Fully populated nonsymmetric matrices generated.
- Comment: The two methods generally produce matrices of different sizes due to the differences in size of the domain mesh compared to the surface mesh. There are problems where either method can give rise to the smaller system and quickest solution, depending partly on the volume to surface ratio. For problems involving infinite or semi-infinite domains, the BEM is to be favored.

**Numerical Integration**

- FEM: Element integrals are easy to evaluate.
- BEM: Integrals are more difficult to evaluate, and some contain integrands that become singular.
- Comment: In general, BEM integrals are harder to evaluate. Also the integrals that are the most difficult (those containing singular integrands) have a significant effect on the accuracy of the solution, so these integrals need to be evaluated accurately.

**Applicability**

- FEM: Widely applicable. Handles nonlinear problems well.
- BEM: Cannot even handle all linear problems.
- Comment: A fundamental solution must be found (or at least an approximate one) before the BEM can be applied. There are many linear problems (e.g., virtually any nonhomogeneous equation) for which fundamental solutions are not known. There are certain areas in which the BEM is clearly superior, but it can be rather restrictive in its applicability.

**Implementation**

- FEM: Relatively easy to implement.
- BEM: Much more difficult to implement.
- Comment: The need to evaluate integrals involving singular integrands makes the BEM at least an order of magnitude more difficult to implement than a corresponding finite element procedure.

In general, the FEM requires more computation. However, as computers increase in speed, this advantage diminishes and the FEM becomes more attractive for the advantages listed above. In addition, there are many more standard FEM programs to choose, therefore increasing user support. The BEM also works poorly in applications involving a large number of shells which makes it especially restrictive in this case where more realistic head geometries typically include 3 to 4 shells. For these reasons and the advantages above, the FEM was chosen as the numerical tool for these simulations.

# 2.0 METHODS : ANALYTICAL AND NUMERICAL

## 2.1 CONSTRUCTION OF THE ANTENNA ELEMENTS

The construction of the antenna elements is illustrated in Figure 3. The same techniques were used to create both the analytical and numerical antenna shapes. The control points were found by adding $x_{max}$ and $y_{max}$ and dividing the sum by the number of control points ($P_c$) plus one:

$$d_{inc} = \frac{x_{max} + y_{max}}{P_c + 1} \qquad . \tag{2.1}$$

This gives the incremental distance to travel along the perimeter of the bounding box as seen on the right side of Figure 3. The following give the equations for the sphere, parabola, ellipse, and hyperbola respectively.

$$\frac{x^2}{1} + \frac{y^2}{1} = R^2 \qquad , \tag{2.2}$$

$$a\frac{x^2}{1} - \frac{y}{1} = 0 \qquad , \tag{2.3}$$

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \qquad , and \tag{2.4}$$

$$\frac{y^2}{b^2} - \frac{x^2}{a^2} = 1 \qquad . \tag{2.5}$$

Thus we need to know $R$ for the sphere, $a$ for the parabola, and $a$ and $b$ for the ellipse and hyperbola. The following equations derive the formulas used to calculate the salient

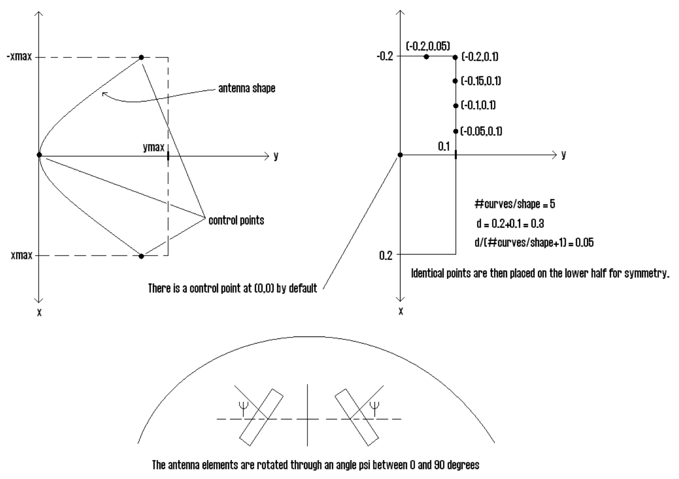

-xmax
xmax
ymax
y
x
antenna shape
control points
There is a control point at (0,0) by default
-0.2
0.2
0.1
(-0.2,0.05)
(-0.2,0.1)
(-0.15,0.1)
(-0.1,0.1)
(-0.05,0.1)
#curves/shape = 5
d = 0.2+0.1 = 0.3
d/(#curves/shape+1) = 0.05
Identical points are then placed on the lower half for symmetry.
Ψ
Ψ
The antenna elements are rotated through an angle psi between 0 and 90 degrees

Figure 3: Top left: Illustrates the use of control points around a bounding box (2*xmax wide and ymax high) to determine parameters of the specified shape. Top right: Shows a specific example of Top left using 5 control points. Bottom: Shows how the antenna elements can be rotated by an angle $\Psi$.

parameters of the conic section equations. For the sphere, $R$ can be calculated with the following equation, where the subscript, $c$, stands for "control points" as per Figure 3:

$$R = \sqrt{{x_c}^2 + {y_c}^2} \qquad . \tag{2.6}$$

For the parabola, $a$ can be calculated using

$$a = \frac{{y_c}^2}{x_c} \qquad . \tag{2.7}$$

For the ellipse, it is assumed that the $x$ and $y$ control points are at the inflection points, thus

$$a = x_c \qquad , and \tag{2.8}$$

$$b = y_c \qquad . \tag{2.9}$$

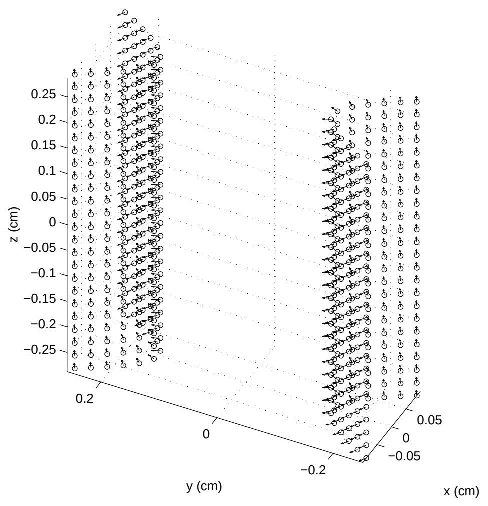


Figure 4: Vector field of ideal dipoles evenly spaced on hyperbolically shaped antenna elements. The dipoles were placed in an orientation perpendicular to the antenna elements and evenly spaced with-respect-to the arclength.

The hyperbola is a distance $\frac{d}{2}$ from the x-axis, where $d$ is specified by the user when designing the distance between the antenna. Thus,

$$\frac{y^2}{\left(\frac{d}{2}\right)^2} - \frac{x^2}{a^2} = 1 \qquad , \tag{2.10}$$

where

$$b = \frac{d}{2} \qquad . \tag{2.11}$$

Solving for $a^2$ we find

$$a^2 = \frac{{y_c}^2 b^2}{{x_c}^2 + d x_c} \qquad . \tag{2.12}$$

For the ideal dipole case, the dipoles have moments which are vectors, so a position and orientation must be specified. It was determined that putting the vectors at a perpendicular orientation to the surface would be consistent with the notion that the current leaves the surface of the antenna elements in a perpendicular direction. Determining this orientation is done by simply taking the gradient of the function set equal to zero:

$$y - f(x) = 0 \qquad , \tag{2.13}$$

$$\vec{m_d} = \vec{\nabla}(y - f(x)) \qquad . \tag{2.14}$$

An example of the application of (2.13) and (2.14) can be seen in Figure 4. The sources and sinks were placed at evenly spaced intervals along the arc length of the shape. This same length was also used as the spacing along the long axis. Had the sources been placed at even intervals along the axis, the density of sources and sinks around the areas of high curvature would have been exaggerated. Thus, it was assumed that the current density leaving the antenna elements was uniform.

## 2.2 ANALYTICAL METHODS

Analytical methods are important because they offer fast, closed-form solutions. Methods were developed that allow the antennas to be simulated with both ideal and nonideal current dipoles. Nonideal current dipoles are characterized by having an arbitrary distance between source and sink monopoles.

### 2.2.1 Ideal Current Dipoles in an Infinite Homogeneous Medium

Ideal current monopoles can be characterized by defining the current magnitude and conductivity as seen in the following equation [7]:

$$\Phi_{I_o} = \frac{I_o}{4\pi\sigma r} \qquad . \tag{2.15}$$

The field produced by a current monopole and the electrostatic field from a point charge are identical, provided that $I_o$ is replaced by $Q_o$, $\sigma$ is replaced by $\epsilon$ (the permittivity), and

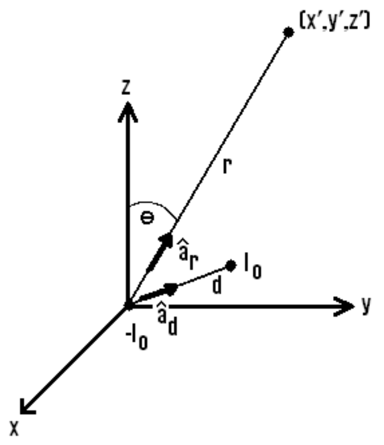


Figure 5: Dipole consisting of a sink $-I_o$ at the origin and a source $I_o$ at $d\hat{a}_d$, where d→0. Also illustrated is a field point at vector $r\hat{a}_r$ and polar angle $\theta$ [7].

$\vec{J}$ (current density) replaced by $\vec{E}$. The following equation shows this similarity:

$$\Phi_{Q_o} = \frac{Q_o}{4\pi\epsilon_o r} \qquad . \tag{2.16}$$

This example of *duality* is convenient as there are many closed-form equations for electrostatic point charges that can be converted to electrostatic point current sources. Although, a point current source cannot exist, the potential can be calculated from the presence of a current sink and source together [7].

The ideal dipole equation for an infinite homogeneous medium will be derived here, and the results will be used to determine whether it is a good or bad assumption to place ideal dipoles on the surface of the antenna elements. This is of interest, as equations have previously been derived for sophisticated models of the geometry of the head i.e., ideal dipoles in four-shell models of the head.

A dipole of arbitrary orientation is illustrated in Figure 5, where the negative pole is placed at the origin of the coordinate system. If the positive pole were at the origin, the potential would be zero. Thus, the dipole potential arises due to the separation of the positive and negative poles. This potential can be found by examining the potential for the positive monopole and evaluating the change in potential brought about by moving the the monopole from the origin to its dipole position. This can be approximated from the first derivative of the potential. A derivative of $\Phi$ is taken with respect to the direction $\hat{a}_d$ (a directional derivative) and then multiplied by the magnitude, d. Thus, we have

$$\Phi_d = \frac{\partial(\frac{I_o}{4\pi\sigma r})}{\partial a_d} d \qquad . \tag{2.17}$$

The directional derivative in (2.17) equals the gradient in the direction $\vec{a}_d$ so that

$$\Phi_d = \nabla(\frac{I_o}{4\pi\sigma r}) \cdot d\hat{a}_d \qquad , \tag{2.18}$$

and finally, since $I_o d = p$:

$$\Phi_d = \frac{p}{4\pi\sigma}\nabla(\frac{1}{r}) \cdot \hat{a}_d \qquad . \tag{2.19}$$

Using the relation

$$\nabla\frac{1}{r} = \frac{\hat{a}_r}{r^2} \qquad , \tag{2.20}$$

(2.19) becomes

$$\Phi_d = \frac{p}{4\pi\sigma r^2}\hat{a}_r \cdot \hat{a}_d \qquad . \tag{2.21}$$

Equation (2.21) can be simplified further given that the dipole is oriented along the z-axis and located at the origin [7]:

$$\Phi_d = \frac{p\ cos\theta}{4\pi\sigma r^2} \qquad . \tag{2.22}$$

The reason that an ideal dipole was investigated first, rather than two monopoles, is that neuronal activity in the brain is typically modeled by a configuration of ideal dipoles. Thus, it was a simple extension to set up a model which incorporated ideal dipoles. For instance, to implement these ideal current dipoles, a program was used, written by Sun for EEG analysis (Appendix .1).

Conceptually it is clear that the use of two sets of dipoles will not return the same solution as using two monopoles (source and sink). The following set of equations derives

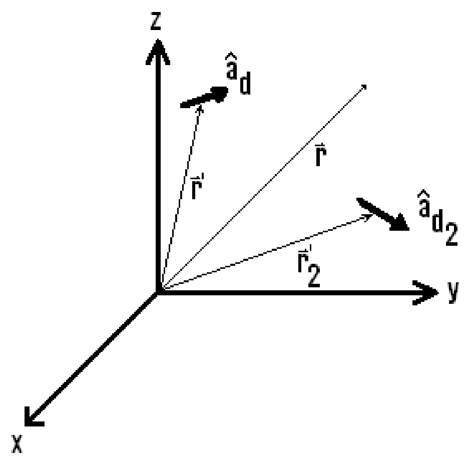


Figure 6: Two dipoles shown in a 3D coordinate system. The primed variables are the locations of the dipoles and the unprimed variable is the location of interest.

the field for two sets of dipoles in an infinite homogeneous medium. The coordinate system can be seen in Figure 6.

Starting from (2.21) and using superposition, one can write the following equation for the second dipole, where here it is assumed that the moments ($p$) are constant:

$$\Phi_{2dipoles} = \frac{p}{4\pi\sigma x^2}\hat{a}_x \cdot \hat{a}_d + \frac{p}{4\pi\sigma {x_2}^2}\hat{a}_{x_2} \cdot \hat{a}_{d_2} \qquad , \tag{2.23}$$

where:

$$x = |\vec{r} - \vec{r}\,'| \qquad , \tag{2.24}$$

and

$$x_2 = |\vec{r} - \vec{r_2}\,'| \qquad , \tag{2.25}$$

where the primed variables represent the location of the dipoles.

### 2.2.2 Nonideal Current Dipoles in an Infinite Homogeneous Medium

Nonideal current dipoles are classified as a set of opposing monopoles spaced at arbitrary distances. Each monopole obeys the laws of superposition and therefore the fields can simply be added together. Beginning with (2.15) and Figure 6, using superposition and rearranging, one can arrive at [7]:

$$\Phi_{tot} = \frac{I_o}{4\pi\sigma}(\frac{1}{x_2'} - \frac{1}{x'}) \qquad , \tag{2.26}$$

where $x'$ and $x_2{}'$ are from (2.24) and (2.25), respectively.

Then comparing (2.23) and (2.26), one can find the error that arises from using two dipoles as compared to two monopoles for the antenna elements. This is shown in Figure 7. One can see that the error is quite large and so ideal dipoles are not an acceptable means of simulation in this case. Therefore, the previous literature for describing bioelectric sources as ideal dipoles in the head is not applicable for these simulations.

Although the equations for the infinite medium are insightful and provide a minimum baseline for error, they do not elucidate the characteristics of the dipoles inside of the head. To simulate the head, a spherical shape is assumed for the next section.

### 2.2.3 Ideal Current Dipoles in a Spherical Homogeneous Medium

There are many equations that have been developed to solve this problem. Probably the simplest equation was developed by Sidman [12]:

$$\Phi = \frac{1}{4\pi\sigma q}\sum_{i=1}^{3} m_i[\frac{2(e_i - d_i)}{q^2} + e_i + \frac{e_i s - d_i}{q + 1 - s}] \qquad , \tag{2.27}$$

where $d_i$, $m_i$, and $e_i$, i=1,2,3 (1=x,2=y,3=z), are, respectively, the vector elements of $\vec{d}$, $\vec{m}$, and $\vec{e}$ which represent the dipole location, current moment, and scalp location; $\sigma$ and $\phi$ are the conductivity and potential at $\vec{e}$, respectively; and q and s are equal to $\mid \vec{e} - \vec{d} \mid$ and $\vec{e} \cdot \vec{d}$, respectively.

### 2.2.4 Nonideal Current Dipoles in a Spherical Homogeneous Medium

To increase simulation sophistication nonideal dipoles in a spherical medium were used. The derivation of this equation is quite complicated, using Legendre polynomials. It was

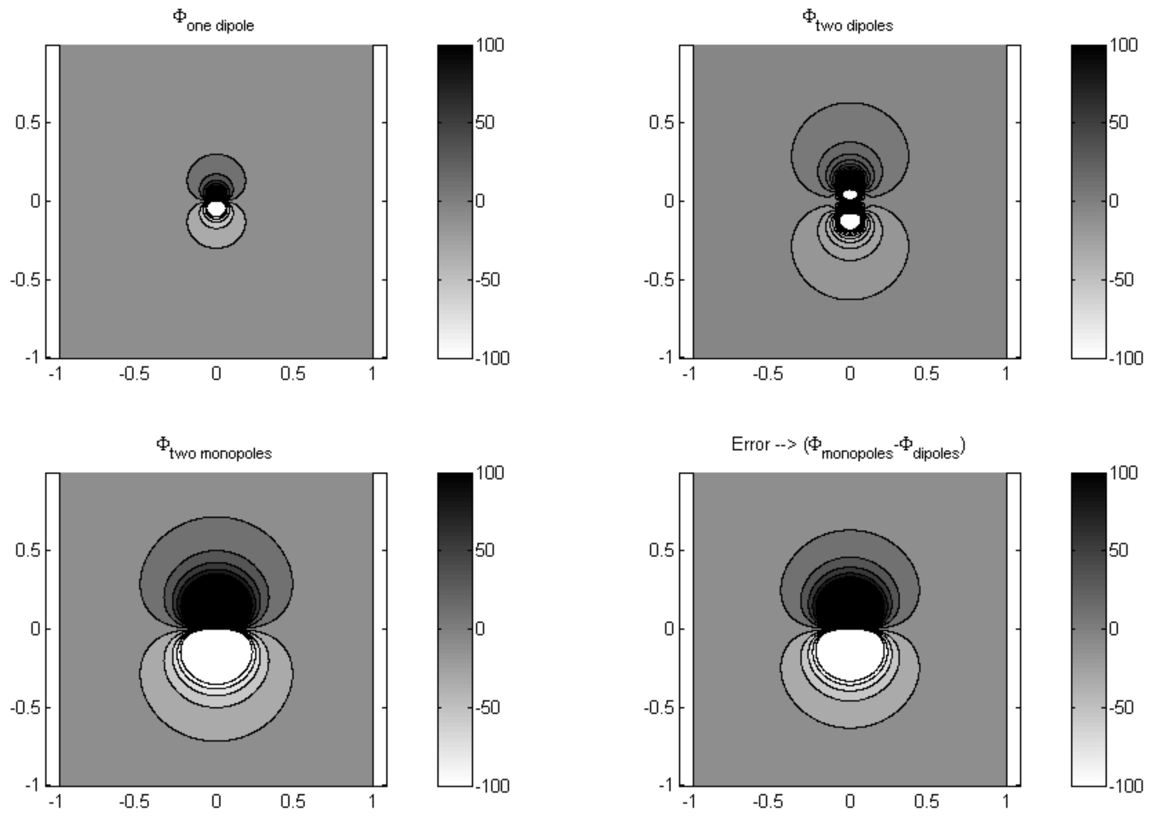


Figure 7: Top left: The field created by an ideal dipole. Top Right: Superposition of two ideal dipoles. Bottom Left: Two monopoles (source and sink). Bottom Right: Error between 2 dipoles and 2 monopoles (Appendix .7).

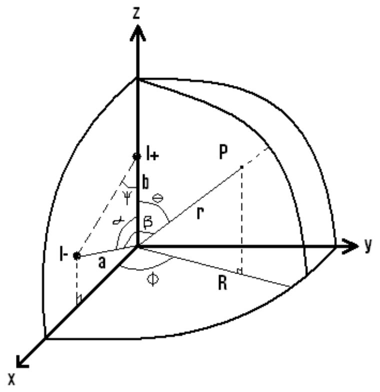


Figure 8: Coordinate system illustrating the variables from Equation (2.28)

developed by Ernest Frank [13] and is

$$\phi(\theta,\beta) = \frac{I}{4\pi\sigma}\left[\frac{2}{r_b} + \frac{2}{r_a} + \frac{1}{R} ln \frac{r_a + R - a * cos\ \beta}{r_b + R - b * cos\ \theta}\right] \quad , \tag{2.28}$$

where $r_b$ and $r_a$ extend from the positive monopole to the field point and from the negative monopole to the field point, respectively.

It is easiest to illustrate the variables before they are described as shown in Figure 8 From Figure 8 one can see that $d$ does not have to go to zero and thus this is a nonideal dipole arrangement inside of a spherical homogeneous medium.

With (2.27) and (2.28) one can find the error created by using dipoles. This error can be seen in Figure 9. Looking at this error, one can draw the conclusion that although (2.27) is quite useful for representing electrical activity due to neurons, it is not ideal for the closed-form simulation of the antenna.

### 2.2.5 Ideal Current Dipoles in a Four Shell Spherical Head Geometry

A four shell model of the head is important because it incorporates the different layers of tissue. The different tissues layers are brain, cerebral spinal fluid (CSF), bone, and scalp. Their conductivities are given in Table 1. The closed-form computation of the voltage in this case can be computationally intensive, but fortunately Sun found a compact mathematical solution which can be run efficiently in C [15]. Although the source model is an ideal dipole,

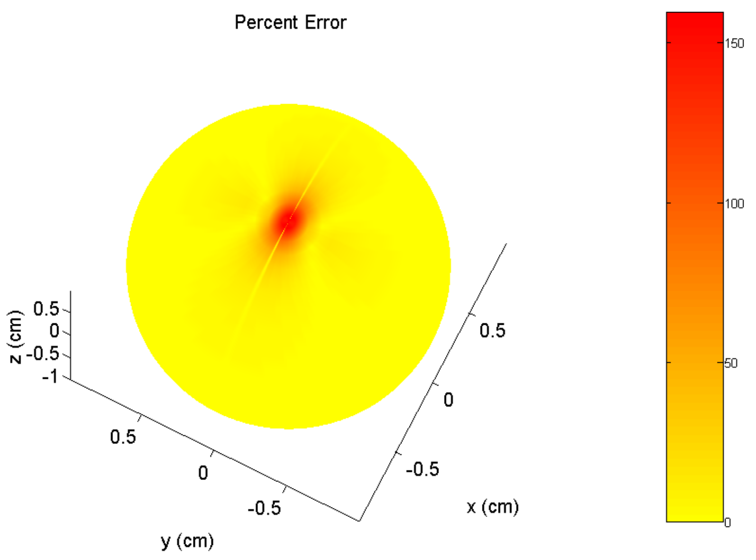


Figure 9: Shaded plot showing the percent error for two sets of dipoles as compared to one nonideal dipole. The error is concentrated in the region between the antenna elements as the approximation breaks down in the near-field (Appendix .8).

the program still holds value as it can be used to compare an ideal dipole arrangement of current sources and sinks on the antenna elements to FEM four shell results. A Mex function was written to interface this code with MatLab script (Appendix .4).

### 2.2.6 Graphical User Interface

Initially it was difficult to visualize how the antenna was to be placed inside the head and how the antenna elements would be simulated. It was decided that a graphical user interface (Figure 10) would aid in this process. The total current traversing from the source to sink was set to be 1 mA. Because it was hypothesized that the shape of the antenna would affect the signal strength in both the near and far field, it was decided that conic sections would provide a good framework to test the shapes of the antennas. Figure 11 shows a semicircle, parabola, ellipse, and hyperbola.

Because the shapes have subtle differences, care was taken so that the dipoles were oriented perpendicularly to the surface of the antenna elements. Further, the dipoles were

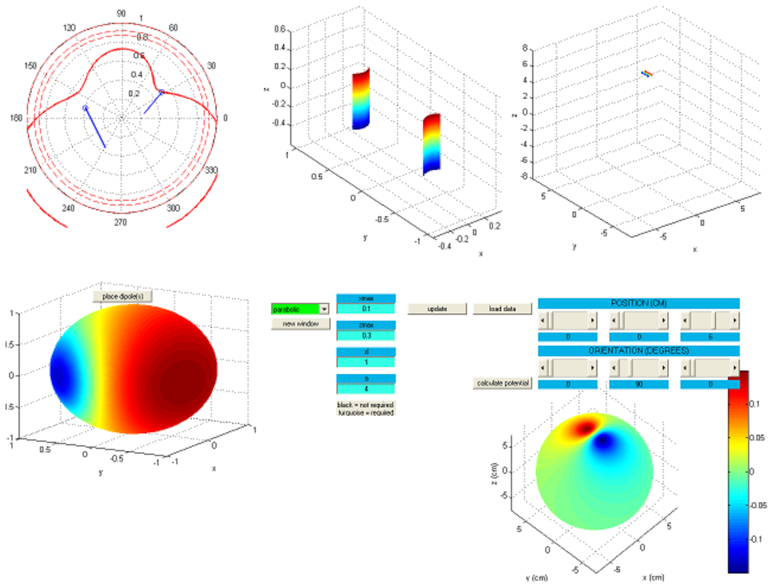


Figure 10: Graphical User Interface allowing user to specify antenna element shape, size, position, and orientation. It is also capable of simulating one or two dipoles alone by user positioning in polar coordinates (Appendix .1).

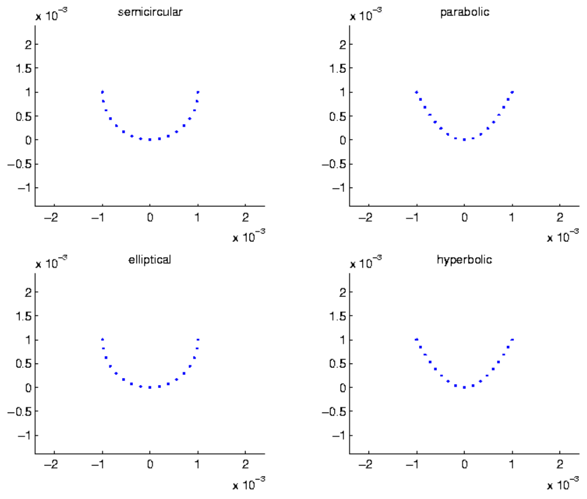


Figure 11: Various conic sections. Top left: Semicircular; Top right: Parabolic; Bottom left: Elliptical; Bottom right: Hyperbolic.

placed so that the current density would be constant. Figure 4 shows how these dipoles were placed. For the semicircle the increment along the arc length could be calculated easily, but for the parabola, ellipse and hyperbola an iterative method was used to specify dipole placement. One interesting question is how many points is one required to seed on the shape so that the results will reflect the different shapes. This aspect of the model is a problem of sampling. The fewer the samples, the less computation needed to arrive at a solution. A suitable solution to this problem has not been worked out at this point. A frequency domain representation of the curves may lead to insights into the minium number of points required to characterize the shape. To circumvent this problem, the number of monopoles was simply made very dense so that this would not be a factor.

## 2.3 NUMERICAL METHODS

Numerical methods offer a way to simulate more complex geometries. Rarely can a closed-form equation be derived to represent a real system. The numerical analysis was carried out in FEMLab (Finite Element Method Laboratory). This program runs together with MatLab and uses many similar features.

### 2.3.1 FEMLab

FEMLab has many modules for finite element solving (FEMLab standard package, Electromagnetics, Structural Mechanics, and Chemical Engineering). Initially it was thought that the Electromagnetics module would be required but it was found that the module did not have any increased functionality but only compiled commonly used finite element techniques from the FEMLab standard package. The FEMLab module contains an application mode called Conductive Media DC, which gives the user the ability to solve (1.2) for current sources. If $\sigma$ is a tensor, (1.2) is written as:

$$\vec{\nabla} \cdot \overleftrightarrow{\sigma} \vec{\nabla} \Phi = -\gamma \tag{2.29}$$

Equations (1.2) and (2.29) mean that the divergence of the electric field is nonzero where

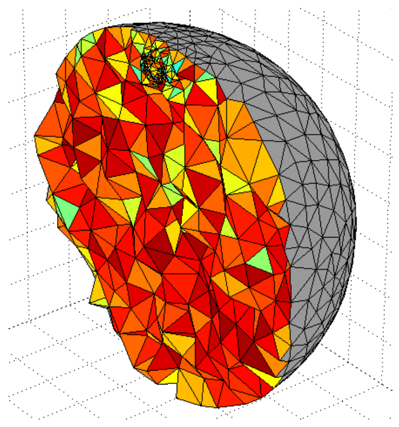


Figure 12: An example of a finite element mesh

there are current sources. After discretizing the Poisson equation over a tetrahedral mesh domain, the divergence of the electric field is approximated by (1.21) where $\overleftrightarrow{K}$ is typically called the stiffness matrix, $\vec{U}$ is a vector composed of voltages at the nodes, and $\vec{F}$ is a source vector indicating the flux through the nodes where it is only nonzero for nodes that are corners of elements containing a current source.

For the boundary conditions, equation (1.10) is utilized by FEMLab. The sphere and the geometry within the sphere is separated into subdomains. For each subdomain a conductivity and source current density can be specified. For the sphere and the epoxy the source current density is set equal to zero. For the antenna elements the source current density is set such that 1 mA will be sent and received. The values of the conductivities for the subdomains was set according to Table 1. The geometry is meshed using a seven step process which is transparent to the user [14]. Figure 12 shows an example of the resulting mesh.

For this problem the Good Broyden iterative solver was used to solve (1.21). I will not go into the details of this, but normally these solvers are used to solve linear systems with a positive definite matrix $\overleftrightarrow{K}$, however for this condition to be true, the matrix must be preconditioned. Incomplete LU factorization was used in this case to give $\overleftrightarrow{K}$ the appropriate properties to be solved effectively by the Good Broyden method [14].

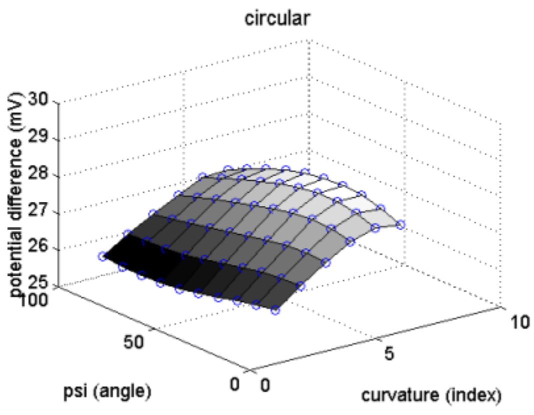
# 3.0 RESULTS

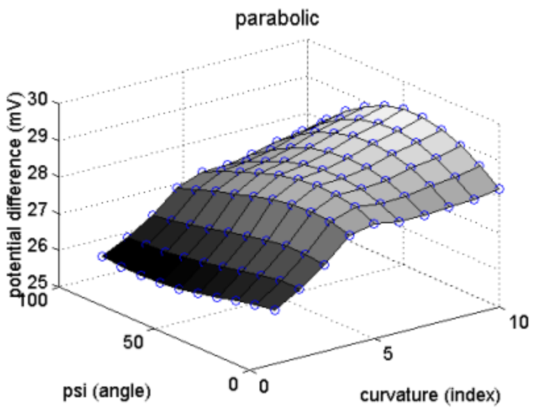
## 3.1 NONIDEAL DIPOLES IN A SPHERICAL MEDIUM: ANALYTICAL INVESTIGATION

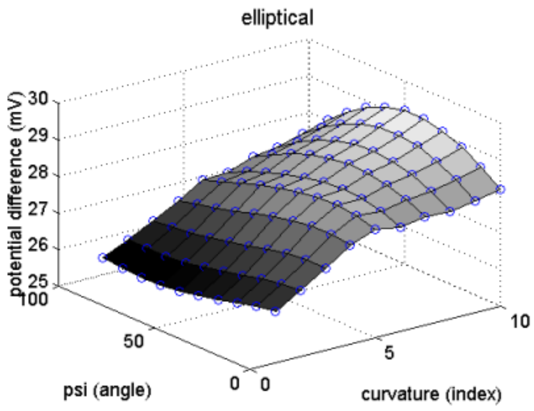
The following results were obtained using Equations (2.6)-(2.12) to specify the antenna element points. These points were then plugged into (2.28). The z-axis shows the maximum potential found on the surface of the sphere, the y-axis the orientation of the antenna, and the x-axis an index of the relative curvature of the shape. To describe how the curvature index is measured, Figure 3 shows five control points, therefore, in 1the curvature has an index from 1 to 5. Notice that the circle only has 6 curvature indices, whereas the others have 10. This is because the curvature is uniform over the surface of the circle and is proportional to $\frac{1}{R}$. Therefore, as the curvature is increased, the circle becomes smaller and smaller. For indices higher than 6, the shape defined was too small to cross through the control points so these points were not simulated. The more indices used to describe the curvature, the higher the resolution of the results, however since the results are not changing quickly, it was found that 10 control points was sufficient for the results of Figure 13.

### 3.1.1 Investigation of Optimum Angle

From the previous section one can see that the optimum angle seems to be around 45 degrees. Further, it makes sense that this angle would be a function of the distance that the antenna is from the surface. For instance, if the antenna was at the middle, it would make sense that the optimum angle is 0 degrees. Further, if the antenna was near the surface, the optimum angle should be steeper because the antenna elements would have to rotate to point at the







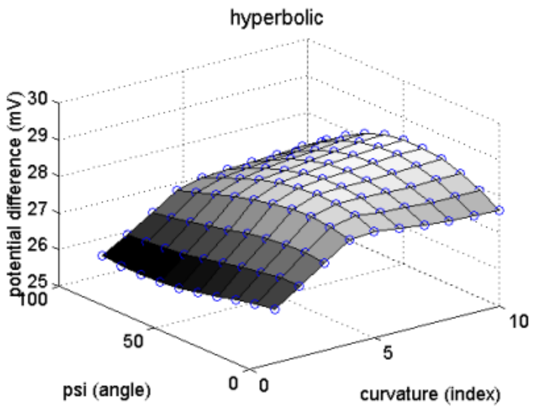


Figure 13: Top left: Semicircular; Top right: Parabolical; Bottom left: Elliptical; Bottom right: Hyperbolic (Appendix .2)

surface. A hypothetical optimum angle was derived purely based on distance:

$$\Psi_{hypothetical} = arctan(\frac{y}{d/2}) \quad , \tag{3.1}$$

and a plot is shown in Figure 15. The results in Figure 15 agree with (3.1) in the sense that the angle does increase, however the actual curve is concave up. This indicates that the antenna elements destructively interfere with the generated surface voltages, requiring the antenna elements to be rotated away from each other.

## 3.2 FINITE ELEMENT RESULTS: NUMERICAL INVESTIGATION

### 3.2.1 Proof of Concept in 2D

To show proof of concept that the far field can be affected by near field changes, a 2D numerical analysis was done comparing:

1. Antenna elements without epoxy (Figure 16).
2. Antenna elements with epoxy (Figure 17).
3. Antenna elements with epoxy and a reflector below (Figure 18).

Equipotential lines are plotted for all three with polar plots adjacent showing the potential on the surface. Note that the absolute value of the voltage was taken so that it could be plotted in polar coordinates. Notice the voltage on the surface is lower for the case without epoxy between the elements. Therefore, an insulator between the elements is important for higher transmission efficiency. Notice also that without the reflector, the equipotential lines are nearly symmetric from the top to the bottom of the antenna, however, with the reflector, the equipotential lines are skewed towards the surface, thus improving directivity of the antenna, proving that the reflector can interact with the near field and thus alter the far field. This also translates into a higher surface potential.

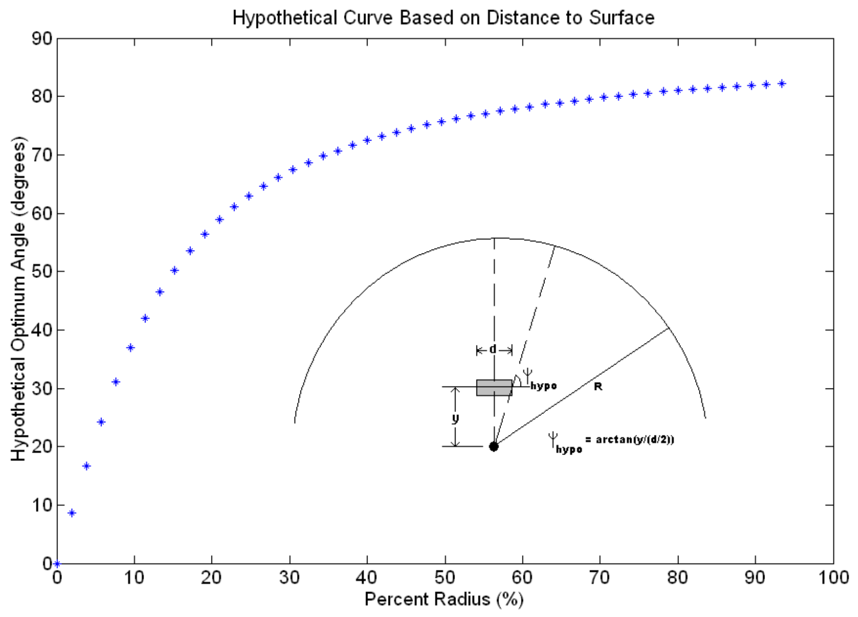


Figure 14: As the antenna moves towards the surface, the optimum angle increases and is concave down. This graph is based on (3.1) and code is found in Appendix .9.

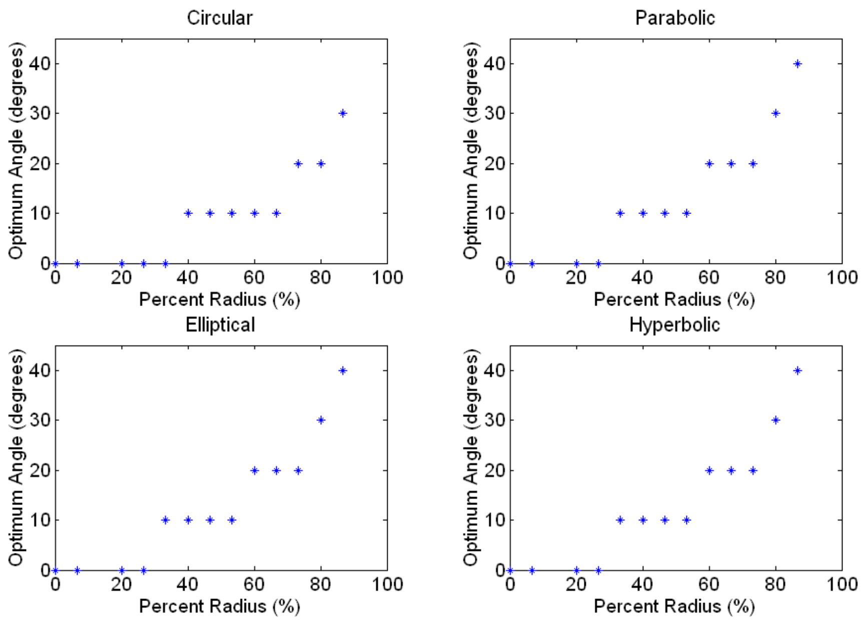


Figure 15: Shows that as the antenna is moved toward the surface, the optimum angle rotates so that the antenna elements will point towards the surface (Appendix .9).

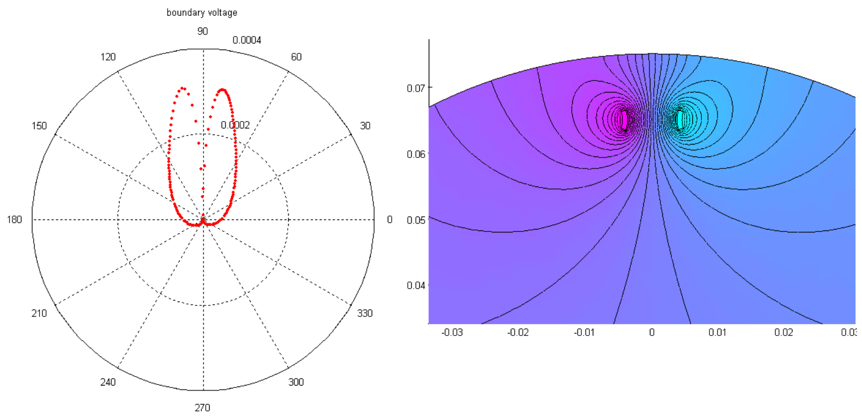


Figure 16: The antenna elements are simulated without an insulator between the elements. The polar plot displays the surface voltage. The right plot shows equipotential lines surrounding the antenna elements. The polar plot has units of Volts (Appendix .12).

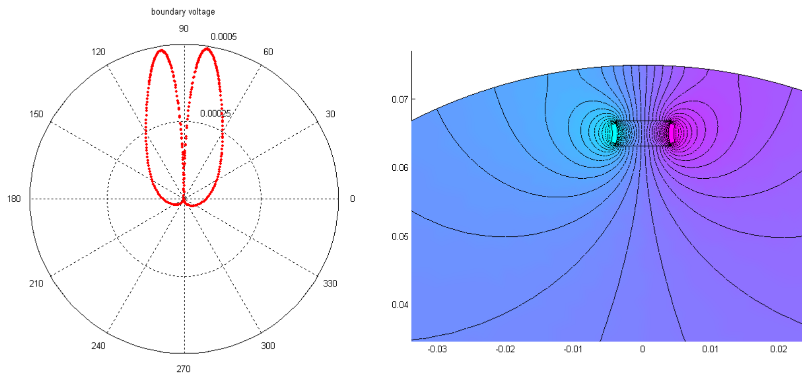


Figure 17: The antenna elements are simulated only with an insulator between the elements. The polar plot displays the surface voltage. The right plot shows equipotential lines surrounding the antenna elements. The polar plot has units of Volts (Appendix .11).

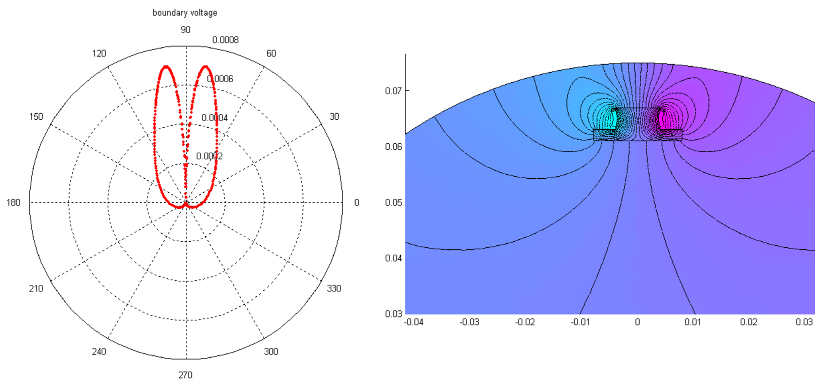


Figure 18: An insulating material is placed below the antenna elements to act as a current reflector. The polar plot displays the surface voltage. The right plot shows equipotential lines surrounding the antenna elements. The polar plot has units of Volts (Appendix .13).

Table 3: Results for Proofs of Concept

| Analysis | Max. Surface Voltage (mV) |
|---|---|
| No Epoxy (2D) | 0.3 |
| Epoxy (2D) | 0.5 |
| Reflector (2D) | 0.7 |
| No Epoxy (3D) | 12 |
| Epoxy (3D) | 16 |

### 3.2.2 Proof of Concept in 3D

Again, to show proof of concept that the far field can be affected by near field changes, a 3D numerical analysis was done comparing:

1. Antenna elements without epoxy (Figures 20 and 21).
2. Antenna elements with epoxy (Figures 22 and 23 ).

For the three-dimensional analysis, two slices were taken. These slices are shown in Figure 19. Again, as in the 2D case, note that the absolute value of the voltage was taken so that polar plots could be made. It is clear that the simulations with epoxy between the elements have higher surface voltages. Although no 3D simulations were done with a reflector, from the given 2D and 3D results, it is evident that one can use an insulator between the elements to increase the surface voltage. Table 3 compiles the results for the proofs of concept.

### 3.2.3 Finite Element Analysis in 3D

The results of the 3D FEA are shown in Figure 24. These results incorporate the epoxy into the simulation. Fewer points were simulated due to the large amount of time and user input required to carry out the simulation. Although the shape of these results is much like Figure 13, the FE results are approximately 30-35 percent larger than the analytical results.

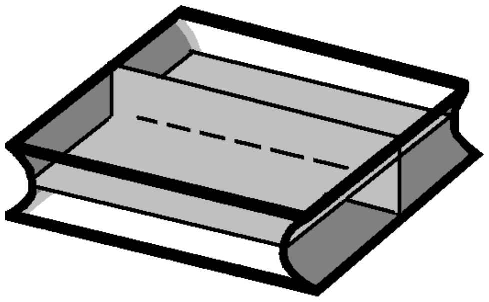


Figure 19: Depicts the 2 planes where the slices were taken.

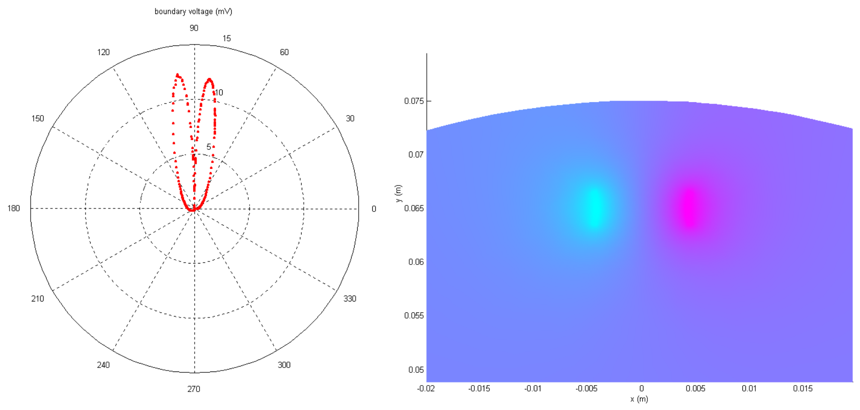


Figure 20: The antenna elements are simulated without an insulator between the elements. The polar plot displays the surface voltage (Code to extract slices from FEMLab in Appendix .10).

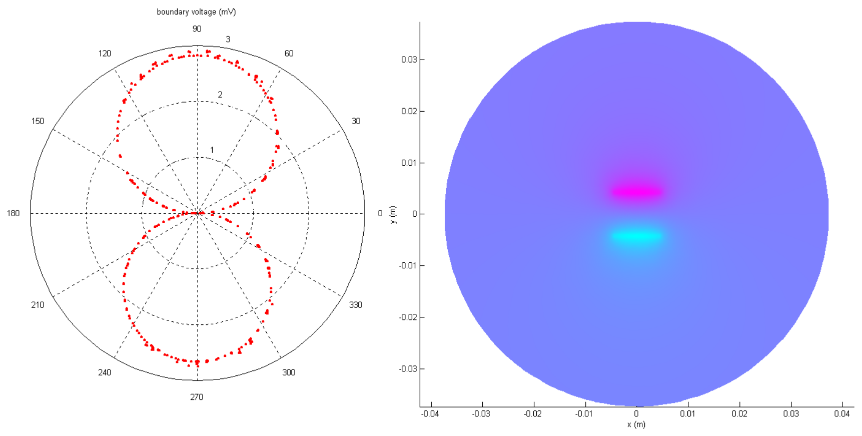


Figure 21: The antenna elements are simulated without an insulator between the elements. The polar plot displays the surface voltage.

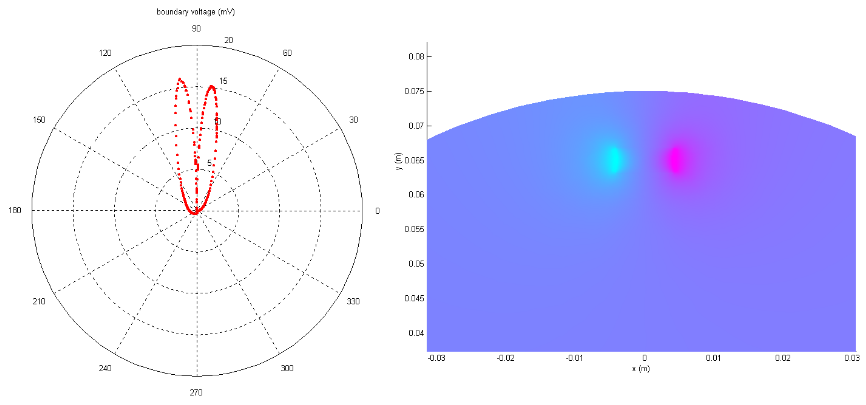


Figure 22: The antenna elements are simulated with an insulator between the elements. The polar plot displays the surface voltage.

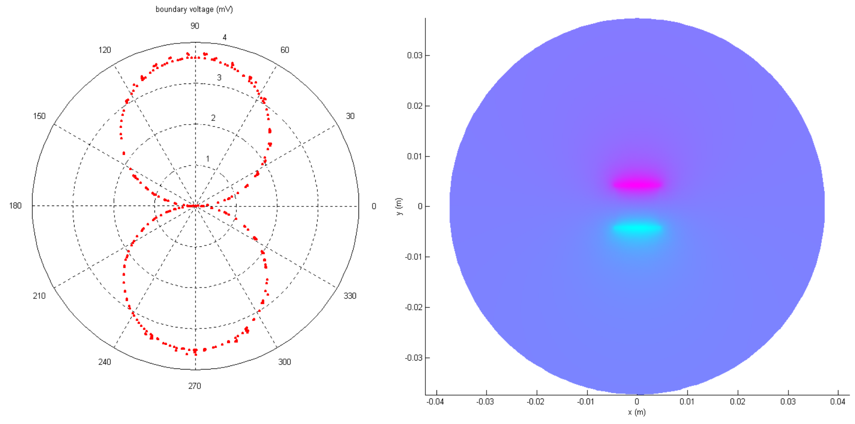


Figure 23: The antenna elements are simulated with an insulator between the elements. The polar plot displays the surface voltage.

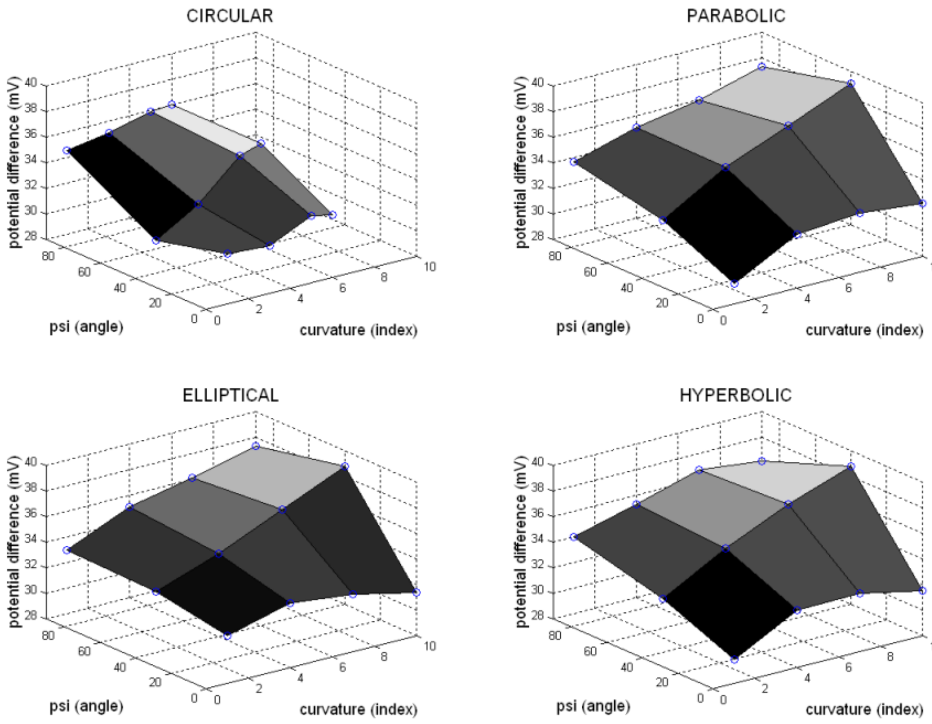


Figure 24: Results for the FEA with epoxy between the antenna elements. Top left: Semi-circular Top right: Parabolical Bottom left: Elliptical Bottom right: Hyperbolic

# 4.0 DISCUSSION

Several issues will be explored in the discussion. The first will be the verification of the finite element results by creating similar geometries and source conditions in the analytical domain. Second will be the parameters used in creating the finite element results. Specifically, convergence of the solution will be tested for increasing difference between the epoxy and antenna element conductivities. Ideally, the different conductivities could be explicitly expressed in the FEA but numerical instability does not allow this. Thus, as long as the conductivities are sufficiently different, the solution should not vary to a significant degree.

## 4.1 VERIFYING ANALYTICAL AND NUMERICAL RESULTS FOR THE 1-SHELL MODEL

For comparability, it is important that similar geometries give similar results for the analytical and numerical cases. Further, it also proves that the FEA is being set up correctly. To this end, FE models were made which were similar to closed-form models already created. The following models were compared:

1. Nonideal dipole around the center of a sphere (Figure 25).
2. Nonideal dipole near the surface of a sphere (Figure 26).
3. Antenna elements at the surface (Figure 27).
4. All shapes, curvatures and angles (Figure 28).

From Figures 25-27 one can see that the magnitudes are the same for all three types of simulated shapes. For these cases it was easy to recreate the geometries from the analytical model to the FE one, however this proved to be a more difficult task for the fourth case, which

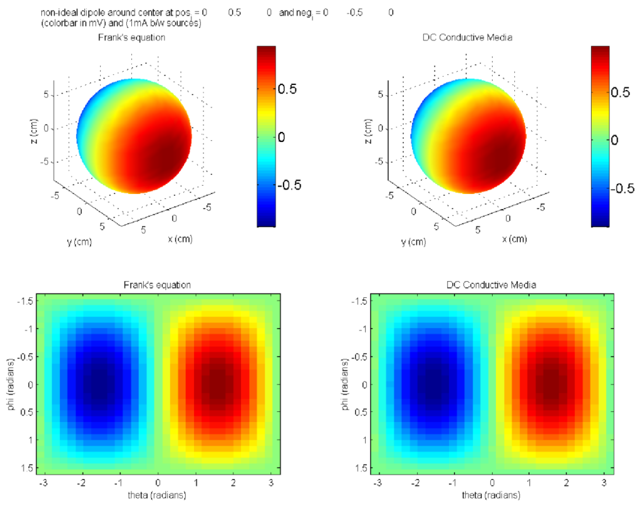


Figure 25: Comparison of Finite Element and Frank equation solution for (nonideal dipoles) current sources around the center. The top plots show the potential plotted on the surface of a spherical model of the head. The bottom plots show the same voltage distribution plotted on a grid (Appendix .5).

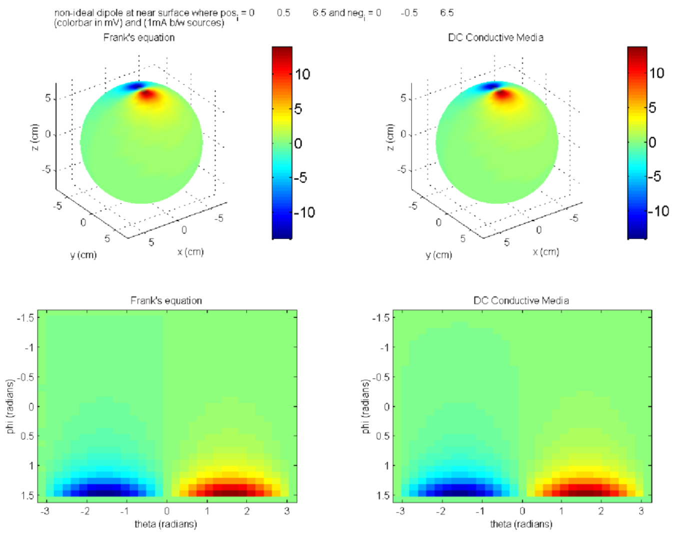


Figure 26: Comparison of FEM and Frank equation solution for current sources near the surface. The top plots show the potential plotted on the surface of a spherical model of the head. The bottom plots show the same voltage distribution plotted on a grid.

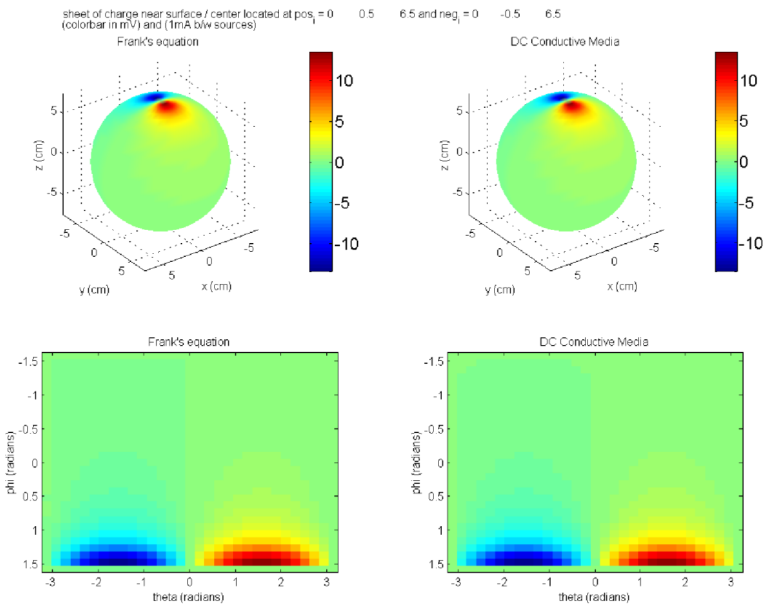


Figure 27: Comparison of FEM and Frank equation solution for current sheets near the surface.

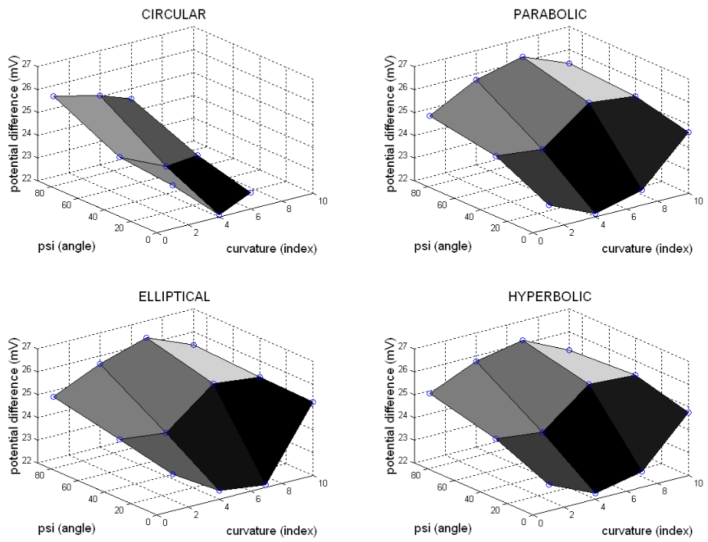


Figure 28: Finite element solution for all shapes, angles and curvatures where there is no epoxy between the elements. These results should compare well with Figure 13. The error is shown in Figure 29 (Code found in Appendix .16).

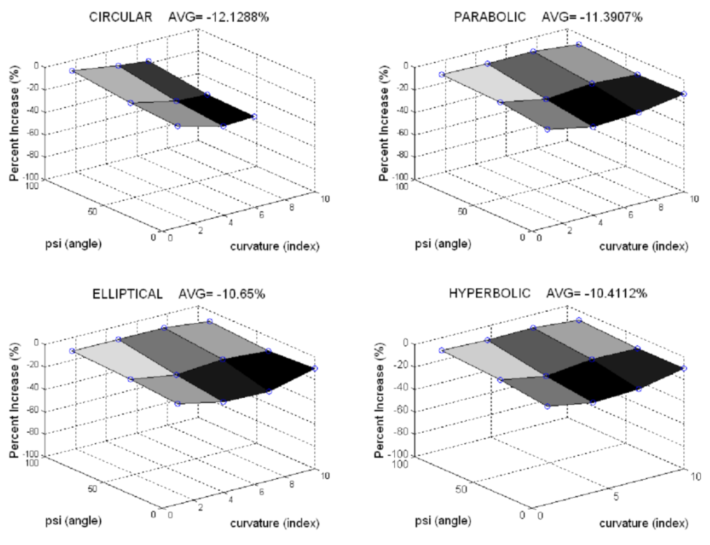


Figure 29: Comparison of FEM and Frank equation solution for all shapes, angles and curvatures. The percent increase represents how different the finite element solution is compared to the analytical.

is shown Figure 28. In the analytical model the antenna elements were modeled as sheets and thus had an infinitesimal thickness. This cannot be done in a FE domain because if the antenna element thickness was made too small, the solution did not converge. Therefore, Figure 29 describes this error. The typical error value is about -11%, meaning the FE results were lower in voltage. The percent error was calculated by the following equation:

$$\%\ error = \frac{FiniteElement - Analytical}{Analytical} * 100 \quad . \tag{4.1}$$

### 4.1.1 Comparing FE with Epoxy to Analytical Solutions

Now that it has been established that the analytical can be compared to the FEM, it is important to see how simulating the epoxy alters the result. Therefore, the percent increase can be calculated using the results from (4.1) and Figures 13 and 24. Therefore, one can see that the epoxy increases the surface voltage between 30 and 35 percent.

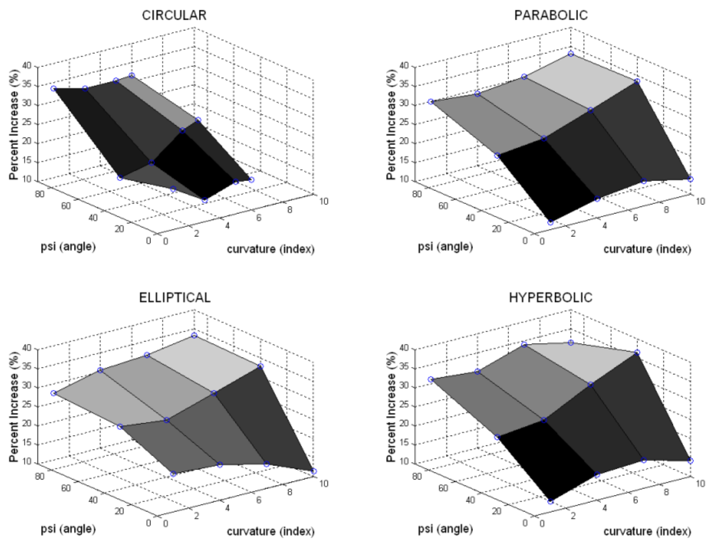


Figure 30: Comparison of FEM and Frank equation solution for all shapes, angles and curvatures with epoxy. The percent increase represents how the epoxy does not allow current to short between the antenna elements and thus forces it to go around the antenna and give better surface voltages (Appendix .15).

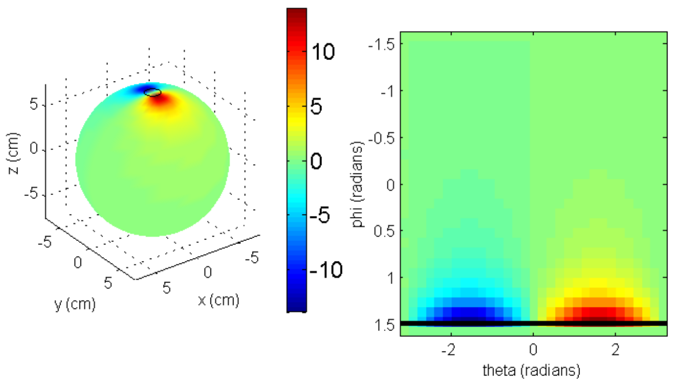


Figure 31: The black line shows the maximum value for each column which is confined to only one row or in other words, one value of angle $\phi$ (Appendix .5).

### 4.1.2 Basis Functions

Originally, the supplied volume for the current source and sink had been incorrectly specified. This caused the volume current density to be underestimated. This error was quantified by taking the maximum values for each column of the $\phi$ vs $\theta$ array. Because of the symmetry of the dipoles, this turned out simply to be one row i.e., a constant value of $\phi$. This can be seen in Figure 31.

After finding the maximum values from the Frank equation (2.28) and FEA, they were subtracted to find the error which resulted in the top plot of Figure 32.

From the bottom plot of Figure 32 it is easy to assume that the error is a sinusoid. At first it was thought a mistake had been made because the error was sinusoidal, but it was soon realized that if both plots were sinusoids with different amplitudes of the same frequency then their difference (error) is also a sinusoid. Thus, it was found that at the maximum, the voltage is sinusoidal. But was this true across all constant values of $\phi$? For insight the voltage was plotted for all values of $\phi$ and visually inspected. Figure 33 shows all rows together from several views. These are indeed all sinusoids of varying amplitude.

Thus, for this configuration of the current source the amplitude of the voltage on the surface of the sphere is simply a modulated sine function. It would be interesting to know why this occurs so a closed-form equation for an ideal dipole was manipulated to see how

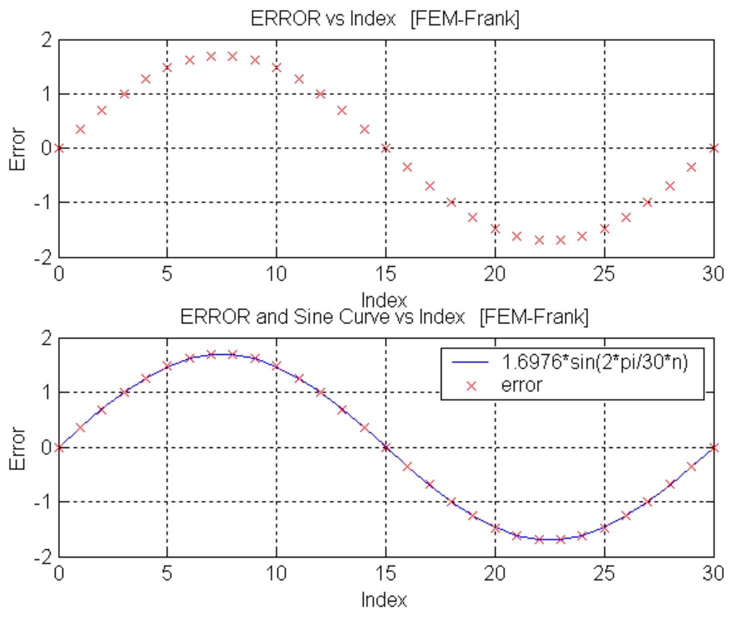


Figure 32: Top Plot: Error. Bottom Plot: Error with a sine curve overlaid (Appendix .5).

the sine curve emerges. The equation for a unit-radius spherical volume conductor with a homogeneous conductivity and an ideal dipole inside has the form in (2.27). The dipole whose surface voltage had been found had the following simplifications:

$\vec{m}$ = (0,$m_y$,0) i.e., the dipole moment is in (or opposite to) the direction of the y-axis

$\vec{d}$ = (0,0,$d_z$) i.e., the dipole was placed on the z-axis

$\vec{e} = (\cos\phi \cos\theta \, , \cos\phi \sin\theta \, , \sin\phi)$

$q = |e\text{-}d| = |(\cos\phi \cos\theta \, ,\cos\phi \sin\theta ,\sin\phi\text{-}d_z)|$

$= (1\text{-}2d_z \sin\phi + d_z^2)^{1/2}$

$s = e\cdot d = d_z \sin\phi$

Plugging this back into the equation we get

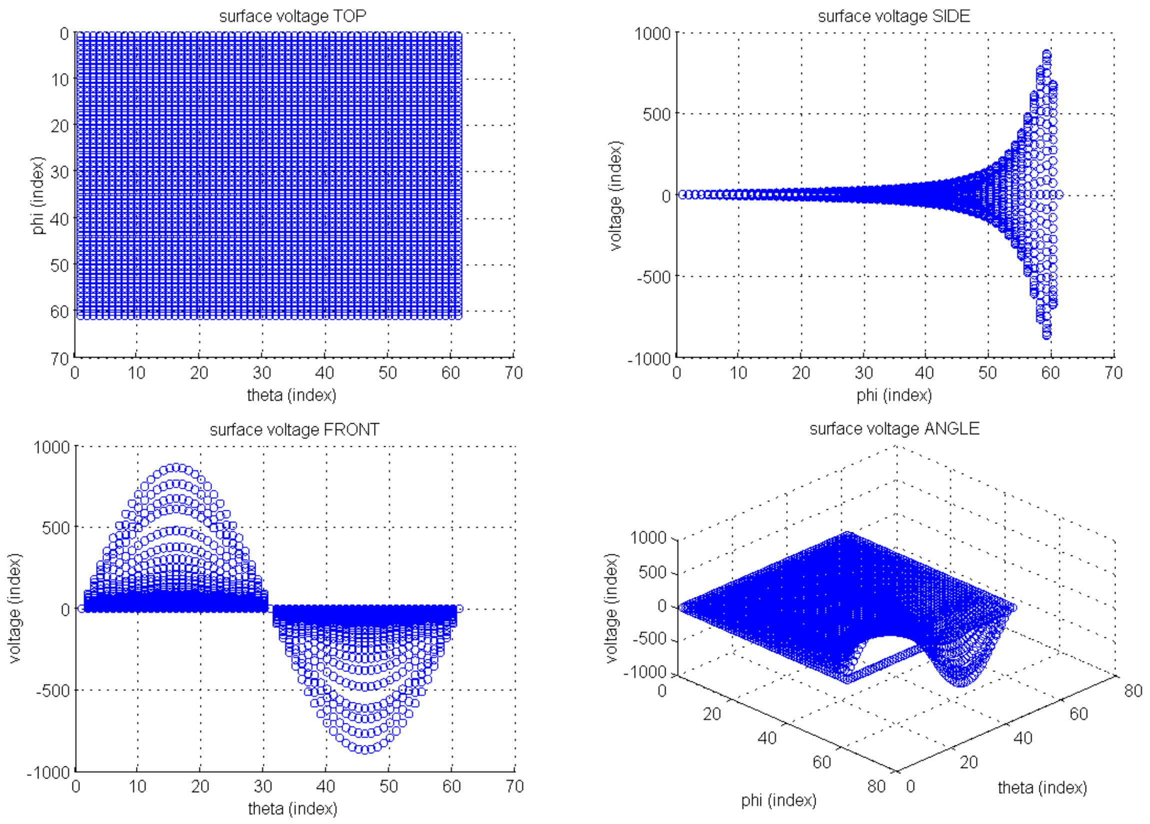


Figure 33: Different angles of the voltages plotted on a rectangular grid (Appendix .5).

$$\Phi = \frac{1}{4\pi\sigma q} m_y \{ \frac{2cos\phi sin\theta}{(1-2d_z sin\phi + {d_z}^2)^{3/2}} + \frac{cos\phi sin\theta}{(1-2d_z sin\phi + {d_z}^2)^{1/2}} + \frac{cos\phi sin\theta sin\phi d_z}{(1-2d_z sin\phi + {d_z}^2)^{1/2}[(1-2d_z sin\phi + {d_z}^2)^{1/2} + 1 - d_z sin\phi]} \} \quad . \tag{4.2}$$

Pulling out the $sin\theta$ term:

$$\Phi = \frac{1}{4\pi\sigma q} m_y sin\theta \{ \frac{2cos\phi}{(1-2d_z sin\phi + {d_z}^2)^{3/2}} + \frac{cos\phi}{(1-2d_z sin\phi + {d_z}^2)^{1/2}} + \frac{cos\phi sin\phi d_z}{(1-2d_z sin\phi + {d_z}^2)^{1/2}[(1-2d_z sin\phi + {d_z}^2)^{1/2} + 1 - d_z sin\phi]} \} \quad . \tag{4.3}$$

From the equation above, one can see that when $\phi$ is constant, the voltage $\Phi$ is a sine wave that is only a function of $\theta$. So we can break $\Phi$ into two parts:

$$\Phi = \Phi_\theta \Phi_\phi \quad , \tag{4.4}$$

where,

$$\Phi_\theta = sin\theta \quad , \tag{4.5}$$

and

$$\Phi_\phi = \frac{1}{4\pi\sigma q} m_y \{ \frac{2cos\phi}{(1-2d_z sin\phi + {d_z}^2)^{3/2}} + \frac{cos\phi}{(1-2d_z sin\phi + {d_z}^2)^{1/2}} + \frac{cos\phi sin\phi d_z}{(1-2d_z sin\phi + {d_z}^2)^{1/2}[(1-2d_z sin\phi + {d_z}^2)^{1/2} + 1 - d_z sin\phi]} \} \quad . \tag{4.6}$$

Plugging q and s back in:

$$\Phi_\theta = sin\theta \quad , \tag{4.7}$$

and

$$\Phi_\phi = cos\phi \{ \frac{1}{4\pi\sigma} m_y [ \frac{2}{q(\phi)^3} + \frac{1}{q(\phi)} + \frac{s}{q(\phi)(q(\phi) + 1 - s)} ] \} \quad . \tag{4.8}$$

Thus one can see more clearly that $\Phi_\theta$ is modulated by $\Phi_\phi$ . These equations may be useful to better interpolate between known potentials gathered from the head. Following that the voltage on the head is created by a superposition of many current dipoles in the head, these two equations could be used to reduce the possible number of ways in which one can intelligently interpolate between measured voltages. Similar methods exist for MEG analysis [20].

## 4.2 SAMPLING THE SURFACE OF THE SHAPES

It is important to note that 100 monopoles per curve were evenly placed according to the arc length. This spacing was also used for the long axis. This is a measure of the density of sources and sinks used. As previously mentioned, it is important that enough sources and sinks be placed on the surface of the antenna elements so that the shape is captured. The values for 40 and 10 monopoles per curve were also calculated. The max deviation was 0.05 and 0.25 mV, respectively. This corresponds to a percent error of 0.23 and 1.1 percent. Thus, it does not seem necessary to have 100 monopoles per curve which is much more time consuming than the lower values.

## 4.3 CONVERGENCE OF THE SOLUTION VS. CONDUCTIVITY DIFFERENCE

This problem arose due to the disparity in conductivity values of the antenna elements and the epoxy but only needed to be addressed for the 3D analysis. From Table 1 it is evident that these conductivity ($\frac{1}{Resistivity}$) values are different by 20 orders of magnitude. When these two materials are juxtaposed together in the simulation it can cause numerical problems. Therefore, a graph was made as can be seen in Figure 34 that shows the point at which at which the solution converges. The curve seems to level off at a value of 4. This means that the antenna element conductivity should be ten thousand times larger than the epoxy conductivity. These values are centered around the sphere conducitivity. Therefore, to obtain

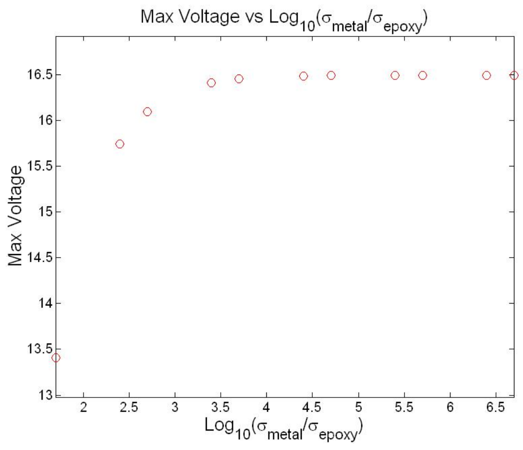


Figure 34: Graph shows that for an increasing conductivity difference, the solution does not change (Appendix .17) .

the antenna element conductivity, the sphere conductivity was multiplied by $\sqrt{10,000}$. To obtain the epoxy conductivity, the sphere conductivity was divided by $\sqrt{10,000}$. Thus, the antenna element conductivity is 10,000 times larger than the epoxy conductivity.

## 4.4 IMPEDANCE

From Figures 35 and 36 one can see that the average voltages required by the antenna are about 70 and 120 mV. For a constant current of 1 mA, the distributed resistance seen by the antenna is about 140 and 240 Ω. These will be important parameters to be verified experimentally. Further, one will have to take these values into account when designing the power source and ciruit to drive signal transmission. It is also important to note that as the curvature is increased, the required source voltage increases. This is because as the antennas close (increase curvature) the current is forced through a smaller area. Therefore, to maintain a constant current of 1 mA, the voltage must increase accordingly.

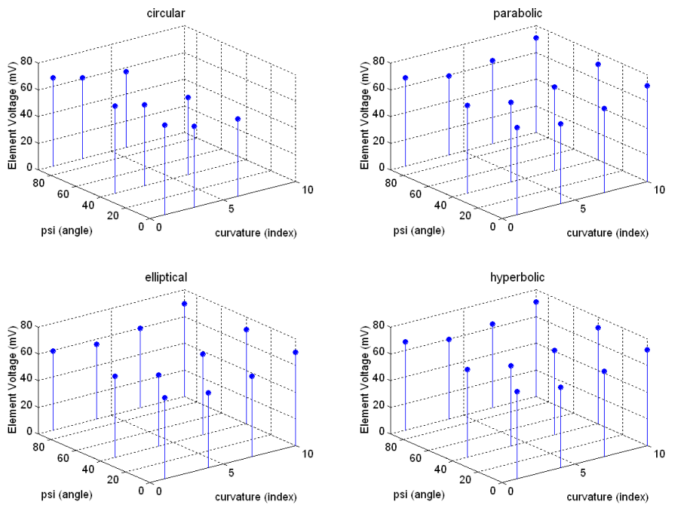


Figure 35: Maximum element voltage plotted versus angle and curvature with no epoxy between the antenna.

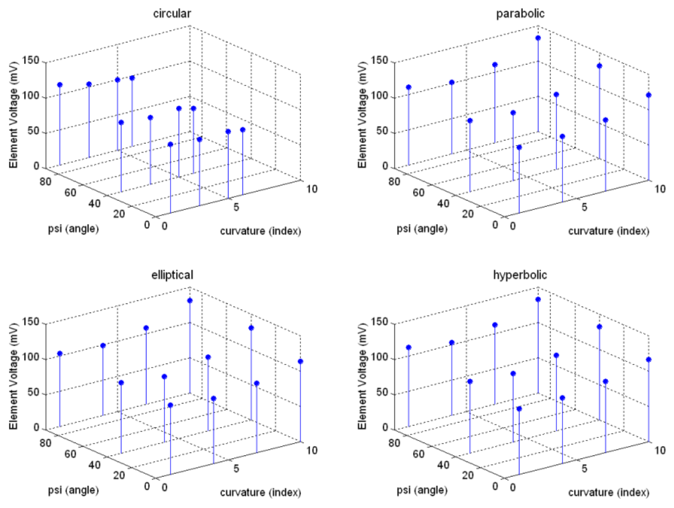


Figure 36: Maximum element voltage plotted versus angle and curvature with epoxy between the antenna.

# 5.0 CONCLUSIONS

In conclusion, two- and three-dimensional finite element analyses show proof of concept that changes to the near field geometry and properties of the antenna can alter the far field results. It was determined that the correct model to analytically simulate the volume conduction antenna was a nonideal dipole. It was also established that the analytical and numerical results were in agreement for similar geometries, therefore the results could be compared and appropriate conclusions drawn when performance changed. The analytical and numerical analysis showed that building an antenna pointed at the surface but slightly away from the other antenna and having large curvature results in the most efficient signal transmission. Numerical results agree but are capable of simulating the antenna more realistically by integrating the epoxy into the analysis. This more realistic analysis yielded a 30-35% increase in surface potential.

# APPENDIX

# MATLAB AND MEX CODE

## .1 GRAPHICAL USER INTERFACE

Contains the following files:

- shapescreator.m
- shapescreator2d.m
- shapesfun.m
- shapesfundersq.m
- shapesgui.m
- shapesnext.m
- shapesnumerical.m
- shapesplt3d.m
- frankpotential.c
- potential.c

```
% shapescreator.m
% Brian Wessel
% creates a shape so that the user can reposition and orient it in space


a = 1;
ndp = get(h15,'Value');
xmax = str2num(get(h37,'String'));
zmax = str2num(get(h39,'String'));
d = str2num(get(h43,'String'));
if ndp==5|ndp==6|ndp==7
a = str2num(get(h47,'String'));
end


clear xpoints
xpoints = [];

delxp = 1/100000;
xp = 0;
lsum = 0;

% chooses the best arc length increment given the xmax and zmax so the shapes has enough points
if xmax<zmax
linc = xmax/2;
else
    linc = zmax/2;
end

i = 1;
fpsq = [];
% numerically determines the x value for a specified unit of arc length
while xp<xmax

    while lsum<linc
        fpsq = shapesfundersq(xp,ndp,a,xmax);
        lsum = lsum+delxp*(1+fpsq)^0.5;
        xp = xp + delxp;
        if xp>xmax
            break
        end
    end

    lsum = 0;

    if xp<xmax
        xpoints(i) = xp;
    end

    i = i+1;
end

% % Shows the x values calculated by the numerical arc length calculation
% figure(70)
% stem(xpoints,xpoints.*xpoints) %only for a parabola
% axis equal
%
% % calculates the real arc length from zero given the x points chosen by arc length
% integralcheck(xpoints) %only works for a parabola

% creates negative values and adds a zero point for the xpoints
nxpoints = -xpoints;
xpoints = [fliplr(nxpoints) 0  xpoints];
length(xpoints);


shapepts = [];
negshapepts = [];
zpoints = -zmax:linc:zmax;
ypoints = [];
% calculates ypoints according to which shape the user specifies.
if ndp == 3

    ypoints = ones(length(xpoints),1)*d/2;

elseif ndp == 4     %circle points

   ypoints = (xmax^2-(xpoints.*xpoints)).^(0.5)-xmax-d/2;

elseif ndp == 5 %parabola pts

   ypoints = a*(xpoints.*xpoints) +d/2;

elseif ndp == 6  %ellipse points

   ypoints = (a^2-a^2/xmax^2*(xpoints.*xpoints)).^(0.5)-a-d/2;

elseif ndp == 7   %hyperbola points

   ypoints = (xmax^2+xmax^2/a^2*(xpoints.*xpoints)).^(0.5)-xmax+d/2;

end

% because a circle and ellipse are closed surfaces this code moves the negative values to the positive side of the axis for those two shapes only
if ndp==6 | ndp==4
    ypoints = -ypoints;
end

ymax = 2*xmax;

for j = 1:length(zpoints)
    for k = 1:length(xpoints)
        if ypoints(k)<(ymax+d)
            shapepts = [shapepts; [xpoints(k) ypoints(k) zpoints(j)]];
            negshapepts = [negshapepts; [xpoints(k) -ypoints(k) zpoints(j)]];
        end
    end
end



% The next 40 lines of code creates a surface from the xyz points
zlin = linspace(min(shapepts(:,3)),max(shapepts(:,3)),50);
xlin = linspace(min(shapepts(:,1)),max(shapepts(:,1)),50);

axes(a12)
[Z,X] = meshgrid(zlin,xlin);
%hold on
Y = griddata(shapepts(:,3),shapepts(:,1),shapepts(:,2),Z,X,'linear');
%h = surf(X,Y,Z);
axis([-8 8 -8 8 -8 8]);
axis on;
xlabel('x');
ylabel('y');
zlabel('z');

nzlin = linspace(min(negshapepts(:,3)),max(negshapepts(:,3)),50);
nxlin = linspace(min(negshapepts(:,1)),max(negshapepts(:,1)),50);

[nZ,nX] = meshgrid(nzlin,nxlin);

axes(a12)
nY = griddata(negshapepts(:,3),negshapepts(:,1),negshapepts(:,2),Z,X);
%h = surf(nX,nY,nZ);
axis([-8 8 -8 8 -8 8]);
axis on;
xlabel('x');
ylabel('y');
```

```
zlabel('z');

XX = [X;nX];
YY = [Y;nY];
ZZ = [Z;nZ];

h = surf(XX,YY,ZZ);
shading interp
axis([-8 8 -8 8 -8 8]);
axis on;
xlabel('x');
ylabel('y');
zlabel('z');
%hold off
```

```
% shapescreator2d.m
% Brian Wessel
% creates shapes for one to inspect the curvature and general
% shape resulting from input parameters

numpts = 81; %should be odd I think
xmax = str2num(get(h37,'String'));
zmax = str2num(get(h39,'String'));
d = str2num(get(h43,'String'));
if ndp==5|ndp==6|ndp==7
a = str2num(get(h47,'String'));
end
% b = str2num(get(h49,'String'))
xpoints = -xmax:(2*xmax)/numpts:xmax;
zpoints = -zmax:(2*zmax)/numpts:zmax;
%sheet points
if ndp == 3

   ypoints = d/2*ones(length(xpoints),1);

end
%circle points (y = (xmax^2-x^2)^.5 +d/2)
if ndp == 4

   ypoints = (xmax^2-(xpoints.*xpoints)).^(0.5)-xmax-d/2;

end

%parabola pts (y = k*x^2 +d/2)
if ndp == 5

   ypoints = a*(xpoints.*xpoints) +d/2;

end
%ellipse points (y = (xmax^2-x^2)^.5 -d/2)
if ndp == 6

   ypoints = (a^2-a^2/xmax^2*(xpoints.*xpoints)).^(0.5)-a-d/2;

end
%hyperbola points (y = (xmax^2+xmax^2/a^2*(x^2))^0.5-a+d/2)
if ndp == 7

   ypoints = (xmax^2+xmax^2/a^2*(xpoints.*xpoints)).^(0.5)-xmax+d/2;

end

datapts=[];
if ndp == 4 | ndp==6
    ypoints = -ypoints;
end
ymax = 2*xmax;
% +1 is added to numpts because when I create xpoints and zpoints I go from xmax:(2*xmax)/numpoints:xmax
% (I don't subtract 2*xmax/numpoints from the end to make it the same number of pts as numpts)
for j = 1:numpts+1
     for k = 1:numpts+1
         if ypoints(k)<(ymax+d)
            datapts = [datapts; [xpoints(k) ypoints(k) zpoints(j)]];
        end
     end
end



% creates a scatter plot to display the data
% axes(a14)
% scatter3(datapts(:,1),datapts(:,2),datapts(:,3));
% axis([-2*xmax 2*xmax -2*max(datapts(:,2)) 2*max(datapts(:,2)) -2*zmax 2*zmax])
% xlabel('x');
% ylabel('y');
% zlabel('z');
% hold on
% scatter3(datapts(:,1),-datapts(:,2),datapts(:,3));
% grid on
% axis equal
% axis([-4*xmax 4*xmax -2*max(datapts(:,2)) 2*max(datapts(:,2)) -2*zmax 2*zmax])
% hold off

negdatapts = [datapts(:,1) -datapts(:,2) datapts(:,3)];
% The next 40 lines of code creates a surface from the xyz points
zlin = linspace(min(datapts(:,3)),max(datapts(:,3)),50);
xlin = linspace(min(datapts(:,1)),max(datapts(:,1)),50);
axes(a14)
[Z,X] = meshgrid(zlin,xlin);
Y = griddata(datapts(:,3),datapts(:,1),datapts(:,2),Z,X,'linear');
surf(X,Y,Z);
axis([-8 8 -8 8 -8 8]);
axis on;
xlabel('x');
ylabel('y');
zlabel('z');
hold on
nzlin = linspace(min(negdatapts(:,3)),max(negdatapts(:,3)),50);
nxlin = linspace(min(negdatapts(:,1)),max(negdatapts(:,1)),50);
[nZ,nX] = meshgrid(nzlin,nxlin);
nY = griddata(negdatapts(:,3),negdatapts(:,1),negdatapts(:,2),Z,X);
surf(nX,nY,nZ);
axis([-8 8 -8 8 -8 8]);
axis on;
xlabel('x');
ylabel('y');
zlabel('z');
axis equal
axis([-4*xmax 4*xmax -2*max(datapts(:,2)) 2*max(datapts(:,2)) -2*zmax 2*zmax])
hold off
shading interp
% XX = [X;nX];
% YY = [Y;nY];
% ZZ = [Z;nZ];
%
% h = surf(XX,YY,ZZ);
% axis([-8 8 -8 8 -8 8]);
% axis on;
% xlabel('x');
% ylabel('y');
% zlabel('z');
```

```
% shapesfun.m
% Brian Wessel
% outputs a point for input parameters
function f = shapesfun(xx,n,a,xmax,d)

if n == 3  %sheet points
    f = d/2;
elseif n == 4,%circle points (y = (xmax^2-x^2)^.5 +d/2)
    f = (xmax^2-(xx.*xx)).^(0.5)-xmax-d/2;
elseif n == 5%parabola pts (y = k*x^2 +d/2)n == 5,
    f = a*(xx.*xx)+d/2;
elseif n == 6,   %ellipse points (y = (xmax^2-x^2)^.5 -d/2)
    f = (a^2-a^2/xmax^2*(xx.*xx)).^(0.5)-a-d/2;
elseif n == 7, %hyperbola points (y = (xmax^2+xmax^2/a^2*(x^2))^0.5-a+d/2)
    f = (xmax^2+xmax^2/a^2*(xx.*xx)).^(0.5)-xmax+d/2;
end
```

```
% shapesfundersq.m
% Brian Wessel
% Calculates the square root of the derivative
% of the function evaluated at a point

function f = shapesfundersq(xx,ndp,a,xmax);

%sheet points
if ndp == 3

    f = 0;

end

%circle points
if ndp == 4

    if xmax==xx
        f = 0;
    else
    f = xx^2/(xmax^2 - xx^2);
    end

end


%parabola pts
if ndp == 5

    f = a^2*4*xx^2;

end


%ellipse points
if ndp == 6

    f = xx^2*xmax^2/a^2*(a^2-xx^2)^-1;

end



%hyperbola points
if ndp == 7

    f = xx^2*xmax^2/a^2*(a^2+xx^2)^-1;

end
```

```
% shapesgui.m
% Brian Wessel
% Began May 2002
% Main control for the GUI

headrad = 1; % proportionality factor to scale the head values

[X Y Z]=sphere(10);
t=2*(0:1/128:1);
t1=0:1/128:1;
theta=pi*t1';
alpha=pi*t;
xs=headrad*sin(theta)*cos(alpha);
ys=headrad*sin(theta)*sin(alpha);
zs=headrad*cos(theta)*ones(1,129);

px=xs(1:2:129,1:2:129);
py=ys(1:2:129,1:2:129);
pz=zs(1:2:129,1:2:129);
x=cos(alpha);
y=sin(alpha);
e(1:3:387)=xs(64,:);
e(2:3:387)=ys(64,:);
e(3:3:387)=zeros(1,129);
clear center

figure('position',[-100 -500 1000 1000]);axis('off');


%h20=uicontrol('style','frame','position',[422 310 455 190]);

h1=uicontrol('style','pushbutton', 'position',...
[150 455 100 20],'string','place dipole(s)', 'callback', 'shapesnext');

h15=uicontrol('style','popupmenu', 'position', [430 455 100 20],'string','1 dipole|2 dipoles|sheet|circular|parabolic|elliptical|hyperbolic','backgroundcolor',[0 1 0],...
'callback','ndp = get(h15,''Value'');if ndp==1|ndp==2,set([h37 h39 h43 h47],''Backgroundcolor'',[0 0 0]),end,if ndp==3|ndp==4,set([h37 h39 h43],''Backgroundcolor'',[0.2,1,0.9]),set([h47],''Backgroundcolor

ndp = 1;

slider1 = 0;
slider2 = 0;
slider3 = 0;
slider1old = 0;
slider2old = 0;
slider3old = 0;
inc1 = 0;
inc2 = 0;
inc3 = 0;

h19=uicontrol('style','text', 'position', [880 385 340 18],'string','ORIENTATION (DEGREES)','FontSize',10,'backgroundcolor',[0,.7,0.9]);

h14=uicontrol('style','slider','position',[880 350 100 30],'Min',0,'Max',360,'SliderStep',[0.0027777777777777777 0.0277777777777777777],...
    'callback','slider1 = get(h14,''Value'');,inc1=slider1-slider1old;,h19=uicontrol(''style'',''text'', ''position'', [880 330 100 15],''string'',slider1,''backgroundcolor'',[0,.7,0.9]);,rotate(h,[1 0 0]
h19=uicontrol('style','text', 'position', [880 330 100 15],'string',slider1,'backgroundcolor',[0,.7,0.9]);

h25=uicontrol('style','slider','position',[1000 350 100 30],'Min',0,'Max',360,'SliderStep',[0.0027777777777777777 0.0277777777777777777],...
    'callback','slider2 = get(h25,''Value'');,inc2=slider2-slider2old;,h26=uicontrol(''style'',''text'', ''position'', [1000 330 100 15],''string'',slider2,''backgroundcolor'',[0,.7,0.9]);,rotate(h,[0 1 0
h26=uicontrol('style','text', 'position', [1000 330 100 15],'string',slider2,'backgroundcolor',[0,.7,0.9]);

h27=uicontrol('style','slider','position',[1120 350 100 30],'Min',0,'Max',360,'SliderStep',[0.0027777777777777777 0.0277777777777777777],...
    'callback','slider3 = get(h27,''Value'');,inc3=slider3-slider3old;,h28=uicontrol(''style'',''text'', ''position'', [1120 330 100 15],''string'',slider3,''backgroundcolor'',[0,.7,0.9]);,rotate(h,[0 0 1
h28=uicontrol('style','text', 'position', [1120 330 100 15],'string',slider3,'backgroundcolor',[0,.7,0.9]);


slider4 = 0;
slider5 = 0;
slider6 = 0;
slider4old = 0;
slider5old = 0;
slider6old = 0;
inc4 = 0;
inc5 = 0;
inc6 = 0;

h29=uicontrol('style','text', 'position', [880 465 340 18],'string','POSITION (CM)','FontSize',10,'backgroundcolor',[0,.7,0.9]);

h30=uicontrol('style','slider','position',[880 430 100 30],'Min',0,'Max',10,'SliderStep',[0.01 0.1],...
    'callback','slider4 = get(h30,''Value'');,inc4=slider4-slider4old;,h31=uicontrol(''style'',''text'', ''position'', [880 410 100 15],''string'',slider4,''backgroundcolor'',[0,.7,0.9]);,set(h,''XData'',
h31=uicontrol('style','text', 'position', [880 410 100 15],'string',slider4,'backgroundcolor',[0,.7,0.9]);

h32=uicontrol('style','slider','position',[1000 430 100 30],'Min',0,'Max',10,'SliderStep',[0.01 0.1],...
    'callback','slider5 = get(h32,''Value'');,inc5=slider5-slider5old;,h33=uicontrol(''style'',''text'', ''position'', [1000 410 100 15],''string'',slider5,''backgroundcolor'',[0,.7,0.9]);,set(h,''YData''
h33=uicontrol('style','text', 'position', [1000 410 100 15],'string',slider5,'backgroundcolor',[0,.7,0.9]);

h34=uicontrol('style','slider','position',[1120 430 100 30],'Min',0,'Max',10,'SliderStep',[0.01 0.1],...
    'callback','slider6 = get(h34,''Value'');,inc6=slider6-slider6old;,h35=uicontrol(''style'',''text'', ''position'', [1120 410 100 15],''string'',slider6,''backgroundcolor'',[0,.7,0.9]);,set(h,''ZData''
h35=uicontrol('style','text', 'position', [1120 410 100 15],'string',slider6,'backgroundcolor',[0,.7,0.9]);


h50=uicontrol('style','text', 'position', [540 285 100 25],'string','black = not required  turquoise = required');

%create shape
h36=uicontrol('style','text', 'position', [540 475 100 15],'string','xmax','backgroundcolor',[0,0.7,0.9]);
h37=uicontrol('style','edit', 'position', [540 455 100 20],'backgroundcolor',[0.2,1,0.9]);

h38=uicontrol('style','text', 'position', [540 430 100 15],'string','zmax','backgroundcolor',[0,0.7,0.9]);
h39=uicontrol('style','edit', 'position', [540 410 100 20],'backgroundcolor',[0.2,1,0.9]);

h42=uicontrol('style','text', 'position', [540 385 100 15],'string','d','backgroundcolor',[0,.7,0.9]);
h43=uicontrol('style','edit', 'position', [540 365 100 20],'backgroundcolor',[0.2,1,0.9]);

h46=uicontrol('style','text', 'position', [540 340 100 15],'string','a','backgroundcolor',[0,.7,0.9]);
h47=uicontrol('style','edit', 'position', [540 320 100 20],'backgroundcolor',[0.2,1,0.9]);

set([h37 h39 h43 h47],'Backgroundcolor',[0 0 0]);

h45=uicontrol('style','pushbutton', 'position',...
[770 455 100 20],'string','load data', 'callback', 'shapescreator');

h44=uicontrol('style','pushbutton', 'position',...
[660 455 100 20],'string','update', 'callback', 'shapescreator2d');

h48=uicontrol('style','pushbutton', 'position',...
[770 330 100 20],'string','calculate potential', 'callback', 'shapesnumerical');


h3=uicontrol('style','pushbutton', 'position',...
[430 430 100 20],'string','new window', 'callback', 'shapesgui');

a1=axes('position', [0.03 0.56 .25 .45]);axis('off');
radi = headrad*ones(length(theta),1);
polar(2*theta,radi,'--r')
hold on;
polar(2*theta,radi*0.92,'--r')
polar(2*theta,radi*0.87,'--r')
xval = [0 10];
yval = [0 0];
text(11.5,0,'X','FontSize',15);
text(6.5,12,'<-- POSITION -->','FontSize',15);
text(0,12,'Y','FontSize',15);
plot(yval,xval);
plot(xval,yval);
hold off;

a12=axes('position', [0.68 0.6 .3 .4]);axis('off');
axis([-8 8 -8 8 -8 8])
xlabel('x');
ylabel('y');
```

```
zlabel('z');
rotate3d;
center = [0 0 0];
a14=axes('position', [0.35 0.6 .3 .4]);axis('off');

a13=axes('position', [0.62 0.02 .35 .35]);axis('off');
xlabel('x');
ylabel('y');
zlabel('z');
rotate3d
a18=axes('position', [0.02 0.20 .3 0.3]);axis('off');
rotate3d
```

```
% shapesnext.m
% Dr. Sun created this code.
% Used to create the polar plots on the
% left side of the GUI

axes(a1);

polar(2*theta,radi,'--r')
 hold on;
 polar(2*theta,radi*0.92,'--r')
 polar(2*theta,radi*0.87,'--r')

if ndp<3
[xx, yy]=ginput(2*ndp);
[xx yy];

if ndp ==1,
 d=[xx(1),yy(1),0];
 m=[xx(2)-xx(1),yy(2)-yy(1),0];
  p=potential(129,d,m,e);
elseif ndp ==2,
 d1=[xx(1),yy(1),0];
 m1=[xx(2)-xx(1),yy(2)-yy(1),0];
  p1=potential(129,d1,m1,e);
 d2=[xx(3),yy(3),0];
 m2=[xx(4)-xx(3),yy(4)-yy(3),0];
  p2=potential(129,d2,m2,e);
 p=p1+p2;
end

axes(a1);

 polar(alpha',headrad*(0.01*p+ones(size(p))),'r');
 hold on;
 uu=get(gca,'children');
 set(uu(1),'linewidth',2);

 plot(xx(1),yy(1),'o');
 plot(xx(1:2),yy(1:2),'linewidth',2);


 if ndp == 2,
 plot(xx(3),yy(3),'o');
 plot(xx(3:4),yy(3:4),'linewidth',2);
 end

 hold off;

 %axis([-1.5 1.5 -1.5 1.5]);
 %axes(a2);
 %plot(t,p);
 %grid;

 %axes(a3);
if ndp ==1,
 d=[xx(1),yy(1),0];
 m=[xx(2)-xx(1),yy(2)-yy(1),0];
for i=1:129
elec(1:3:387)=xs(i,:);
elec(2:3:387)=ys(i,:);
elec(3:3:387)=zs(i,:);
  pot(:,i)=potential(129,d,m,elec);
end
elseif ndp ==2,
for i=1:129
elec(1:3:387)=xs(i,:);
elec(2:3:387)=ys(i,:);
elec(3:3:387)=zs(i,:);
 d1=[xx(1),yy(1),0];
 m1=[xx(2)-xx(1),yy(2)-yy(1),0];
  pot1(:,i)=potential(129,d1,m1,elec);
 d2=[xx(3),yy(3),0];
 m2=[xx(4)-xx(3),yy(4)-yy(3),0];
  pot2(:,i)=potential(129,d2,m2,elec);
end
 pot=pot1+pot2;
end
%imagesc(t1,t,pot);
%xlabel('angle to z-axis (*pi)');
%ylabel('angle to x-axis (*pi)');
grid;
pv=pot(1:2:129,1:2:129);



elseif ndp>2
    [xx, yy]=ginput(1);
[xx,yy];
hold on;
%plot(xx(1),yy(1),'o');
hold off;
end

% call to plot the dipole placement(s) potential in 3D
shapesplt3d
```

```
% shapesnumerical .m
% Brian Wessel
% Used to create the input to calculate the surface
% potential given the shape of the sphere

%initialize variables
headradnew = 1;
%headradnew = 7.5;
ndp = get(h15,'Value');
xmax = str2num(get(h37,'String'));
d = str2num(get(h43,'String'));
if ndp==5|ndp==6|ndp==7
a = str2num(get(h47,'String'));
end
bothshapepts = [shapepts;negshapepts];
bothshapepts = [];
centershapepts = [];
negcentershapepts = [];
unit = [];
bothcentershapepts = [];
centerunit = [];
normvec = [];

% The following is the way in which  the next code reads the data
% aa  bb
% dd  cc
lxpt = length(xpoints);
lshpts = length(shapepts);
j=0;
%Calculates the normal vectors of the positive section of the antenna and also the center points where these normal vectors are located
for i = 1:(length(shapepts)-length(xpoints))
    if mod(i,length(xpoints))~=0
       j = j+1;
        aa = shapepts(i,:);
        bb = shapepts(i+1,:);
        cc = shapepts(i+1+lxpt,:);
        dd = shapepts(i+lxpt,:);
        aacc = cc-aa;
        ddbb = bb-dd;
        centershapepts = [centershapepts; (aa(1)+bb(1))/2  shapesfun((aa(1)+bb(1))/2,ndp,a,xmax,d)  (aa(3)+dd(3))/2];
        normvec(j,:) = cross(aacc,ddbb);
    end
end

negnormvec = [];
nlxpt = length(xpoints);
j=0;
%Calculates the normal vectors of the negative section of the antenna and also the center points where these normal vectors are located
for i = 1:(length(negshapepts)-length(xpoints))
    if mod(i,length(xpoints))~=0
       j = j+1;
        aa = negshapepts(i,:);
        bb = negshapepts(i+1,:);
        cc = negshapepts(i+1+lxpt,:);
        dd = negshapepts(i+lxpt,:);
        aacc = cc-aa;
        ddbb = bb-dd;
        negcentershapepts = [negcentershapepts; (aa(1)+bb(1))/2 -shapesfun((aa(1)+bb(1))/2,ndp,a,xmax,d) (aa(3)+dd(3))/2];
        negnormvec(j,:) = cross(aacc,ddbb);
    end
end

unitnormvec = [];
negunitnormvec = [];
%creates unit vectors from the normal vectors (normvec)
for i = 1:length(normvec(:,1))
    magnormvec          = ( (normvec(i,1))^2+(normvec(i,2))^2+(normvec(i,3))^2 )^0.5;
    unitnormvec(i,:)    = normvec(i,:)/magnormvec;
    negmagnormvec       =  ( (negnormvec(i,1))^2+(negnormvec(i,2))^2+(negnormvec(i,3))^2 )^0.5;
    negunitnormvec(i,:) = negnormvec(i,:)/negmagnormvec;
end

% switch placement for a circle and ellipse since I have to split them in half and pull them above and below the x-axis
% this code also consolidates the positive and negative unit normal vectors into a variable called unit
if ndp==4 | ndp == 6
    unit(:,1) = [negunitnormvec(:,1);unitnormvec(:,1)];
    unit(:,2) = [negunitnormvec(:,2);unitnormvec(:,2)];
    unit(:,3) = [negunitnormvec(:,3);unitnormvec(:,3)];
else
    unit(:,1) = [unitnormvec(:,1);negunitnormvec(:,1)];
    unit(:,2) = [unitnormvec(:,2);negunitnormvec(:,2)];
    unit(:,3) = [unitnormvec(:,3);negunitnormvec(:,3)];
end

% consolidates the positive and negaitive center shape points and combines this with the unit vectors
% this is done so that all the points can be rotated at once
bothcentershapepts = [centershapepts;negcentershapepts];
centerunit = [bothcentershapepts;[unit(:,1) unit(:,2) unit(:,3)]];

% read in the angles
theta_r = 3.1415*1/180*get(h14,'Value');
phi_r   = 3.1415*1/180*get(h25,'Value');
psi_r   = 3.1415*1/180*get(h27,'Value');
%rotation matrices
temppts = [centerunit(:,1) centerunit(:,2)]*[cos(psi_r) sin(psi_r);-sin(psi_r) cos(psi_r)];
centerunit(:,1) = temppts(:,1);
centerunit(:,2) = temppts(:,2);
temppts = [centerunit(:,2) centerunit(:,3)]*[cos(theta_r) sin(theta_r);-sin(theta_r) cos(theta_r)];
centerunit(:,2) = temppts(:,1);
centerunit(:,3) = temppts(:,2);
temppts = [centerunit(:,3) centerunit(:,1)]*[cos(phi_r) sin(phi_r);-sin(phi_r) cos(phi_r)];
centerunit(:,3) = temppts(:,1);
centerunit(:,1) = temppts(:,2);

lcu = length(centerunit);
% separates the points from unit vectors and also adds the center values to the corresponding center shape points so
% that repositioning can be accomplished
bothcentershapepts = [(centerunit(1:lcu/2,1)) (centerunit(1:lcu/2,2)) (centerunit(1:lcu/2,3))];
bothcentershapepts(:,1) = bothcentershapepts(:,1)+center(1,1);
bothcentershapepts(:,2) = bothcentershapepts(:,2)+center(1,2);
bothcentershapepts(:,3) = bothcentershapepts(:,3)+center(1,3);

unit = [centerunit(lcu/2+1:lcu,1) centerunit(lcu/2+1:lcu,2) centerunit(lcu/2+1:lcu,3)];
%end reorienting and repositioning

% plots the shapes and corresponding unit vectors
% creates a scale factor which accounts for errors in the coding of quiver3...the error(s) is that when the object is moved
% in the y or z direction to a higher value, the vectors become longer in length.  Thus, this code determines if the user has
% selected to move the shape to a high value in these directions and scales down.  Note that movement in the x direction does
% nothing and if the object is moved in both the y and z direction, it has a cancelling effect and only small scaling (0.5) is
% needed.  This follows xor logic for the variables y and z
if center(1,2)>2 & center(1,3)<2
    sf = d/2;
elseif center(1,2)<2 & center(1,3)>2
    sf = d/2;
else sf = 0.5;
```

```
end

% don't need this now that I'm not using dipoles
% axes(a14)
% quiver3(bothcentershapepts(:,1),bothcentershapepts(:,2),bothcentershapepts(:,3),unit(:,1),unit(:,2),unit(:,3),sf)
% axis equal
% hold on
% scatter3(bothcentershapepts(:,1),bothcentershapepts(:,2),bothcentershapepts(:,3),15,'--r');
% xlabel('x');
% ylabel('y');
% zlabel('z');
% xmin = min(bothcentershapepts(:,1));
% xmax = max(bothcentershapepts(:,1));
% ymin = min(bothcentershapepts(:,2));
% ymax = max(bothcentershapepts(:,2));
% zmin = min(bothcentershapepts(:,3));
% zmax = max(bothcentershapepts(:,3));
% axis([(xmin-(xmax-xmin)/2) (xmax+(xmax-xmin)/2) (ymin-(ymax-ymin)/2) (ymax+(ymax-ymin)/2) (zmin-(zmax-zmin)/2) (zmax+(zmax-zmin)/2)])
% hold off

% % used this code to make a figure for a paper
% figure(69)
% quiver3(bothcentershapepts(:,1),bothcentershapepts(:,2),bothcentershapepts(:,3),unit(:,1),unit(:,2),unit(:,3),sf)
% axis equal
% hold on
% scatter3(bothcentershapepts(:,1),bothcentershapepts(:,2),bothcentershapepts(:,3),15,'--r');
% xlabel('x (cm)');
% ylabel('y (cm)');
% zlabel('z (cm)');
% xmin = min(bothcentershapepts(:,1));
% xmax = max(bothcentershapepts(:,1));
% ymin = min(bothcentershapepts(:,2));
% ymax = max(bothcentershapepts(:,2));
% zmin = min(bothcentershapepts(:,3));
% zmax = max(bothcentershapepts(:,3));
% %axis([(xmin-(xmax-xmin)/2) (xmax+(xmax-xmin)/2) (ymin-(ymax-ymin)/2) (ymax+(ymax-ymin)/2) (zmin-(zmax-zmin)/2) (zmax+(zmax-zmin)/2)])
% hold off

% code to calculate the normal vectors according to MatLab - doesn't do a very good job
%     newxpts = [];
%     newypts = [];
%     newzpts = [];
% for i = 0:(length(centershapepts)/(length(xpt)-1)-1)
%     newxpts = [newxpts, centershapepts((i*(length(xpt)-1)+1):(i+1)*(length(xpt)-1),1)];
%     newypts = [newypts, centershapepts((i*(length(xpt)-1)+1):(i+1)*(length(xpt)-1),2)];
%     newzpts = [newzpts, centershapepts((i*(length(xpt)-1)+1):(i+1)*(length(xpt)-1),3)];
% end
%
% figure(5)
% [Nx,Ny,Nz] = surfnorm(newxpts',newypts',newzpts');
% surfnorm(newxpts',newypts',newzpts');
% axis equal

% % % % UNCOMMENT THE NEXT 60 LINES OR SO TO USE POTENTIAL.C
% % % % code to call vector form homogeneous solution
% % % clear totpoten
% % % clear poten
% % %
% % % %sets up parametric values to be read into potential.meglx
% % % %t=2*(0:1/128:(1-1/128)); 128 pts code
% % % t=2*(0:1/128:1);
% % %
% % % %t1=0:1/128:0.99999;  128 pts code
% % % t1=0:1/128:1;
% % %
% % % theta=pi*t1';
% % % alpha=pi*t;
% % % xs=headradnew*sin(theta)*cos(alpha);
% % % ys=headradnew*sin(theta)*sin(alpha);
% % % %zs=headradnew*cos(theta)*ones(1,128); 128 pts code
% % % zs=headradnew*cos(theta)*ones(1,129);
% % %
% % % poten = [];
% % % clear totpoten
% % % %totpoten = zeros(128); 128 pts code
% % % totpoten = zeros(129);
% % %
% % % %testpoten = potential(1,[0 0 0],[0 0 0],[1 1 1]);
% % %
% % % for j = 1:length(unitnormvec)
% % % %for j = 1:1
% % %     for i=1:129
% % %         elec(1:3:387)=xs(i,:);
% % %         elec(2:3:387)=ys(i,:);
% % %         elec(3:3:387)=zs(i,:);
% % %         poten(:,i) = potential(129,bothcentershapepts(j,:),unit(j,:),elec);
% % %     end
% % %     totpoten = totpoten+poten;
% % % end
% % %
% % % %create a dummy variable to be graphed so as to reinitialize axes a13
% % % testpoten = zeros(129);
% % % axes(a13)
% % % pv=testpoten(1:2:129,1:2:129);
% % % px=xs(1:2:129,1:2:129);py=ys(1:2:129,1:2:129);pz=zs(1:2:129,1:2:129);
% % % surface(px,py,pz,pv);
% % %
% % % %creat the real plot
% % % %axes(a13)
% % % figure(3)
% % % pv = [];
% % % %imagesc(t1,t,totpoten);
% % % pv=totpoten(1:2:129,1:2:129);
% % % px=xs(1:2:129,1:2:129);py=ys(1:2:129,1:2:129);pz=zs(1:2:129,1:2:129);
% % % surface(px,py,pz,pv');
% % % rotate3d;
% % % grid on;
% % % shading interp
% % % colormap('jet');
% % % %colorbar
% % % xlabel('x (cm)','FontSize',15);
% % % ylabel('y (cm)','FontSize',15);
% % % zlabel('z (cm)','FontSize',15);
% % % set(gca,'Visible','On');
% % % rotate3d


% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %
% % % Using Ernest Frank's arbitrary placed and spaced dipole (this code does not use the Mex function, thus it is slow)
% %
% % clear Vr ra rb theta phi Px Py Pz theta phi
% %
% % % radius of the sphere
% % R = 7.5;
% %
% % % (n+1)^2 will be the number of points on the sphere
% % n = 30;
% %
% % theta = pi*(-n:2:n)/n;
% % phi = (pi/2)*(-n:2:n)'/n;
% % Px = R*cos(phi)*cos(theta);
% % Py = R*cos(phi)*sin(theta);
% % Pz = R*sin(phi)*ones(1,length(sin(phi)));
% %
% %
% % % figure(1)
% % % plot3(Px(:),Py(:),Pz(:))
% % % title('points to find the potenial at')
% %
% % % current (in mA) I divide by the number of simulated non-ideal dipoles so that when I them all up, the resulting voltage is from
% % % that total amount of current...so in this case, the total current is 1 mA
% % I = 1/(length(centershapepts))^2;
% %
% % % the conductivity in [1/(ohms*cm)]
% % gamma = 1/222;
% %
% % % dipole placement (must adhere to the constraints of Ernest Frank's assumptions) (They must be in the x-z plane)
% % % --actually, his equation is general, he just uses the symmetry of the xz plane for ease of derivation
% %
% % % % realistic case
% % % neg_i = [0.997 0 5.79]/7.5;
% % % pos_i = [0   0 5.875]/7.5;
% %
% % % power = 1;
% % % test case
% % % neg_i = [1*10^(-power) 0 1*10^(-power-2)];
% % % pos_i = [1*10^(-power)   0 1*10^(-(power+1))];
% % % neg_i = [-0.05 -0.05 0];
% % % pos_i = [0 0 0.05];
% % % a = norm(neg_i);
% % % b = norm(pos_i);
% %
% % % the distance between the source and sink - calculate this to ensure that both moments are the same
% % % dab = (a^2+b^2-2*dot(neg_i,pos_i))^(0.5);
% %
% % Vr_tot = zeros(length(Px));
% %
% % % centershapepts = [0.9 0.0 0.0; 0.9 0.9 0];
% % % negcentershapepts = [0.0 0.9 0.0; -0.9 -0.9 0];
% %
% % for k = 1:(length(bothcentershapepts)/2)
% %     pos_i = bothcentershapepts(k,:);
% %     neg_i = bothcentershapepts(k+length(bothcentershapepts)/2,:);
% %     a = norm(neg_i);
% %     b = norm(pos_i);
% %     for i = 1:length(Px)
% %         for j = 1:length(Px)
% %             ra(i,j) = ((Px(i,j)-neg_i(1,1))^2 +(Py(i,j)-neg_i(1,2))^2 +(Pz(i,j)-neg_i(1,3))^2)^0.5;
% %             rb(i,j) = ((Px(i,j)-pos_i(1,1))^2 +(Py(i,j)-pos_i(1,2))^2 +(Pz(i,j)-pos_i(1,3))^2)^0.5;
% %             p = [Px(i,j) Py(i,j) Pz(i,j)];
% %             sink   = [neg_i(1,1) neg_i(1,2) neg_i(1,3)];
% %             source = [pos_i(1,1) pos_i(1,2) pos_i(1,3)];
% %
% %             costheta = dot(p,source)/(norm(p)*norm(source));
% %             cosbeta  = dot(p,sink)/(norm(p)*norm(sink));
% %
% %             Vr(i,j) = I/(4*pi*gamma)*[2/rb(i,j)-2/ra(i,j)+1/R*log((ra(i,j)+R-a*cosbeta)/(rb(i,j)+R-b*costheta))] ;
% %         end
% %     end
% %     Vr_tot = Vr_tot + Vr;
% % end
% %
% % figure(3)
% % %axes(a13)
% % subplot(2,2,1)
% % surf(Px,Py,Pz,Vr_tot);
% % title('Franks equation - m code')
% % colorbar
% % xlabel('x (cm)');
% % ylabel('y (cm)');
% % zlabel('z (cm)');
% % axis equal
% % shading interp
% %
% % figure(3)
% % %axes(a13)
% % subplot(2,2,3)
% % imagesc(theta,flipud(phi),Vr_tot);
% % xlabel('theta (radians)');
% % ylabel('phi (radians)');
% % title('Franks equation')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%This code calls frankpotential.dll to increase the speed of processing

clear Vr ra rb theta phi Px Py Pz theta phi

% radius of the sphere
R = 7.5;

% (n+1)^2 will be the number of points on the sphere
n = 500;

% channel density (not used as an input but is interesting to know
channeldensity = (n+1)^2/(4*pi*R^2)

theta = pi*(-n:2:n)/n;
phi = (pi/2)*(-n:2:n)'/n;
Px = R*cos(phi)*cos(theta);
Py = R*cos(phi)*sin(theta);
Pz = R*sin(phi)*ones(1,length(sin(phi)));
P = [Px,Py,Pz];

% figure(1)
% plot3(Px(:),Py(:),Pz(:))
% title('points to find the potenial at')

% centershapepts = [0.9 0.0 0.0; 0.9 0.9 0];
% negcentershapepts = [0.0 0.9 0.0; -0.9 -0.9 0];

% current (in mA) I divide by the number of simulated non-ideal dipoles so that when I them all up, the resulting voltage is from
% that total amount of current...so in this case, the total current is 1 mA
I = 1/(length(centershapepts))^2;

% % realistic case
% neg_i = [0.997 0 5.79]/7.5;
% pos_i = [0   0 5.875]/7.5;

% power = 1;
% test case
% neg_i = [1*10^(-power) 0 1*10^(-power-2)];
% pos_i = [1*10^(-power)   0 1*10^(-(power+1))];
% neg_i = [-0.05 -0.05 0];
% pos_i = [0 0 0.05];
% a = norm(neg_i);
% b = norm(pos_i);

k = length(bothcentershapepts);

Vr = frankpotential(I,bothcentershapepts(k/2+1:k,:),bothcentershapepts(1:k/2,:),P,R);


%figure(3)
axes(a13)
%subplot(2,2,2)
surf(Px,Py,Pz,Vr);
%title('Franks equation - c code')
colorbar
xlabel('x (cm)');
ylabel('y (cm)');
zlabel('z (cm)');
axis equal
shading interp

% figure(3)
% %axes(a13)
% subplot(2,2,4)
% imagesc(theta,flipud(phi),Vr);
% xlabel('theta (radians)');
% ylabel('phi (radians)');
% title('Franks equation')

% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% % % probably won't need this code at all
% %
% % % new code to call inhomogeneous (4 shell) model
% %
% % clear poten
% % clear totpoten
% %
% % MXPTS = 11;
% % increment = 2*3.14/MXPTS;
% %
% % a = fix(MXPTS);
% %
% % x = [];
% % y = [];
% % z = [];
% % for t = 0:(MXPTS-1)
% %     for p = 0:(a-1)
% %         theta = increment*t;
% %         phi = increment*p;
% %
% %         x = [x,cos(phi)*sin(theta)];
% %         y = [y,sin(phi)];
% %         z = [z,cos(phi)*cos(theta)];
% %     end
% % end
% %
% % figure(99)
% % scatter3(x,y,z);
% %
% % % x = [0 0 0 0 0 0];
% % % y = [1 1 1 1 1 1];
% % % z = [2 2 2 2 2 2];
% %
% % totpoten = zeros(6,1);
% %
% % for j = 1:length(bothcentershapepts)
% % %for j = 1:6
% %    poten = pot_vec_order3(121,bothcentershapepts(j,:),unit(j,:),[x' y' z']);
% %    totpoten = totpoten+poten;
% % end
% shapesplt3d.m
% Brian Wessel
% Used to plot the solution in the frame
% below the polar plot

axes(a18)
rotate3d
surface(px,py,pz,pv');
shading interp
colormap('jet');
xlabel('x');
ylabel('y');
zlabel('z');
grid on;
rotate3d;
set(gca,'Visible','On')
```

```c
/* frankpotential.c
   Brian Wessel     */
#include "mex.h"
#include "math.h"

#define PI 4.*atan(1.0)

/*
 * frankpotential.c --Calculates the potential on the scalp for a given set of non-ideal dipoles
 *
*/

double norm(double a, double b, double c)
{
    return(  sqrt(  pow(a,2)+pow(b,2)+pow(c,2)  )  );
}

void frankpotential(double cur, double *neg, double *pos,double *chpts, double *pot,int l,int m,double R)
{
  int i,j,countin=0,countout=0;

double normsource,normsink,costheta,cosbeta,chptdotsink,chptdotsource,normchpt,rb,ra,a,b;

  for (i=0;i<m;i++){

     normsink =   norm(*(neg+countout),*(neg+countout+m),*(neg+countout+2*m));
     normsource = norm(*(pos+countout),*(pos+countout+m),*(pos+countout+2*m));

    for (j=0;j<l*l;j++){
        ra = norm(*(chpts+countin)-*(neg+countout),*(chpts+countin+l*l)-*(neg+countout+m),*(chpts+countin+2*l*l)-*(neg+countout+2*m));
            rb = norm(*(chpts+countin)-*(pos+countout),*(chpts+countin+l*l)-*(pos+countout+m),*(chpts+countin+2*l*l)-*(pos+countout+2*m));
        a  = norm(*(neg+countout),*(neg+countout+m),*(neg+countout+2*m));
        b  = norm(*(pos+countout),*(pos+countout+m),*(pos+countout+2*m));

        chptdotsink =    ( (*(chpts+countin) * *(neg+countout))+(*(chpts+countin+l*l) * *(neg+countout+m))+(*(chpts+countin+2*l*l) * *(neg+countout+2*m)) );
        chptdotsource =  ( (*(chpts+countin) * *(pos+countout))+(*(chpts+countin+l*l) * *(pos+countout+m))+(*(chpts+countin+2*l*l) * *(pos+countout+2*m)) );
        normchpt = norm(*(chpts+countin),*(chpts+countin+l*l),*(chpts+countin+2*l*l));

        cosbeta  = chptdotsink/(normsink*normchpt);
        costheta = chptdotsource/(normsource*normchpt);

        *(pot+countin) += cur/(4*PI*1/222)*(2/rb-2/ra+1/R*log((ra+R-a*cosbeta)/(rb+R-b*costheta)));

            countin++;
    }
    countout++;
    countin = 0;
  }


}

/* the gateway function */
void mexFunction( int nlhs, mxArray *plhs[],
                  int nrhs, const mxArray *prhs[])
{
  double *neg,*pos,*chpts,*pot,cur,R;
  int     status,mrows,ncols,nchs;

  /*  check for proper number of arguments */
  /* NOTE: You do not need an else statement when using mexErrMsgTxt
     within an if statement, because it will never get to the else
     statement if mexErrMsgTxt is executed. (mexErrMsgTxt breaks you out of
     the MEX-file) */
  if(nrhs!=5)
    mexErrMsgTxt("Five inputs required.");
  if(nlhs!=1)
    mexErrMsgTxt("One output required.");

  /* check to make sure the first input argument is a scalar */
  if( !mxIsDouble(prhs[0]) || mxIsComplex(prhs[0]) ||
      mxGetN(prhs[0])*mxGetM(prhs[0])!=1 ) {
    mexErrMsgTxt("Input x must be a scalar.");
  }

  /*  get the scalar input cur */
  cur = mxGetScalar(prhs[0]);

  /*  create a pointer to the input matrix neg */
  neg = mxGetPr(prhs[1]);
  /*  create a pointer to the input matrix pos */
  pos = mxGetPr(prhs[2]);

    /*  create a pointer to the input matrix chpts */
  chpts = mxGetPr(prhs[3]);

  /*  get the # of channels */
  nchs = mxGetM(prhs[3]);

    /*  get the scalar input R */
  R = mxGetScalar(prhs[4]);

  /*  get the dimensions of the matrix input neg or pos (they have the same dimensions) */
  mrows = mxGetM(prhs[1]);
  ncols = mxGetN(prhs[1]);

  /*  set the output pointer to the output matrix */
  plhs[0] = mxCreateDoubleMatrix(nchs,nchs, mxREAL);

  /*  create a C pointer to a copy of the output matrix */
  pot = mxGetPr(plhs[0]);

  /*  call the C subroutine */
  frankpotential(cur,neg,pos,chpts,pot,nchs,mrows,R);

}
```

```c
/* potential.c
   Dr. Mingui Sun */
/* NOTE: THE AVERAGING ACROSS CHANNELS HAS BEEN COMMENED OUT */
#include "mex.h"
#include <math.h>
#define max(A, B)       ((A) > (B) ? (A) : (B))
#define min(A, B)       ((A) < (B) ? (A) : (B))
#define PI 4.*atan(1.0)


void potential(int nchs, double *d,double *m,double *e,double *v)
{
  int i,n;
  double t,s,q, sq, et, dt, ave[4],**u,**dmatrix();
  void free_dmatrix();
  u=dmatrix(0,nchs-1,0,3);
  /* printf("nchs=%d, d=(%f %f %f), m=(%f %f %f),  e=(%f %f %f)\n",
nchs, d[0],d[1],d[2],m[0],m[1],m[2],e[0],e[1],e[2]); */

  for(n=0; n<3; n++)ave[n] = 0.;
  for(n=0; n<3*nchs; n+=3){
    for(q=0,s=0,i=0; i<3; i++){
      et=e[n+i];
      dt=d[i];
      t = et-dt;
      q += t*t;
      s += et*dt;
    }
    sq = sqrt(q);
    for (i=0; i<3; i++){
      et=e[n+i];
      dt=d[i];
      t= (2.*(et-dt)/q + et + (et*s-dt)/(sq+1.-s))/sq;
      ave[i] += t;
      u[n/3][i] =t;
     }
 }
  for(n=0; n<3;n++)ave[n] /= (double) nchs;
  for(i=0; i<nchs; i++){
   v[i]=0;
   for(n=0; n<3; n++){

 /*     u[i][n] -= ave[n];
     v[i]+=m[n]*u[i][n];
         */

     v[i]+=(1./(4.*PI*(1./222.)))*m[n]*u[i][n];
   }
 }
/* for(n=0; n<nchs; n++)printf("v(%d)=%f\n",n,v[n]); */
free_dmatrix(u,0,nchs-1,0,3);

}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
        double  *d,*mo,*e,*v, tn;
        int nchs;
        unsigned int    m,n;

        /* Check for proper number of arguments */

        if (nrhs != 4) {
                mexErrMsgTxt("POTENTIAL requires four input arguments.");
        } else if (nlhs > 1) {
                mexErrMsgTxt("POTENTIAL requires one output argument.");
        }

    tn =  mxGetScalar(prhs[0]);
    nchs = (int) tn;


        /* Check the dimensions of d.  d must be 3 x 1 or 1 x 3. */


        m = mxGetM(prhs[1]);
        n = mxGetN(prhs[1]);
        if (!mxIsNumeric(prhs[1]) || mxIsComplex(prhs[1]) ||
                mxIsSparse(prhs[1])  || !mxIsDouble(prhs[1]) ||
                !((m == 3 && n == 1) || (m == 1 && n == 3 ))) {
                mexErrMsgTxt("d must be a vector of length 3.");
        }

        /* Check the dimensions of mo.  mo must be 3 x 1 or 1 x 3. */


        m = mxGetM(prhs[1]);
        n = mxGetN(prhs[1]);
        if (!mxIsNumeric(prhs[2]) || mxIsComplex(prhs[2]) ||
                mxIsSparse(prhs[2])  || !mxIsDouble(prhs[2]) ||
                !((m == 3 && n == 1) || (m == 1 && n == 3 ))) {
                mexErrMsgTxt("mo must be a vector of length 3.");
        }

        /* Check the dimensions of e.  e must be 3*nchs x 1 or 1 x 3*nchs */


        m = mxGetM(prhs[3]);
        n = mxGetN(prhs[3]);
        if (!mxIsNumeric(prhs[3]) || mxIsComplex(prhs[3]) ||
                mxIsSparse(prhs[3])  || !mxIsDouble(prhs[3]) ||
                !((m == 3*nchs && n == 1) || (m == 1 && n == 3*nchs ))) {
                printf("nchs = %d\n",nchs);
                mexErrMsgTxt("e must be a vector of length 3*nchs.");
        }

        /* Create a matrix for the return argument */
   plhs[0]=mxCreateDoubleMatrix(nchs,1,0);
        d = mxGetPr(prhs[1]);
        mo = mxGetPr(prhs[2]);
        e = mxGetPr(prhs[3]);
    v= mxGetPr(plhs[0]);
     potential(nchs,d,mo,e,v);
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) printf("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) printf("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}
void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;
        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}
```

## .2 SHAPE OPTIMIZATION

Contains the following files:

- shapesfundersq.m (refer to .1)
- shapesoptimize_sym.m
- shapesptsopt_v2.m
- shapestotarclength.m
- frankpotential.c (refer to .1)

```
% shapesoptimize_sym.m
% Brian Wessel
% Simulates the potential difference on a spherical surface
% began 7/19/02
% The antenna elements are symmetric

% ymax and xmax define the size (cm) of the box in which the shape is confined
xmax = .15;
ymax = .1;
zmax = 0.45;

% distance between the antenna elements
d = 0.95;

%location of the antenna
center = [0 0 6.5];

sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];
results = [];
totcurves = [];

% increment value for the rotation angle
psi_r_inc = 10;

% has to be an even number (the number of monopoles placed on each antenna plate x-section is then numbsections+1)
numbsections = 12;

% Determines how many curves will be used to increment through the curvature variable
desiredcurves = 10;


for ndp = 5:7  % alters conic sections

    ndp

    % outputs control points on a box and the total number of curves that will be used
    % (the total number of curves to be used is only different for the circle shape)
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    for k= 1:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k

        % Determines the shape parameters given the control points determined above
        a=1;
        b=1;
        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            a = d/2;
            b = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end


        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            xpoints = [];


            delxp = 1*10^(-5);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                linc/delxp;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;
                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end


            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end
        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end
        % creates negative values and adds a zero point for the xpoints
        xpoints = [fliplr(-xpoints) 0  xpoints];
        zpoints = -zmax:linc:zmax;
        ypoints = [];
        % calculates ypoints according to which shape the user specifies.
        if ndp == 3
            ypoints = zeros(length(xpoints),1);
        elseif ndp == 4     %circle points
            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);
        elseif ndp == 5 %parabola pts
            ypoints = a*(xpoints.*xpoints);
        elseif ndp == 6  %ellipse points
            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);
        elseif ndp == 7   %hyperbola points
            ypoints = (a^2+a^2/b^2*(xpoints.*xpoints)).^(0.5)-a;
        end

        shapepts = [];
        negshapepts = [];
        % creates the matrices containing the positive and negative antenna points
        for j = 1:length(zpoints)
            for w = 1:length(xpoints)
                shapepts = [shapepts; [xpoints(w) ypoints(w) zpoints(j)]];
                negshapepts = [negshapepts; [xpoints(w) -ypoints(w) zpoints(j)]];
            end
        end
        % bothshapepts = [shapepts;negshapepts];
        % figure(1)
        % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3));
        % xlabel('x');
        % ylabel('y');
        % zlabel('z');
        % axis equal

        % rotates psi_r for a fixed shape and curvature
        for psi_r = 0:psi_r_inc:90

            psi_r
            rshapepts = shapepts;
            rnegshapepts = negshapepts;
            %rotate the antenna
            % positive side
            temppts = [rshapepts(:,1) rshapepts(:,2)]*[cos(psi_r*pi/180) sin(psi_r*pi/180);-sin(psi_r*pi/180) cos(psi_r*pi/180)];
            rshapepts(:,1) = temppts(:,1);
            rshapepts(:,2) = temppts(:,2);
            % negative side
            temppts = [rnegshapepts(:,1) rnegshapepts(:,2)]*[cos(psi_r*pi/180) -sin(psi_r*pi/180);sin(psi_r*pi/180) cos(psi_r*pi/180)];
            rnegshapepts(:,1) = temppts(:,1);
            rnegshapepts(:,2) = temppts(:,2);
            phi_r = pi/2;  % rotates to y-axis
            % pos side
            temppts = [rshapepts(:,3) rshapepts(:,1)]*[cos(phi_r) sin(phi_r);-sin(phi_r) cos(phi_r)];
            rshapepts(:,3) = temppts(:,1);
            rshapepts(:,1) = temppts(:,2);
            % neg side
            temppts = [rnegshapepts(:,3) rnegshapepts(:,1)]*[cos(phi_r) sin(phi_r);-sin(phi_r) cos(phi_r)];
            rnegshapepts(:,3) = temppts(:,1);
            rnegshapepts(:,1) = temppts(:,2);

            bothshapepts = [rshapepts(:,1) rshapepts(:,2)+d/2 rshapepts(:,3)+center(1,3);rnegshapepts(:,1) rnegshapepts(:,2)-d/2 rnegshapepts(:,3)+center(1,3)];
            % figure(3)
            % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3));
            % xlabel('x');
            % ylabel('y');
            % zlabel('z');
            % axis([-zmax zmax -d/2-2*ymax d/2+2*ymax center(1,3)-2*xmax center(1,3)+2*xmax]);
            % view(-90,0);
            % axis equal


            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %%%% begin movie code
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            % lbpts = length(bothshapepts);
            %
            %
            % xlin = linspace(min(bothshapepts(1:lbpts/2,1)),max(bothshapepts(1:lbpts/2,1)),12);
            % if psi_r > 45
            %     ylin = linspace(min(bothshapepts(1:lbpts/2,2)),max(bothshapepts(1:lbpts/2,2)),6);
            % else
            %     zlin = linspace(min(bothshapepts(1:lbpts/2,3)),max(bothshapepts(1:lbpts/2,3)),6);
            % end
            %
            % figure(2)
            % %subplot(2,1,1)
            %
            % if psi_r > 45
            %     [X,Y] = meshgrid(xlin,ylin);
            %     mZ = griddata(bothshapepts(1:lbpts/2,1),bothshapepts(1:lbpts/2,2),bothshapepts(1:lbpts/2,3),X,Y,'linear');
            %     surf(X,Y,mZ);
            % else
            %     [Z,X] = meshgrid(zlin,xlin);
            %     mY = griddata(bothshapepts(1:lbpts/2,1),bothshapepts(1:lbpts/2,3),bothshapepts(1:lbpts/2,2),X,Z,'linear');
            %     surf(X,mY,Z);
            % end
            % axis([-1 1 -1 1 5.5 6.5]);
            % %axis on;
            % xlabel('x');
            % ylabel('y');
            % zlabel('z');
            % hold on
            %
            % nxlin = linspace(min(bothshapepts(lbpts/2+1:lbpts,1)),max(bothshapepts(lbpts/2+1:lbpts,1)),12);
            % if psi_r > 45
            %     nylin = linspace(min(bothshapepts(lbpts/2+1:lbpts,2)),max(bothshapepts(lbpts/2+1:lbpts,2)),6);
            % else
            %     nzlin = linspace(min(bothshapepts(lbpts/2+1:lbpts,3)),max(bothshapepts(lbpts/2+1:lbpts,3)),6);
            % end
            %
            % if psi_r > 45
            %   [nX,nY] = meshgrid(nxlin,nylin);
            %   mnZ = griddata(bothshapepts(lbpts/2+1:lbpts,1),bothshapepts(lbpts/2+1:lbpts,2),bothshapepts(lbpts/2+1:lbpts,3),nX,nY,'linear');
            %   surf(nX,nY,mnZ);
            % else
            %     [nZ,nX] = meshgrid(nzlin,nxlin);
            %   mnY = griddata(bothshapepts(lbpts/2+1:lbpts,1),bothshapepts(lbpts/2+1:lbpts,3),bothshapepts(lbpts/2+1:lbpts,2),nX,nZ,'linear');
            %   surf(nX,mnY,nZ);
            % end
            % axis([-1 1 -1 1 5.5 6.5]);
            % % axis on;
            % xlabel('x');
            % ylabel('y');
            % zlabel('z');
            % %axis equal
            % % axis([-4*xmax 4*xmax -2*max(datapts(:,2)) 2*max(datapts(:,2)) -2*zmax 2*zmax])
            % colormap('gray')
            % view(-72,0)
            % grid off

            % figure(2)
            % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3))
            % title('circular','FontSize',40);
            % axis equal
            % axis([-0.8 0.8 -1.2 1.2 5.5 6.5]);
            % view(-55,10)
            % grid off
            % xlabel('x (cm)');
            % ylabel('y (cm)');
            % zlabel('z (cm)');
            % set(gca,'Color',[.8,0.8,.8])
            % set(gcf,'Color',[.8,0.8,.8])

            % % the following 20 lines of code is actually used in making the movie
            % figure(1)
            % subplot(2,1,1)
            % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3))
            % if ndp==4
            %     shpe = 'circular';
            % elseif ndp==5
            %     shpe = 'parabolic';
            % elseif ndp==6
            %     shpe = 'elliptical';
            % elseif ndp==7
            %     shpe = 'hyperbolic';
            % end
            % title(shpe,'FontSize',40);
            % axis equal
            % axis([-0.8 0.8 -1.2 1.2 5.5 6.5]);
            % view(-55,10)
            % grid off
            % xlabel('x (cm)');
            % ylabel('y (cm)');
            % zlabel('z (cm)');
            % set(gca,'Color',[.8,0.8,.8])
            % set(gcf,'Color',[.8,0.8,.8])

            %hold off
            %%%%  end movie code
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            %This code calls frankpotential.meglx to increase the speed of processing

            clear Vr ra rb theta phi Px Py Pz

            % radius of the sphere
            R = 7.5;

            % (n+1)^2 will be the number of points on the sphere
            n = 50;

            % channel density (not used as an input but is interesting to know
            channeldensity = (n+1)^2/(4*pi*R^2)

            theta = pi*(-n:2:n)/n;
            phi = (pi/2)*(-n:2:n)'/n;
            Px = R*cos(phi)*cos(theta);
            Py = R*cos(phi)*sin(theta);
            Pz = R*sin(phi)*ones(1,length(sin(phi)));
            P = [Px,Py,Pz];

            % figure(1)
            % plot3(Px(:),Py(:),Pz(:))
            % title('points to find the potenial at')

            % centershapepts = [0.9 0.0 0.0; 0.9 0.9 0];
            % negcentershapepts = [0.0 0.9 0.0; -0.9 -0.9 0];

            % current (in mA) I divide by the number of simulated non-ideal dipoles
            % so that when I them all up, the resulting voltage is from
            % that total amount of current...so in this case, the total current is 1 mA
            % I need to divide by the length of bothshapepts/2 and not just shapepts because I actually take points off of bothshapepts which are
            % not reflected in shapepts
            I = 1/(length(bothshapepts)/2);

            % % realistic case
            % neg_i = [0.997 0 5.79]/7.5;
            % pos_i = [0   0 5.875]/7.5;

            % power = 1;
            % test case
            % neg_i = [1*10^(-power) 0 1*10^(-power-2)];
            % pos_i = [1*10^(-power)   0 1*10^(-(power+1))];
            % neg_i = [-0.05 -0.05 0];
            % pos_i = [0 0 0.05];
            % a = norm(neg_i);
            % b = norm(pos_i);

            g = length(bothshapepts);

            Vr = frankpotential(I,bothshapepts(g/2+1:g,:),bothshapepts(1:g/2,:),P,R);

            %
            % figure(3)
            % %axes(a13)
            % %subplot(2,2,2)
            % surf(Px,Py,Pz,Vr);
            % %title('Franks equation - c code')
            % colorbar
            % xlabel('x (cm)');
            % ylabel('y (cm)');
            % zlabel('z (cm)');
            % axis equal
            % shading interp
```

```
            % figure(3)
            % %axes(a13)
            % subplot(2,2,4)
            % imagesc(theta,flipud(phi),Vr);
            % xlabel('theta (radians)');
            % ylabel('phi (radians)');
            % title('Franks equation')

            results = [results ; max(max(Vr)) min(min(Vr)) ndp psi_r k a b MaxR xmax ymax zmax d numbsections desiredcurves center(1,3)];

        end

    end
end

% used in movie code
%shapesmovieplayer(PotentialFrames)

save resultsX_parab_ell_hyp.dat results -ascii
% load results.dat

for j = 1:length(results)
    if results(j,3)==3
        sheetresults = [sheetresults;results(j,:)];
    elseif results(j,3)==4
        circleresults = [circleresults;results(j,:)];
    elseif results(j,3)==5
        parabolaresults = [parabolaresults;results(j,:)];
    elseif results(j,3)==6
        ellipseresults = [ellipseresults;results(j,:)];
    elseif results(j,3)==7
        hyperbolaresults = [hyperbolaresults;results(j,:)];
    end
end

for j = 4:7
    if j==3
        shaperesults = sheetresults;
    elseif j==4
        shaperesults = circleresults;
    elseif j==5
        shaperesults = parabolaresults;
    elseif j==6
        shaperesults = ellipseresults;
    elseif j==7
        shaperesults = hyperbolaresults;
    end

    figure(1)
    subplot(2,2,j-3)
    if j~=3
        xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
        ylin = linspace(min(shaperesults(:,4)),max(shaperesults(:,4)),90/psi_r_inc+1);

        [Y,X] = meshgrid(ylin,xlin);

        Z = griddata(shaperesults(:,5),shaperesults(:,4),shaperesults(:,1)-shaperesults(:,2),X,Y);
        surf(X,Y,Z);
        if j==3
            title('sheet');
        elseif j==4
            title('circular');
        elseif j==5
            title('parabolic');
        elseif j==6
            title('elliptical');
        elseif j==7
            title('hyperbolic');
        end
        hold on;
        xlabel('curvature (index)');
        ylabel('psi (angle)');
        zlabel('potential difference (mV)');
        %colorbar;
        colormap gray;
        rotate3d
    end
    scatter3(shaperesults(:,5),shaperesults(:,4),shaperesults(:,1)-shaperesults(:,2))
    hold off;

end
```

```
% shapesptsopt_v2.m
% Brian Wessel
% Determines control points for specified boudning box, shape, and number
% of curves desired

function [totcurves,xpt,ypt] = shapesptsopt(xmax,ymax,ndp,desiredcurves);

if ndp==3
    totcurves = 1;
    xpt = xmax;
    ypt = 0;
else
    totcurves = desiredcurves;
    % adding an extra because you don't want the last point to be at x=0.
    numcurves=desiredcurves+1;
end


if ndp~=3

    if ndp==4

        % must refigure the total number of curves so the circle does
        % not bend in on itself if xpt becomes lower than ypt
        % Look at 1/9/04 notes to see what it doesn't matter if ymax>xmax
        % or ymax<xmax

        totcurves = floor(numcurves*xmax/(xmax+ymax));

    end


    for k = 1:totcurves
        if ymax>(k*(xmax+ymax)/numcurves) | ymax==(k*(xmax+ymax)/numcurves)
            xpt(k) = xmax;
            ypt(k) = k*(xmax+ymax)/numcurves;
        else
            ypt(k) = ymax;
            xpt(k) = (numcurves-k)*(xmax+ymax)/numcurves;
        end
    end

end
```

```
% shapestotarclength.m
% Brian Wessel
% Calculates arc length for certain parabolas, ellipses, and hyperbolas

function arclength = shapestotarclength(ndp,A,B,a,b);


if ndp ==5

    arclength = 1/4*(2*B*(1+4*a*B^2)^(1/2)*a^(1/2)+log(2*a^(1/2)*B+(1+4*a*B^2)^(1/2))-2*A*(1+4*a*A^2)^(1/2)*a^(1/2)-log(2*a^(1/2)*A+(1+4*a*A^2)^(1/2)))/a^(1/2);

elseif ndp==6

    if a<b
        B = b^.5;
        c = a;
        a = b;
        b = c;
    end

    argument1 =  real(mfun('EllipticE',B*(1/a)^(1/2),(1/a*(a-b))^(1/2)));
    argument2 =  real(-mfun('EllipticE',0,(1/a*(a-b))^(1/2)));
    arclength =  a^(1/2)*(argument1+argument2);

elseif ndp==7

    argument1 = -mfun('EllipticE',B*(-1/a)^(1/2),((a+b)/a)^(1/2));
    argument2 =  mfun('EllipticE',A*(-1/a)^(1/2),((a+b)/a)^(1/2));
    arclength =  real(i*a^(1/2)*(argument1+argument2));

end
```

## .3 FE SHAPES CREATION

Contains the following files:

- create_antennas_ptcontrol_v5.m
- normalpointsv2.m
- results_shapes.m
- shapesfundersq.m (refer to .1)
- shapesptsopt_v2.m (refer to .2)
- shapestotarclength.m (refer to .2)
- view_solution_Max.m

```
% create_antennas_ptcontrol_v5.m
% Brian Wessel
% creates antennas for input to FE program

% creates points using arc length hence point control

% This seems like overkill but FE software is very sensitive
% to points which are too close to eachother.  Thus one must exactly
% place the points along the shape of the curve at regular spacing so that none
% end up too near eachother as would happen for curves with large curvature where
% points are placed at regular spacing along the axis.

% ymax and xmax define the size of the box in which the shape is confined
% so small because FEMLab requires SI units, thus in meters

clear all
close all

% xmax = 0.2;
% ymax = 0.1;
% zmax = 0.9;
% d = 0.9;
% ant_thick = 0.03;

%xmax = 2e-3;
ymax = 1e-3;
zmax = 4.5e-3;
d = 9e-3;
ant_thick = 7.5e-4;
numangles = 10;

epoxyheight=4e-3; % which is two times the old xmax ...reference 1/8/04 why I made this change
xmax = epoxyheight/2-ant_thick;


if xmax > d/2
    error('xmax is greater than d/2 so the antenna elements will criss-cross at large angles!')
end

if ymax > 2*xmax
    error('ymax is greater than 2*xmax, so the epoxy will not cover the side of the antenna when the antenna is turned at a sharp angle (psi)!')
end

centerchoice = menu('Centering on or off?','on','off');
shapechoice = menu('Shape?','circle','parabola','ellipse','hyperbola');
shapechoice = shapechoice+3;
kchoice = menu('Curvature (k) ?','1','2','3','4','5','6','7','8','9','10');
psichoice = menu('Angle ?','0','10','20','30','40','50','60','70','80','90');
psichoice = (psichoice-1)*10;

% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 6; % MUST BE EVEN b/c linc = 2*totsum/numbsections.  It seems weird but having an even
% number of sections, will put an odd number of points because there is no middle section
% NOTE THAT THE NUBMER OF POINTS WILL BE NUMBSECTIONS+1
desiredcurves = 10; %if k only goes from 1:1 rather than 1:totcurves(ndp) than desiredcurves is overidden and
% therefore, only 1 curve will be created

if numangles==1
    psi_r_inc = 91; % just enough that only zero degrees will be calculated
else
    psi_r_inc = 90/(numangles-1); % increment value for the rotation angle in degrees
end

index = 0; % keeps track of the saving of epoxy and ae's for later use

for ndp = shapechoice:7  % alters shapes
    ndp

    % outputs points on box and the total number of curves that will be used
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    totcurves(ndp)
    %     display('program paused')
    %     pause

    for k= kchoice:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k

        a=1;
        b=1;

        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            b = d/2;
            a = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            clear ypoints
            xpoints = [];


            delxp = 1*10^(-6);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;

                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end


            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k) & i < (numbsections-3)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end
        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end

        % creates negative values and adds a zero point for the xpoints
        %nxpoints = -xpoints;
        xpoints = [fliplr(-xpoints) 0  xpoints];


        % end calculation of xpoints using arc length
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



        % calculates ypoints according to which shape the user specifies.
        % all shapes intersect through the origin
        if ndp == 3

            ypoints = zeros(size(xpoints));

        elseif ndp == 4     %circle points

            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 5 %parabola pts

            ypoints = a*(xpoints.*xpoints);

        elseif ndp == 6  %ellipse points

            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 7   %hyperbola points

            ypoints = (b^2+b^2/a^2*(xpoints.*xpoints)).^(0.5)-b;

        end

        %                 subplot(2,2,ndp-3)
        %                 scatter(xpoints,ypoints,'.')
        %                 axis([(-xmax-xmax/5) (xmax+xmax/5) -ymax/10 (ymax+ymax/5)]);
        %                 axis equal
        %                 if ndp==4
        %                     title('semicircular')
        %                 elseif ndp==5
        %                         title('parabolic')
        %                     elseif ndp==6
        %                             title('elliptical')
        %                         else
        %                             title('hyperbolic')
        %                         end
        %                     end
        %                 end
        %
        %                 pause

        %title(['shape',num2str(ndp),'   ','curve number',num2str(k)])

        [nxpoints,nypoints] = normalpointsv2(ndp,a,b,ant_thick,xpoints,ypoints);

        close(gcf)
        scatter(xpoints,ypoints,'r.');
        axis equal
        hold
        scatter(nxpoints,nypoints,'bx');
        axis equal

        title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
        pause


        %%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%% MAKING ANTENNA ELEMENT USING FEMLab FUNCTIONS
        % Note that the I am keeping the enclosing box x and y axes, so it is switched from normal


        for psi_r = psichoice:psi_r_inc:90   % rotates psi_r for a fixed shape and curvature

            index = index+1;

            psi_r



            %             aex=[fliplr(nxpoints),xpoints];
            %             aey=[fliplr(nypoints),ypoints];

            aex=[xpoints,fliplr(nxpoints)];

            aey=[ypoints,fliplr(nypoints)];

            % rotating neg. ae
            phi = psi_r*pi/180;

            rot_mat = [cos(phi) -sin(phi); sin(phi) cos(phi)];
            ae = [aex;aey];
            rae = rot_mat*ae;
            raex = rae(1,:);
            raey = rae(2,:)+d/2;


            %            %%%%%%%%%%%% Plot which checks for correct rotation
            %             plot(aex,aey,'b',raex,raey,'r')
```

```
            %             axis equal
            %             title('check for correct rotation');
            %             pause
            %%%%%%%%%%%%%%
            % step 1 -- create antenna elements
            aer=line2(raex,raey);
            ael=line2(raex,-raey);
            % %             %%%%%%%%%% Plotting
            % %                         close(gcf)
            % %                         scatter(raex,raey,'r.');
            % %                         axis equal
            % %                         hold
            % %                         scatter(raex,-raey,'b.');
            % %                         axis equal
            % %                         %pause
            % %                         geomplot(aer)
            % %                         axis equal
            % %                         hold
            % %                         geomplot(ael)
            % %                         axis equal
            % %                        % pause
            % %             %%%%%%%%%%%%%%%%%%%%
            %step2 -- create epoxy
            %close(gcf)


            %%%%%%%%%%%%%%%%%%%%%%%% code for creating outside antenna first...uses output of normalptsv2.m %%%%%%%%%%%%%%%%%%%%%%%%%%
            halfx = raex(numbsections+2:length(raex));
            halfy = raey(numbsections+2:length(raey));
            if centerchoice==1 % This choice will center the antenna element within the epoxy
                % This regime is fundamentally different from noncentering because it should expose
                % the difference between having a shield of epoxy vs free space and will give insight
                % into how much a shield is effective at pushing the current to the far-field
                %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
                % will be too close to connect with a line...however, I should add about half the antenna thickness
                % to each side so that it doesn't make a sharp point near the tip of the antenna as in the case of the
                % elliptically shaped antenna elements
                if psi_r == 0
                    xtra = ant_thick/2;
%                     xtra = ant_thick*4/5;
                    totx = [halfx (halfx(length(halfx))-xtra) (halfx(length(halfx))-xtra) fliplr(halfx) (halfx(1)+xtra) (halfx(1)+xtra) ];
                    toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
                    epoxy = line2(totx,toty);
                else
                    % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
                    % therefore cause a problem.
                    % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
                    % old way #2 -- xtra = max(raex)+1/5*xmax;
                    %xtra = min(raex)+2*xmax+ant_thick;
                    halfx = fliplr(halfx);
                    halfy = fliplr(halfy);
                    widthant = max(halfx)-halfx(1);
%                     xout = ant_thick*cos(phi);
                    xout = ant_thick*cos(phi)*3/5;
%                     xtra = (epoxyheight-widthant)/2;
                    xtra = (epoxyheight-widthant)*4/5;
%                     yout = ant_thick*sin(phi);
                    yout = ant_thick*sin(phi)*3/5;
% %                      totx = [halfx max(halfx) (max(halfx)+xtra) (max(halfx)+xtra) max(halfx) fliplr(halfx) halfx(1) (halfx(1)-xtra) (halfx(1)-xtra) halfx(1)];
% %
% %                     toty = [halfy (halfy(length(halfy))+ant_thick/2) (halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -fliplr(halfy) -(halfy(1
% %
                   %%%Sophisticated Modeling which is memory intensive
                   totx = halfx;
                   toty = halfy;
%                     totx = [totx (max(halfx)+xout)];
                   totx = [totx (halfx(length(halfx))+xout)];
                   toty = [toty (halfy(length(halfy))+yout)];
                   totx = [totx (max(halfx)+xtra)];
                   toty = [toty (halfy(length(halfy))+yout)];
                   totx = [totx (max(halfx)+xtra)];
                   toty = [toty -(halfy(length(halfy))+yout)];
%                     totx = [totx (max(halfx)+xout)];
                    totx = [totx (halfx(length(halfx))+xout)];
                   toty = [toty -(halfy(length(halfy))+yout)];
                   totx = [totx fliplr(halfx)];
                   toty = [toty -fliplr(halfy)];
                   totx = [totx (halfx(1)-xout)];
                   toty = [toty -(halfy(1)-yout)];
                   totx = [totx (halfx(1)-xtra)];
                   toty = [toty -(halfy(1)-yout)];
                   totx = [totx (halfx(1)-xtra)];
                   toty = [toty (halfy(1)-yout)];
                   totx = [totx (halfx(1)-xout)];
                   toty = [toty (halfy(1)-yout)];

% % %                  Primitive Modeling
% %
% %                      totx = [(max(halfx)+xtra)];
% %                      toty = [min(halfy)-ant_thick/2];
% %
% %                      totx = [totx (max(halfx)+xtra)];
% %                      toty = [toty max(halfy)];
% %
% %                      totx = [totx (max(halfx)+xtra+ant_thick)];
% %                      toty = [toty max(halfy)];
% %
% %                      totx = [totx (max(halfx)+xtra+ant_thick)];
% %                      toty = [toty -max(halfy)];
% %
% %                      totx = [totx (max(halfx)+xtra)];
% %                      toty = [toty -max(halfy)];
% %
% %                      totx = [totx (max(halfx)+xtra)];
% %                      toty = [toty -(min(halfy)-ant_thick/2)];
% %
% %
% %                      totx = [totx (halfx(1)-xtra)];
% %                      toty = [toty -(min(halfy)-ant_thick/2)];
% %
% %
% %                      totx = [totx (halfx(1)-xtra)];
% %                      toty = [toty (min(halfy)-ant_thick/2)];
```

```
       epoxy = line2(totx,toty);
    end


else
    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line
    if psi_r == 0
        totx = [fliplr(halfx) halfx];
        toty = [-fliplr(halfy) halfy];
        epoxy = line2(totx,toty);
    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        xtra = min(raex)+2*xmax;
        totx = [fliplr(halfx) halfx  xtra xtra];
        toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
        epoxy = line2(totx,toty);
    end

end
%%%%%%%%%%%%%%%%%%%%%%%% end create outside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

% % %%%%%%%%%%%%%%%%%%%%%%%% code for creating inside antenna first...uses output of normalptsv4.m %%%%%%%%%%%%%%%%%%%%%%%%%
% %
% %             halfx = raex(1:numbsections+1);
% %             halfy = raey(1:numbsections+1);
% %             if centerchoice==1 % This choice will center the antenna element within the epoxy
% %                 % This regime is fundamentally different from noncentering because it should expose
% %                 % the difference between having a shield of epoxy vs free space and will give insight
% %                 % into how much a shield is effective at pushing the current to the far-field
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     widthant = max(halfx)-halfx(1);
% %                     xtra = (2*xmax-widthant)/2;
% %                     totx = [halfx (max(halfx)+xtra) (max(halfx)+xtra) fliplr(halfx) (halfx(1)-xtra) (halfx(1)-xtra)];
% %                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %
% %
% %             else
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     xtra = min(raex)+2*xmax;
% %                     totx = [fliplr(halfx) halfx  xtra xtra];
% %                     toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %             end
% % %%%%%%%%%%%%%%%%%%%%%%%% end create inside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

%         %%%%%%%%%%%%%%% Plotting
%             geomplot(epoxy)
%             title('epoxy')
%             axis equal
%             pause
%         %%%%%%%%%%%%%%%%%%%%%%%%%

% %             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% %             % The following code should work but the epoxy cannot be extruded for some reason
% %             halfx = raex(1:numbsections+1);
% %             width = halfx(1,1)-halfx(1,numbsections+1);
% %             halfy = raey(1:numbsections+1);
% %             if width > (2*xmax-xmax/10)
% %                 totx = [halfx fliplr(halfx)]; % the epoxy should be directly behind the ae's
% %                 toty = [halfy -fliplr(halfy)];
% %             else
% %                 extend = xmax-width/2;
% %                 leftx = (-extend+halfx(numbsections+1));
% %                 rightx = (halfx(1)+extend);
% %                 totx = [halfx(1) halfx fliplr(halfx) halfx(1) rightx rightx];
% %                 toty = [(halfy(1)+ymax) halfy -fliplr(halfy) -(halfy(1)+ymax) -(halfy(1)+ymax) (halfy(1)+ymax)];
% %             end
% %
% %             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% creating extrusion plane
c_wrkp=geomgetwrkpln('quick',{'xy',-zmax});

% extruding epoxy and ant. elements
e_epoxy = extrude(epoxy,'wrkpln',c_wrkp,'distance',zmax*2);
e_aer = extrude(aer,'wrkpln',c_wrkp,'distance',zmax*2);
e_ael = extrude(ael,'wrkpln',c_wrkp,'distance',zmax*2);

% storing as positive and negative antenna elements and as insulation
pae{index} = e_aer;
nae{index} = e_ael;
ins{index} = e_epoxy;

% % %%%%%%%%%%%%%%%% Plotting
% %             close(gcf)
```

```
            % %             geomplot(e_aer);
            % %             axis equal
            % %             hold on
            % %             geomplot(e_ael);
            % %             axis equal
            % %             geomplot(e_epoxy);
            % %             axis equal
            % %             hold off
            % %             pause
            % % %%%%%%%%%%%%%%%%%

            clear fem
            v=1
            % Geometry
            fem.geom = pae{v};

            fem.mesh = meshinit(fem);

            % Integrate on subdomains
            ae_vol(v)=postint(fem,'1');

            % check insulation
            clear fem
            v=1
            % Geometry
            fem.geom = ins{v};

            fem.mesh = meshinit(fem,'hauto',3);
            display('insulation is compatible')

            pae{index} = rotate(pae{index},pi/2,[0 1 0 ],[0 0 0]);
            nae{index} = rotate(nae{index},pi/2,[0 1 0 ],[0 0 0]);
            ins{index} = rotate(ins{index},pi/2,[0 1 0 ],[0 0 0]);

            center = [0;0;6.5e-2];

            paei = move(pae{1},center)
            naei = move(nae{1},center)
            insi = move(ins{1},center)

            ae_vol

            geomplot(insi),hold,geomplot(paei),geomplot(naei),axis equal
            set(gcf,'Color',[1 1 1])
            xlabel('x (m)','FontSize',20),ylabel('y (m)','FontSize',20),zlabel('z (m)','FontSize',20)
            set(gca,'FontSize',15)

            display('program finished...ready to import into femlab')
            pause
        end
    end
end


% % %%%%%%%%%%%%%%%%%%%%%%%%%
% % display('broke the program at line 349');
% % break
% % %%%%%%%%%%%%%%%%%%%%%%%%%

% find volume of each antenna element
for v = 1:numangles:(totcurves(4)*10+3*totcurves(5)*10)
    v % display v to user

    clear fem

    % Geometry
    fem.geom = pae{v};

    fem.mesh = meshinit(fem);

    % Integrate on subdomains
    ae_vol(v)=postint(fem,'1');

    for i=1:(numangles-1) % because the others are just rotated so the volume shouldn't change
        ae_vol(v+i) = ae_vol(v);
    end

end

display('break at line 511')
break

% find volume of each antenna element
for v = 1:numangles:(4*numangles)
    v % display v to user

    clear fem

    % Geometry
    fem.geom = pae{v};

    fem.mesh = meshinit(fem);

    % Integrate on subdomains
    ae_vol(v)=postint(fem,'1');

    for i=1:(numangles-1) % because the others are just rotated so the volume shouldn't change
        ae_vol(v+i) = ae_vol(v);
    end

end


subindex = 1;

for index = 1:(4*numangles)

    % rotate for preparation into FEMLab (I'll have FEMLab move it to 6.5 cm in z direction

    % rotate around y-axis
    pae{index} = rotate(pae{index},pi/2,[0 1 0 ],[0 0 0]);
    nae{index} = rotate(nae{index},pi/2,[0 1 0 ],[0 0 0]);
    ins{index} = rotate(ins{index},pi/2,[0 1 0 ],[0 0 0]);

    %     % rotate around z-axis
    %     pae{index} = rotate(pae{index},pi/2,[0 0 1 ],[0 0 0]);
    %     nae{index} = rotate(nae{index},pi/2,[0 0 1 ],[0 0 0]);
    %     ins{index} = rotate(ins{index},pi/2,[0 0 1 ],[0 0 0]);

    %%%%%%%%%%%%%%%%%%%%
    if index==1
        figure(4)
    elseif index==7
        figure(5)
        subindex = subindex-6;
    elseif index==13
        figure(6)
        subindex = subindex-6;
    elseif index==19
        figure(7)
        subindex = subindex-6;
    end

    subplot(3,2,subindex)
    geomplot(ins{index})
    hold

    geomplot(pae{index});
    axis equal
    geomplot(nae{index});
    xlabel('x (m)')
    ylabel('y (m)')
    zlabel('z (m)')
    axis equal
    hold
    %%%%%%%%%%%%%%%%%%%%%%

    subindex = subindex+1;
end

%for i=1:(numangles*totcurves(ndp)),geomplot(pae{i}),axis equal,pause,end
```

```
% normalpointsv2.m
% Brian Wessel
% program finds normal points to a shape using the
% gradient of the function

function [nxpoints,nypoints] = normalpointsv2(ndp,a,b,t,xpoints,ypoints)

% The reason why I have to subtract b from ypoints for the circle and
% ellipse is because the points must lie on the original shape for the
% normal vector to be calculated correctly

% This program uses a normal vector such that the resulting curve is larger
% which makes it hard to control xmax and ymax so I made version 3

if ndp==4

    ypoints = ypoints-b;
    nypoints = t*ypoints/b+(ypoints+b);
    nxpoints = t*xpoints/b+xpoints;

    ypoints = ypoints+b;

elseif ndp==5

    c = -1./(4*a*ypoints+1).^(1/2);

    nypoints = c*t + ypoints;
    nxpoints = -2*a*c.*xpoints*t+xpoints;


elseif ndp==6

    ypoints = ypoints-b;

    for i=1:length(xpoints)

        c(i) = (  (a^4*ypoints(i)^2)/(b^4*xpoints(i)^2+a^4*ypoints(i)^2)  )^(1/2);

        if ypoints(i)==0
            nypoints(i) = ypoints(i);
            if xpoints(i)<0
                nxpoints(i) = xpoints(i)-t;
            else
                nxpoints(i) = xpoints(i)+t;
            end
        else
            nypoints(i) = -c(i)*t+ypoints(i);
            nxpoints(i) = -c(i)*t*(b^2*xpoints(i))/(a^2*ypoints(i))+xpoints(i);
        end
    end

    nypoints = nypoints+b;

elseif ndp==7

    ypoints = ypoints+b;

    c = 1./((xpoints.^2*b^4+a^4*ypoints.^2)./(a^4*ypoints.^2)).^(1/2);
    nypoints = -c*t+ypoints;
    nxpoints = t*c.*xpoints*b./(a^2*(a^2+xpoints.^2)).^(1/2)+xpoints;

    nypoints = nypoints-b;

end

\end{vebatim}
\newpage
\begin{verbatim}
% results_shapes.m
% Brian Wessel
% plots the data given resultsX.dat

load resultsX.dat

%circle points

curve = [1 3 5 6 1 3 5 6 1 3 5 6];


angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==4 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [15.9951 15.8749 16.6184 16.4645 15.291 16.2845 17.7529 18.041 17.2755 17.5412 17.9634 18.0391];


c_per = (maxv-ana_maxv)./ana_maxv*100;


xlin = [1 3 5 6];
ylin = [0 40 90 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,1)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('potential difference (mV)');
title('CIRCULAR','FontSize',15)
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;

pause
%parabola points


curve = [1 4 7 10 1 4 7 10 1 4 7 10];


angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==5 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [14.8218 16.1019 16.3218 16.059 16.0707 17.5165 18.5119 19.5355 16.8201 17.5545 17.9899 18.6729];

p_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 4 7 10];
ylin = [0 40 90];
[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,2)
surf(X,Y,Z);
```

```
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('potential difference (mV)');
title('PARABOLIC','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,2*maxv)
hold off;
pause
%ellipse points
curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==6 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [15.867 16.5356 16.2539 15.67 16.3824 17.2376 18.3474 19.3977 16.4966 17.5417 18.0732 18.6771];
e_per = (maxv-ana_maxv)./ana_maxv*100;
xlin = [1 4 7 10];
ylin = [0 40 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,3)
surf(X,Y,Z);
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('potential difference (mV)');
title('ELLIPTICAL','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,2*maxv)
hold off;
pause
%hyperbola points
curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==7 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end
maxv = [14.9403 16.2585 16.2866 15.7563 16.0993 17.4512 18.5534 19.387 16.9944 17.6567 18.3735 18.0686];
h_per = (maxv-ana_maxv)./ana_maxv*100;
xlin = [1 4 7 10];
ylin = [0 40 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,4)
surf(X,Y,Z);
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('potential difference (mV)');
title('HYPERBOLIC','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,2*maxv)
hold off;
pause
%%%%%%%%%%%%%%%%%%%%%% PERCENT INCREASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
figure(2)
curve = [1 3 5 6 1 3 5 6 1 3 5 6];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
xlin = [1 3 5 6];
ylin = [0 40 90 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,c_per,X,Y);
subplot(2,2,1)
surf(X,Y,Z);
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('Percent Increase');
title('CIRCULAR','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,c_per)
hold off;
pause
curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
xlin = [1 4 7 10];
ylin = [0 40 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,p_per,X,Y);
subplot(2,2,2)
surf(X,Y,Z);
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('Percent Increase');
title('PARABOLIC','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,p_per)
hold off;
pause
Z = griddata(curve,angle,e_per,X,Y);
```

```
subplot(2,2,3)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('Percent Increase');
title('ELLIPTICAL','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,e_per)
hold off;

pause

subplot(2,2,4)
Z = griddata(curve,angle,h_per,X,Y);
surf(X,Y,Z);
hold on;
xlabel('curvature (index)');
ylabel('psi (angle)');
zlabel('Percent Increase');
title('HYPERBOLIC','FontSize',15)
%colorbar;
colormap gray;
rotate3d
scatter3(curve,angle,h_per)
hold off;
```

```
% view_solution_Max.m
% Brian Wessel
% January 11, 2004

% This program extracts the surface solution as outputted by FEMLab using
% ctrl-F while in FEMLab

R = 7.5e-2;
n = 200;
theta = pi*(-n:2:n)/n;
phi = (pi/2)*(-n:2:n)'/n;
Px = R*cos(phi)*cos(theta);
Py = R*cos(phi)*sin(theta);
Pz = R*sin(phi)*ones(1,length(sin(phi)));
P = [Px(:),Py(:),Pz(:)]';

result=postinterp(fem,'V',P,'ext',1);

% % % choice = menu('Choose Method','whole sphere data','half sphere data');
% % %
% % % if choice==1
% % %     C = reshape(result,n+1,n+1);
% % %     figure(2)
% % %     surf(Px,Py,Pz,C);
% % %     title('FEMLab surface solution (Volts)')
% % %     xlabel('x (m)');
% % %     ylabel('y (m)');
% % %     zlabel('z (m)');
% % %     shading interp
% % %     colorbar
% % %     axis equal
% % %     save C.mat C
% % % else
% % %     C = reshape(result,n+1,n+1);
% % %     C(1:(n+1),(n/2+2):(n+1))  =fliplr(-C(1:(n+1),1:(n/2)));
% % %     for i = 1:(n+1)
% % %         for j = 1:(n+1)
% % %             if isnan(C(i,j))
% % %                 C(i,j) = 0;
% % %             end
% % %         end
% % %     end


    figure(2)
    C = reshape(result,n+1,n+1);
    surf(Px,Py,Pz,C);
    title('FEMLab surface solution (Volts)')
    xlabel('x (m)');
    ylabel('y (m)');
    zlabel('z (m)');
    shading interp
    colorbar
    axis equal
    display(['max(max(C)) = ',num2str(max(max(C))),'    min(min(C)) = ',num2str(min(min(C)))]);

    display(['The average absolute maxes = ',num2str( (max(max(C))+abs(min(min(C))))/2)]);

    %save('G:\FEMLab\Sphere_Models\1p7\sheetsurf.mat','C')
    %clear fem ans P* R n phi result theta
    %save C

%%%end
```

## .4 FOUR SHELL FILES

Contains the following files:

- test_potential4shell_VII.m
- potential4shell.c

```
% test_potential4shell_vII.m
% Brian Wessel
% CODE FOR THE 4 SHELL MODEL

clear all

R = 1;

nchs = 1;

d = [0 0 0];
m = [0 1 0];
elec = [0 1 0];

poten = potential4shell(nchs,d,m,elec,R);
poten

break


figure(1)
subplot(2,2,1)
surf(Px,Py,Pz,poten);
view(-216,30)
%title('Frank''s equation  (colorbar in mV) and (1mA b/w plates)','FontSize',25)
title('Frank''s equation  (colorbar in mV) and (1mA b/w plates)')
set(gca,'FontSize',15)
hc = colorbar;
set(hc,'FontSize',15)
xlabel('x (cm)','FontSize',18);
ylabel('y (cm)','FontSize',18);
zlabel('z (cm)','FontSize',18);
axis equal
shading interp

figure(1)
subplot(2,2,3)
imagesc(theta,phi,poten);
set(gca,'FontSize',15)
xlabel('theta (radians)','FontSize',25);
ylabel('phi (radians)','FontSize',25);
% title('Frank''s equation','Fontname','Courier','FontSize',25)
title('Frank''s equation')

pause
gtext(['non-dipole around center at pos_i = ',num2str(pos_i),' and neg_i = ',num2str(neg_i)])


%%%%%%%%%%%%%%%% END CODE FOR THE 4 SHELL MODEL
```

```c
/* potential4shell.c
   created by Dr. Mingui Sun
   modified by Wessel
*/
/* NOTE: THE AVERAGING ACROSS CHANNELS HAS BEEN COMMENED OUT */
#include <math.h>
#include "mex.h"
#define dot(a,b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
#define alv (double *)malloc((unsigned) c*sizeof(double))


void potential(int nchs, double *d,double *m,double *ev,double *R,double *v)
{

int i,j,c = nchs;
double cn1[3],*Q,*Q2,*Q3,*Q5,*x,*x2,*x3,
       t,t2,*tx,S,*vt,*vr,p,*q,**T,*TT,*RR, **e, **dmatrix(),r = *R;
static double K=24.11438531695384,
cn[3]={0.0, 2.396851543074,  2.903046901856},
a[3]={3.473289839721, -0.193784805097, 0.005423738295};

T = dmatrix(0,nchs-1,0,3);
e = dmatrix(0,nchs-1,0,3);

for(i=0;i<nchs;i++)
{
   e[i][0]=ev[i*3];
   e[i][1]=ev[i*3+1];
   e[i][2]=ev[i*3+2];
}


tx=alv;
q=alv;
vt=alv; vr=alv; RR=alv; x=alv;
Q=alv; Q2=alv; Q3=alv;Q5=alv; x2=alv; x3=alv; TT=alv;

//for(i=0;i<3;i++)T[i]=alv;

p=dot(d,d);
t=sqrt(p);

if(t == 0.)
{
        for(i=0;i<c;i++)
                v[i]=57.799*dot(m,e[i])/(r*r*r);
                return;
}
for(i=0;i<c;i++) q[i]=dot(d,e[i]);

for(i=0;i<c;i++)for(j=0;j<3;j++)T[i][j]=e[i][j]*p-d[j]*q[i];

for(i=0;i<c;i++){
 TT[i]=dot(m,T[i])/sqrt(dot(T[i],T[i]));
 RR[i]=dot(m,d)/t;x[i]=q[i]/(t*r);
}

t/=r;
for(i=0; i<3;i++) cn1[i]=cn[i]-(a[0]+i*(a[1]+i*a[2]));

for(i=0;i<c;i++){

        Q2[i]=1./(1.-2.*x[i]*t+t*t); Q[i]=sqrt(Q2[i]); Q3[i]=Q[i]*Q2[i]; Q5[i]=Q3[i]*Q2[i];
        t2=t*t;
        x2[i]=x[i]*x[i];
        tx[i]=t*x[i];
        S=sqrt(1-x2[i]);
        x3[i]=x2[i]*x[i];
        vr[i] = cn1[1]*x[i] +cn1[2]*t*(1.5*x2[i]-0.5) + a[0]*(Q[i]-1.)/t
                + a[1]*(x[i]-t)*Q3[i] + a[2]*(t*(x2[i]+t2-2.)+x[i]*(1.-t2))*Q5[i];

        if(fabs(x[i])==1) vt[i] =0.; else {
vt[i] = S*(cn1[1] +3.*tx[i]*cn1[2]/2.+a[0]*Q[i]*(1.+Q[i])/(Q[i]-tx[i]*Q[i]+1.)
   + a[1]*Q3[i]  + a[2]*Q5[i]*(1.- 2.*t2 + tx[i])); }

v[i]= K*(TT[i]*vt[i]+RR[i]*vr[i])/(r*r);
}

  free_dmatrix(T,0,nchs-1,0,3);
  free_dmatrix(e,0,nchs-1,0,3);

}

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{
        double  *d,*dm,*ev,*v, tn,*R;
        int nchs;
        unsigned int    m,n;

        /* Check for proper number of arguments */

        if (nrhs != 5) {
                mexErrMsgTxt("POTENTIAL requires five input arguments.");
        } else if (nlhs > 1) {
                mexErrMsgTxt("POTENTIAL requires one output argument.");
        }

    tn =  mxGetScalar(prhs[0]);
    nchs = (int) tn;


        /* Check the dimensions of d.  d must be 3 x 1 or 1 x 3. */


        m = mxGetM(prhs[1]);
        n = mxGetN(prhs[1]);
        if (!mxIsNumeric(prhs[1]) || mxIsComplex(prhs[1]) ||
                mxIsSparse(prhs[1])  || !mxIsDouble(prhs[1]) ||
                !((m == 3 && n == 1) || (m == 1 && n == 3 ))) {
                mexErrMsgTxt("d must be a vector of length 3.");
        }

        /* Check the dimensions of mo.  mo must be 3 x 1 or 1 x 3. */


        m = mxGetM(prhs[1]);
        n = mxGetN(prhs[1]);
        if (!mxIsNumeric(prhs[2]) || mxIsComplex(prhs[2]) ||
                mxIsSparse(prhs[2])  || !mxIsDouble(prhs[2]) ||
                !((m == 3 && n == 1) || (m == 1 && n == 3 ))) {
                mexErrMsgTxt("mo must be a vector of length 3.");
        }

        /* Check the dimensions of e.  e must be 3*nchs x 1 or 1 x 3*nchs */

        m = mxGetM(prhs[3]);
        n = mxGetN(prhs[3]);
        if (!mxIsNumeric(prhs[3]) || mxIsComplex(prhs[3]) ||
                mxIsSparse(prhs[3])  || !mxIsDouble(prhs[3]) ||
                !((m == 3*nchs && n == 1) || (m == 1 && n == 3*nchs ))) {
                printf("nchs = %d\n",nchs);
                mexErrMsgTxt("e must be a vector of length 3*nchs.");
        }

        /* Create a matrix for the return argument */
   plhs[0]=mxCreateDoubleMatrix(nchs,1,0);
        d = mxGetPr(prhs[1]);
        dm = mxGetPr(prhs[2]);
        ev = mxGetPr(prhs[3]);
        R  = mxGetPr(prhs[4]);
```

```
         v = mxGetPr(plhs[0]);
     potential(nchs,d,dm,ev,R,v);
}

double **dmatrix(nrl,nrh,ncl,nch)
int nrl,nrh,ncl,nch;
{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) printf("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) printf("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}
void free_dmatrix(m,nrl,nrh,ncl,nch)
double **m;
int nrl,nrh,ncl,nch;
{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}
```

## .5 FE VS FRANK EQUATION

Contains the following files:

- EF_FEM_Cond_DC_1shellmodelv3.m
- frankpotential.c (refer to .1)

```
% EF_FEM_Cond_DC_1shellmodelv3.m
% Brian Wessel
% This is from the file ErnestFrankTest.  I will be using this file
% to compare results with FEMLab results.

%testing Ernest Frank's arbitrary placed and spaced dipole

clear all

choice = menu('Choose the type of source','nonideal dipole around center','nonideal near surface','parallel sheets around center','parallel sheet near surface')

nsp = 25;
if choice == 1
    pos_i = [0  0.5 0];
    neg_i = [0 -0.5 0];
    % current (in mA) I divide by the number of simulated non-ideal dipoles so that when I them all up, the resulting voltage is from
% that total amount of current...so in this case, the total current is 1 mA
%I = 1/(length(centershapepts))^2;
I = 1; % 1 mA in THIS case puts the solution in terms of mV
elseif choice == 2
    pos_i = [0  0.5 6.5];
    neg_i = [0 -0.5 6.5];
        % current (in mA) I divide by the number of simulated non-ideal dipoles so that when I them all up, the resulting voltage is from
% that total amount of current...so in this case, the total current is 1 mA
%I = 1/(length(centershapepts))^2;
    I = 1; % 1 mA in THIS case
elseif choice == 3
    xlim = 0.45;
    x = linspace(-xlim,xlim,nsp);
    k = 1;
    for i = 1:nsp
        for j = 1:nsp

            % this filters out the values that are not between -0.2 and 0.2 in the z-direction
            if abs(x(j)) < 0.2
                neg_i(k,:) = [x(i) -0.5 x(j)];
                k = k+1;
            end

        end
    end
    pos_i = [neg_i(:,1) neg_i(:,2)*-1 neg_i(:,3)];
    I = 1/length(pos_i)
elseif choice == 4
    xlim = 0.45;
    x = linspace(-xlim,xlim,nsp);
    k = 1;
    for i = 1:nsp
        for j = 1:nsp
            % this filters out the values that are not between -0.2 and 0.2 in the z-direction
            if abs(x(j)) < 0.2
                neg_i(k,:) = [x(i) -0.5 (x(j)+6.5)];
                k = k+1;
            end
        end
    end
    pos_i = [neg_i(:,1) neg_i(:,2)*-1 neg_i(:,3)];
    I = 1/length(pos_i)
end

%     % DEBUGGING CODE -- Shows the current sources I created above
%     scatter3(neg_i(:,1),neg_i(:,2),neg_i(:,3))
%     hold
%     scatter3(pos_i(:,1),pos_i(:,2),pos_i(:,3))
%     hold
%     axis equal
%     break

% radius of the sphere
R = 7.5;

% (n+1)^2 will be the number of points on the sphere
n = 30;

% channel density (not used as an input but is interesting to know)
channeldensity = (n+1)^2/(4*pi*R^2)

theta = pi*(-n:2:n)/n;
phi = (pi/2)*(-n:2:n)'/n;
Px = R*cos(phi)*cos(theta);
Py = R*cos(phi)*sin(theta);
Pz = R*sin(phi)*ones(1,length(sin(phi)));

P = [Px,Py,Pz];


Vr = frankpotential(I,neg_i,pos_i,P,R);


figure(1)
subplot(2,2,1)
surf(Px,Py,Pz,Vr);
view(-216,30)
title('Frank''s equation ')
hc = colorbar;
set(hc,'FontSize',15)
xlabel('x (cm)');
ylabel('y (cm)');
zlabel('z (cm)');
axis equal
shading interp

figure(1)
subplot(2,2,3)
imagesc(theta,phi,Vr);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('Frank''s equation')

pause

if choice==1
    gtext(['non-ideal dipole around center at pos_i = ',num2str(pos_i),'   and neg_i = ',num2str(neg_i),])
elseif choice==2
    gtext(['non-ideal dipole at near surface where pos_i = ',num2str(pos_i),' and neg_i = ',num2str(neg_i)])
elseif choice==3
    gtext(['sheet of charge near center / center located at pos_i = ',num2str([0 0.5 6.5]),' and neg_i = ',num2str([0 -0.5 6.5])])
elseif choice==4
    gtext(['sheet of charge near surface / center located at pos_i = ',num2str([0 0.5 6.5]),' and neg_i = ',num2str([0 -0.5 6.5])])
end

    gtext(['(colorbar in mV) and (1mA b/w sources)'])
    %changes the background to white
    set(gcf,'Color',[1,1,1])

% if choice==1
%     load('F:\FEMLabFiles\2p7p1\C.mat')
% elseif choice==2
%     load('F:\FEMLabFiles\2p9\C.mat')
% elseif choice==3
%     %load('F:\FEMLabFiles\1p5\c.mat')
%     C = ones(n+1,n+1);
%     warning('no data available, surface data shown is junk')
% elseif choice==4
%     load('F:\FEMLabFiles\2p4p1\C.mat')
% end


if choice==1
    load 2p7p1C.mat
elseif choice==2
    load 2p9C.mat
elseif choice==3
    %load('F:\FEMLabFiles\1p5\c.mat')
    C = ones(n+1,n+1);
    warning('no data available, surface data shown is junk')
elseif choice==4
    load 2p4p1C.mat
end



figure(1)
subplot(2,2,2)
surf(Px,Py,Pz,C);
view(-216,30)
title('DC Conductive Media')
hc = colorbar;
set(hc,'FontSize',15)
xlabel('x (cm)');
ylabel('y (cm)');
zlabel('z (cm)');
axis equal
shading interp

figure(1)
subplot(2,2,4)
imagesc(theta,phi,C);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('DC Conductive Media')


% sinusoid error

pause

sub = C-Vr;
div = Vr./C;

md = mean(div);
mean(md);


figure(2)
subplot(2,1,1)
surf(Px,Py,Pz,sub);
view(-216,30)
title('Frank Solution subtracted from FEMLab solution (ERROR) (numbers should all be zero ideally)  (colorbar in mV) and (1mA b/w plates)')
hc = colorbar;
set(hc,'FontSize',15)
xlabel('x (cm)','FontSize',18);
ylabel('y (cm)','FontSize',18);
zlabel('z (cm)','FontSize',18);
axis equal
shading interp

msub = mean(sub);
subplot(2,1,2)
plot(0:n,max(max(sub))*sin((0:1:n)*2*pi/n),0:n,max(max(sub)),'rx')
title('ERROR vs Index   [FEM-Frank]')
sincurve = [num2str(max(max(sub))),'*sin(2*pi/',num2str(n),'*n)'];
legend(sincurve,'error')
xlabel('index')
ylabel('Error')
grid

%next 7 lines are just for making a plot for a presentation
figure(4)
subplot(2,1,1)
plot(0:n,msub,'rx')
title('ERROR vs Index   [FEM-Frank]')
xlabel('Index')
ylabel('Error')
grid
subplot(2,1,2)
plot(0:n,max(max(msub))*sin((0:1:n)*2*pi/n),0:n,msub,'rx')
title('ERROR and Sine Curve vs Index   [FEM-Frank]')
sincurve = [num2str(max(max(msub))),'*sin(2*pi/',num2str(n),'*n)'];
legend(sincurve,'error')
xlabel('Index')
ylabel('Error')
grid
    %changes the background to white
    set(gcf,'Color',[1,1,1])



%%%%%%%%%% NORMALIZING THE IMAGESC DATA %%%%%%%%%%%%%%%%%

mVr = min(min(Vr));
nVr = Vr/mVr;

mC = max(max(C));
nC = C/mC;

figure(3)
subplot(2,1,1)
surf(Px,Py,Pz,nVr-nC);
view(-216,30)
title('Frank Solution subtracted from FEMLab solution and Normalized (numbers should all be zero ideally)  (colorbar in mV) and (1mA b/w plates)')
set(gca,'FontSize',15)
hc = colorbar;
set(hc,'FontSize',15)
xlabel('x (cm)','FontSize',18);
ylabel('y (cm)','FontSize',18);
zlabel('z (cm)','FontSize',18);
axis equal
shading interp



figure(5)
plot(0:n,max(Vr'-C),'o',0:n,-max(Vr-C'),'*',0:n,max(Vr'-C)-max(Vr-C'),'x',0:n,max(max(Vr'-C)-max(Vr-C'))*sin((0:1:n)*2*pi/n))
legend('max(Vr''-C)','-max(Vr-C'')','max(Vr''-C)-max(Vr-C'')',[num2str(max(max(Vr'-C)-max(Vr-C'))),'*sin(2*pi/',num2str(n),'*n)'])
```

## .6 EXTRACTING POINTS FEMLAB OUTPUT (CTRL-F)

Contains the following file:

- exps.m

```
% exps.m
% Brian Wessel
% June 16, 2003

% This program extracts the surface solution as outputed by FEMLab using
% ctrl-F while in FEMLab


R = 7.5e-2;
n = 30;
theta = pi*(-n:2:n)/n;
phi = (pi/2)*(-n:2:n)'/n;
Px = R*cos(phi)*cos(theta);
Py = R*cos(phi)*sin(theta);
Pz = R*sin(phi)*ones(1,length(sin(phi)));
P = [Px(:),Py(:),Pz(:)]';


result=postinterp(fem,'V',P,'ext',1);

choice = menu('Choose Method','whole sphere data','half sphere data');

if choice==1
    C = reshape(result,n+1,n+1);
    figure(2)
    surf(Px,Py,Pz,C);
    title('FEMLab surface solution (Volts)')
    xlabel('x (m)');
    ylabel('y (m)');
    zlabel('z (m)');
    shading interp
    colorbar
    axis equal
    save C.mat C
else
    C = reshape(result,n+1,n+1);
    C(1:(n+1),(n/2+2):(n+1))  =fliplr(-C(1:(n+1),1:(n/2)));
    for i = 1:(n+1)
        for j = 1:(n+1)
            if isnan(C(i,j))
                C(i,j) = 0;
            end
        end
    end

    figure(2)
    surf(Px,Py,Pz,C);
    title('FEMLab surface solution (Volts)')
    xlabel('x (m)');
    ylabel('y (m)');
    zlabel('z (m)');
    shading interp
    colorbar
    axis equal

    %save('G:\FEMLab\Sphere_Models\1p7\sheetsurf.mat','C')
    %clear fem ans P* R n phi result theta
    %save C
end
```

## .7 IDEAL VS NON-IDEAL DIPOLES (2D $\infty$ MEDIUM)

Contains the following file:

- ContourPlotsV2.m

```
% ContourPlotsV2.m
% Brian Wessel
% Ideal vs NonIdeal Dipoles i.e., for an infinite medium
% in 2D

clear all
close all

% [X,Y] = meshgrid(-2:.2:2,-2:.2:3);
% Z = X.*exp(-X.^2-Y.^2);
% [C,h] = contour(X,Y,Z,linspace(min(min(Z)),max(max(Z)),50));
% clabel(C,h)
% colormap cool

% Note, the equation for a dipole oriented along
% the z-axis and placed at the origin is: Phi = p*cos(theta)/(4*pi*sigma*r^2)
% Therefore, if we let p=4*pi*sigma, the equation becomes:
% cos(theta)/r^2

% 1 dipole
figure(1)
subplot(2,2,1)
[X,Y] = meshgrid(-1.0001:.002:1,-1.0001:.002:1);
rc = length(X);

for i = 1:rc
    for j = 1:rc
        if (X(i,j)<0 & Y(i,j)<0) | (X(i,j)>0 & Y(i,j)<0)
            Z(i,j) = cos(atan(X(i,j)/Y(i,j))+pi)/(X(i,j)^2+Y(i,j)^2);
        else
            Z(i,j) = cos(atan(X(i,j)/Y(i,j)))/(X(i,j)^2+Y(i,j)^2);
        end
    end
end

[C,h] = contourf(X,Y,Z,linspace(-100,100,10));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{one dipole}')
% clabel(C,h)

figure(2)
[C,h] = contourf(X,Y,Z,linspace(-100,100,10));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{one dipole}')

%%%%%%%%%%%%%%

% 2 dipoles
figure(1)
subplot(2,2,2)

d=0.075;

Yp = Y+d;
Ym = Y-d;

for i = 1:rc
    for j = 1:rc

        if (X(i,j)<0 & Ym(i,j)<0) | (X(i,j)>0 & Ym(i,j)<0)
            Zm(i,j) = cos(atan(X(i,j)/Ym(i,j))+pi)/(X(i,j)^2+Ym(i,j)^2);
        else
            Zm(i,j) = cos(atan(X(i,j)/Ym(i,j)))/(X(i,j)^2+Ym(i,j)^2);
        end

        if (X(i,j)<0 & Yp(i,j)<0) | (X(i,j)>0 & Yp(i,j)<0)
            Zp(i,j) = cos(atan(X(i,j)/Yp(i,j))+pi)/(X(i,j)^2+Yp(i,j)^2);
        else
            Zp(i,j) = cos(atan(X(i,j)/Yp(i,j)))/(X(i,j)^2+Yp(i,j)^2);
        end

    end
end


Z = Zm+Zp;

[C,h] = contourf(X,Y,Z,linspace(-100,100,20));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{two dipoles}')
% clabel(C,h)

figure(3)
[C,h] = contourf(X,Y,Z,linspace(-100,100,20));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{two dipoles}')


%%%%%%%%%%%%%%%%%

% 2 monopoles
% The dipole moments must equal with the equation being
% Phi = I/(4*pi*sigma*r), where in the other equation p = I*d = 4*pi*sigma,
% then I must equal 4*pi*sigma/d, therefore, we have the equation being
% Phi = 1/(d*r)


Zdip = Z;

clear Z Zm Zp Ym Yp

figure(1)
subplot(2,2,3)
Z = d^(-1)./(X.^2+(Y-d).^2)-d^(-1)./(X.^2+(Y+d).^2);
[C,h] = contourf(X,Y,Z,linspace(-100,100,10));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{two monopoles}')
% clabel(C,h)


figure(4)
Z = d^(-1)./(X.^2+(Y-d).^2)-d^(-1)./(X.^2+(Y+d).^2);
[C,h] = contourf(X,Y,Z,linspace(-100,100,10));
cmp=colormap('gray');
```

```
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('\Phi_{two monopoles}')

% error between 2 dipoles and two monopoles

Zerr = Z-Zdip;

figure(1)
subplot(2,2,4)
[C,h] = contourf(X,Y,Zerr,linspace(-100,100,10));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar

% clabel(C,h)

axis equal
title('Error --> (\Phi_{monopoles}-\Phi_{dipoles})')


figure(5)
[C,h] = contourf(X,Y,Zerr,linspace(-100,100,10));
cmp=colormap('gray');
colormap(flipud(cmp))
set(gcf,'Color',[1 1 1])
colorbar
axis equal
title('Error --> (\Phi_{monopoles}-\Phi_{dipoles})')
```

## .8 IDEAL VS NON-IDEAL DIPOLES (3D SPHERICAL MEDIUM)

Contains the following files:

- ErnestFrankTestv2.m
- frankpotential.c (refer to .1)
- potential.c (refer to .1)

```
% ErnestFrankTestv2.m
% Brian Wessel
%testing Ernest Frank's arbitrary placed and spaced dipole

% version 2 changes phi and theta limits to plot the solution more intuitively
% on the imagesc plane

clear Vr ra rb theta phi

% radius of the sphere
R = 1;

% (n+1)^2 will be the number of points on the sphere
n = 75;

phi = pi*(-n:2:n)'/n-3*pi/2;
theta = (pi/2)*(-n:2:n)/n+pi/2;
Px = R*cos(phi)*cos(theta);
Py = R*cos(phi)*sin(theta);
Pz = R*sin(phi)*ones(1,length(sin(phi)));


% figure(1)
% plot3(Px(:),Py(:),Pz(:))
% title('points to find the potenial at')

%current (in mA)
I = 1;

%dipole placement (must adhere to the constraints of Ernest Frank's assumptions) (They must be in the x-z plane)

% % realistic case
% neg_i = [0.997 0 5.79]/7.5;
% pos_i = [0   0 5.875]/7.5;

power = 1;
% test case
% neg_i = [1*10^(-power) 0 1*10^(-power-2)];
% pos_i = [1*10^(-power)   0 1*10^(-(power+1))];

% pos_i = [0  0.01 0];
% neg_i = [0 -0.01 0];

pos_i = [0 1/7.5 6.5/7.5];
neg_i = [0 -1/7.5 6.5/7.5];

a = norm(neg_i);
b = norm(pos_i);

% the distance between the source and sink - calculate this to ensure that both moments are the same
dab = (a^2+b^2-2*dot(neg_i,pos_i))^(0.5);

for i = 1:length(Px)
    for j = 1:length(Px)
        ra(i,j) = ( (Px(i,j)-neg_i(1,1))^2 + (Py(i,j)-neg_i(1,2))^2 +(Pz(i,j)-neg_i(1,3))^2)^0.5;
        rb(i,j) = ((Px(i,j)-pos_i(1,1))^2 +(Py(i,j)-pos_i(1,2))^2 +(Pz(i,j)-pos_i(1,3))^2)^0.5;
        p = [Px(i,j) Py(i,j) Pz(i,j)];
        sink = [neg_i(1,1) neg_i(1,2) neg_i(1,3)];
        source = [pos_i(1,1) pos_i(1,2) pos_i(1,3)];

        costheta = dot(p,source)/(norm(p)*norm(source));
        cosbeta  = dot(p,sink)/(norm(p)*norm(sink));

        Vr(i,j) = I/(4*pi*1/222)*[2/rb(i,j)-2/ra(i,j)+1/R*log((ra(i,j)+R-a*cosbeta)/(rb(i,j)+R-b*costheta))] ;
    end
end

figure(1)
subplot(2,3,1)
surf(Px,Py,Pz,Vr);
title('Franks equation')
colorbar
xlabel('x');
ylabel('y');
zlabel('z');
axis equal
shading interp

subplot(2,3,4)
imagesc(theta,phi,Vr);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('Franks equation')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%now testing potential.c
clear elec poten temppoten

% creates the current dipole moment vector
mvec = (pos_i-neg_i);
m = mvec/norm(mvec);
m = m*I*dab;

poten = zeros(n+1);

ndp = 1

if ndp == 2
% % for the case where two dipoles are placed at sink and source (like the antenna)
    d = [neg_i ; pos_i];
else
% % for the case where only one dipole is placed for two monopoles
    d = (neg_i+pos_i)/2;
end

poten = zeros(n+1);
for j = 1:ndp
    for i=1:n+1
        elec(1:3:(n+1)*3)=Px(i,:);
        elec(2:3:(n+1)*3)=Py(i,:);
        elec(3:3:(n+1)*3)=Pz(i,:);
        temppoten(:,i) = potential((n+1),d(j,:),m,elec);
    end
    poten = poten+temppoten/ndp;
end

figure(1)
subplot(2,3,2)
surf(Px,Py,Pz,poten');
colorbar
xlabel('x');
ylabel('y');
zlabel('z');
title('potential.c')
axis equal
shading interp

subplot(2,3,5)
imagesc(theta,phi,poten');
xlabel('theta (radians)');
ylabel('phi (radians)');
title('potential.c')

% This shows that, at present, a proportionality factor is needed.
sub = Vr-poten';
mean(sub);
```

```
max(sub);
% "format rat" ensures that if the numbers in sub are all approximately the same thing, it can display it as such and not just the differences
% % in the 10th decimal place
% format rat
subplot(2,3,3)
surf(Px,Py,Pz,sub);
shading interp
xlabel('x');
ylabel('y');
zlabel('z');
title('Error (Vfrank-Videal)')
axis equal
colorbar

subplot(2,3,6)
imagesc(theta,phi,sub);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('Error')

%%%%%%%%%%%%%%%%%%Adding the dB interpretation%%%%%%%%%%%%%%%%%%

%%%% This isn't working well, because I get log(zero) which is undefined
%%%% so it might be a better idea to just use percent difference

DB = 20*log10(abs(sub/max(max(Vr))));

figure(2)
subplot(2,1,1)
surf(Px,Py,Pz,DB);
shading interp
xlabel('x');
ylabel('y');
zlabel('z');
title('dB Error --> 20*log10(|Verror|/max(max(Vr)))')
axis equal
colorbar

subplot(2,1,2)
imagesc(theta,phi,DB);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('dB Error')

figure(3)

DB = 20*log10(abs(sub./Vr));

figure(2)
subplot(2,1,1)
surf(Px,Py,Pz,DB);
shading interp
xlabel('x');
ylabel('y');
zlabel('z');
title('dB Error --> 20*log10(|Verror|./Vr)')
axis equal
colorbar

subplot(2,1,2)
imagesc(theta,phi,DB);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('dB Error')


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%% Percent Difference (PD) Approach

PD = abs(sub./(Vr+1e-8))*100;

figure(4)
subplot(2,1,1)
surf(Px,Py,Pz,PD);
shading interp
xlabel('x');
ylabel('y');
zlabel('z');
title('Percent Difference --> |Verror|/max(max(Vr))*100')
axis equal
colorbar

subplot(2,1,2)
imagesc(theta,phi,PD);
xlabel('theta (radians)');
ylabel('phi (radians)');
title('Percent Difference')

%just a separate plot so I can put in Thesis
figure(5)
surf(Px,Py,Pz,PD);
shading interp
xlabel('x (cm)','FontSize',15);
ylabel('y (cm)','FontSize',15);
zlabel('z (cm)','FontSize',15);
%title('Percent Difference --> Error./FrankSolution*100','FontSize',15)
axis equal
colorbar
view(-55,78)
grid off
set(gcf,'Color',[1 1 1])
set(gca,'FontSize',15)
cmp=colormap('autumn')
colormap(flipud(cmp))
title('Percent Error')
```

## .9 HYPOTHETICAL ANGLE

Contains the following files:

- shapesoptimize_sym_TEST_PSI.m
- shapestotarclength.m (refer to .2)
- shapesfundersq.m (refer to .1)
- shapesptsopt_v2.m (refer to .2)
- frankpotential.c (refer to .1)

```
% shapesoptimize_sym_TEST_PSI.m
% Brian Wessel
% Opimizes the potential difference between the positve and negative potential
% began 7/19/02
% This code is for the symmetric case
% This file is special because it changes the center
% to test if the optimum psi is a function of the position
% which I am hypothesizing that it should be since if the antenna
% was placed at the center, the optimum angle should be 0 degrees

clear all
close all

% ymax and xmax define the size of the box in which the shape is confined
% xmax = .2;
% ymax = .2;
% zmax = 0.6;
% d = 1.4;
xmax = .15;
ymax = .1;
zmax = 0.45;
d = 0.95;
sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];
results = [];
totcurves = [];

% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 12; % has to be an even number (the number of monopoles placed on each antenna plate is then numbsections+1
desiredcurves = 10;
psi_r_inc = 10; % increment value for the rotation angle

zvals = [0 0.5 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5]
% zvals = [0 6.8]
% varying the center according to specified vector of z values

for t = 1:length(zvals)

    center = [0 0 zvals(t)];

    for ndp = 4:7  % alters shapes
        ndp

        % outputs points on box and the total number of curves that will be used
        [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

        for k= 1:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
            k

            a=1;
            b=1;
            if ndp==4
                b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
            elseif ndp==5
                a = ypt(k)/xpt(k)^2;
            elseif ndp==6
                a = xpt(k);
                b = ypt(k);
            elseif ndp==7
                a = d/2;
                b = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
            end


            % uses the arclength to calculate the placement of the number of dipoles
            % otherwise it uses a much simpler way for the sheet points
            % also an easy way to calculate the total arclength of a circle is used

            if ndp~=3

                % calculates the total arc length in a curve from 0 to xmax then
                % multiplies by 2 to get the total arclength from -xmax to xmax
                % due to symmetry of the curves

                clear xpoints
                xpoints = [];


                delxp = 1*10^(-5);

                if ndp==6
                    delxp = 1*10^(-6);
                end

                if ndp ~=4
                    %                 xp = 0;
                    %                 fpsq = 0;
                    %                 totsum = 0;
                    %                 while (xpt(k)-xp)>delxp %using this break out statement because if xp gets too close to xpt(k), fpsq gets huge
                    %                         fpsq = shapesfundersq(xp,ndp,a,b);
                    %                         totsum = totsum+delxp*(1+fpsq)^0.5;
                    %                         xp = xp + delxp;
                    %                 end
                    %                 linc = 2*totsum/numbsections;
                    clear xpoints
                    xpoints = [];

                    totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                    linc = 2*totsum/numbsections;

                    linc/delxp;

                    % calculates the circle arclength increment quickly
                elseif ndp==4

                    xy = [xpt(k) ypt(k)];
                    rad = [0 b];
                    origin = [0 0];
                    cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                    gamma = acos(cosgamma);
                    arclength = 2*b*gamma;
                    linc = arclength/numbsections;

                    for p = 1:numbsections/2
                        xpoints(p) = b*sin((linc*p)/b);
                    end
                end


                xp = 0;
                lsum = 0;

                i = 1;
                fpsq = [];

                %       end,end,end %take out after testing
                if ndp ~=4

                    % numerically determines the x value for a specified unit of arc length
                    while xp < xpt(k)
                        while lsum<linc
                            fpsq = shapesfundersq(xp,ndp,a,b);
                            lsum = lsum+delxp*(1+fpsq)^0.5;
                            xp = xp + delxp;
                            if xp > xpt(k)
                                break
                            end
                        end

                        lsum = 0;

                        if xp< xpt(k)
                            xpoints(i) = xp;
                        end

                        i = i+1;
                    end
                    xpoints = [xpoints, xpt(k)];
                end
            else ndp==3
                linc = xpt(k)*2/numbsections;
                xpoints = linc:linc:xpt(k);
            end

            % creates negative values and adds a zero point for the xpoints
            %nxpoints = -xpoints;
            xpoints = [fliplr(-xpoints) 0  xpoints];
            length(xpoints);


            zpoints = -zmax:linc:zmax;
            ypoints = [];
            % calculates ypoints according to which shape the user specifies.
            if ndp == 3

                ypoints = zeros(length(xpoints),1);

            elseif ndp == 4     %circle points

                ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

            elseif ndp == 5 %parabola pts

                ypoints = a*(xpoints.*xpoints);

            elseif ndp == 6  %ellipse points

                ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

            elseif ndp == 7   %hyperbola points

                ypoints = (a^2+a^2/b^2*(xpoints.*xpoints)).^(0.5)-a;

            end

            % hold on,scatter(xpoints,ypoints),
            % %for i = 1:length(xpoints),arcl(i) = shapestotarclength(ndp,0,xpoints(i),a^2,b^2);,if i>1,figure(10),plot(arcl(i)-arcl(i-1),i,'o'),hold on,end,end,
            % end,end    %delimit this code after testing
            % %hold on,plot(xpoints,ypoints),end,end    %delimit this code after testing

            shapepts = [];
            negshapepts = [];

            for j = 1:length(zpoints)
                for w = 1:length(xpoints)
                    shapepts = [shapepts; [xpoints(w) ypoints(w) zpoints(j)]];
                    negshapepts = [negshapepts; [xpoints(w) -ypoints(w) zpoints(j)]];
                end
            end

            % bothshapepts = [shapepts;negshapepts];
            % figure(1)
            % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3));
            % xlabel('x');
            % ylabel('y');
            % zlabel('z');
            % axis equal


            for psi_r = 0:psi_r_inc:90% rotates psi_r for a fixed shape and curvature

                % [points]=shapesintersect() should go here if I end up wanting to use the assymetric case

                psi_r
                rshapepts = shapepts;
                rnegshapepts = negshapepts;

                %rotate the antenna
                % positive side
                tempppts = [rshapepts(:,1) rshapepts(:,2)]*[cos(psi_r*pi/180) sin(psi_r*pi/180);-sin(psi_r*pi/180) cos(psi_r*pi/180)];
                rshapepts(:,1) = tempppts(:,1);
                rshapepts(:,2) = tempppts(:,2);
                % negative side
                tempppts = [rnegshapepts(:,1) rnegshapepts(:,2)]*[cos(psi_r*pi/180) -sin(psi_r*pi/180);sin(psi_r*pi/180) cos(psi_r*pi/180)];
                rnegshapepts(:,1) = tempppts(:,1);
                rnegshapepts(:,2) = tempppts(:,2);

                phi_r = pi/2;
                % pos side
                tempppts = [rshapepts(:,3) rshapepts(:,1)]*[cos(phi_r) sin(phi_r);-sin(phi_r) cos(phi_r)];
                rshapepts(:,3) = tempppts(:,1);
                rshapepts(:,1) = tempppts(:,2);
                % neg side
                tempppts = [rnegshapepts(:,3) rnegshapepts(:,1)]*[cos(phi_r) sin(phi_r);-sin(phi_r) cos(phi_r)];
                rnegshapepts(:,3) = tempppts(:,1);
                rnegshapepts(:,1) = tempppts(:,2);


                bothshapepts = [rshapepts(:,1) rshapepts(:,2)+d/2 rshapepts(:,3)+center(1,3);rnegshapepts(:,1) rnegshapepts(:,2)-d/2 rnegshapepts(:,3)+center(1,3)];
                %bothshapepts = [shapepts;negshapepts];


                % sets the data within bounds and checks that the farthest point is not touching or outside the sphere
                tempppts=[];
                for u = 1:length(bothshapepts)
                    MaxR(u) = norm(bothshapepts(u,:));
                    %if abs(bothshapepts(u,2))<(ymax+d/2) & abs(bothshapepts(u,3))>(center(1,3)-xmax) & abs(bothshapepts(u,3))<(center(1,3)+xmax)
                    %    tempppts = [tempppts; bothshapepts(u,:)];
                    %end
                end
                MaxR = max(MaxR);
                %bothshapepts = tempppts;


                % figure(3)
                % scatter3(bothshapepts(:,1),bothshapepts(:,2),bothshapepts(:,3));
                % xlabel('x');
                % ylabel('y');
                % zlabel('z');
                % axis([-zmax zmax -d/2-2*ymax d/2+2*ymax center(1,3)-2*xmax center(1,3)+2*xmax]);
                % view(-90,0);
                % axis equal

                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                %This code calls frankpotential.meglx to increase the speed of processing
```

```
                clear Vr ra rb theta phi Px Py Pz
                % radius of the sphere
                R = 7.5;
                % (n+1)^2 will be the number of points on the sphere
                n = 50;
                % channel density (not used as an input but is interesting to know
                channeldensity = (n+1)^2/(4*pi*R^2);
                theta = pi*(-n:2:n)/n;
                phi = (pi/2)*(-n:2:n)'/n;
                Px = R*cos(phi)*cos(theta);
                Py = R*cos(phi)*sin(theta);
                Pz = R*sin(phi)*ones(1,length(sin(phi)));
                P = [Px,Py,Pz];
                % figure(1)
                % plot3(Px(:),Py(:),Pz(:))
                % title('points to find the potenial at')
                % centershapepts = [0.9 0.0 0.0; 0.9 0.9 0];
                % negcentershapepts = [0.0 0.9 0.0; -0.9 -0.9 0];
                % current (in mA) I divide by the number of simulated non-ideal dipoles so that when I them all up, the resulting voltage is from
                % that total amount of current...so in this case, the total current is 1 mA
                % I need to divide by the length of bothshapepts/2 and not just shapepts because I actually take points off of bothshapepts which are
                % not reflected in shapepts
                I = 1/(length(bothshapepts)/2);
                % % realistic case
                % neg_i = [0.997 0 5.79]/7.5;
                % pos_i = [0   0 5.875]/7.5;
                % power = 1;
                % test case
                % neg_i = [1*10^(-power) 0 1*10^(-power-2)];
                % pos_i = [1*10^(-power)   0 1*10^(-(power+1))];
                % neg_i = [-0.05 -0.05 0];
                % pos_i = [0 0 0.05];
                % a = norm(neg_i);
                % b = norm(pos_i);
                g = length(bothshapepts);
                Vr = frankpotential(I,bothshapepts(g/2+1:g,:),bothshapepts(1:g/2,:),P,R);
                %
                % figure(3)
                % %axes(a13)
                % %subplot(2,2,2)
                % surf(Px,Py,Pz,Vr);
                % %title('Franks equation - c code')
                % colorbar
                % xlabel('x (cm)');
                % ylabel('y (cm)');
                % zlabel('z (cm)');
                % axis equal
                % shading interp
                % figure(3)
                % %axes(a13)
                % subplot(2,2,4)
                % imagesc(theta,flipud(phi),Vr);
                % xlabel('theta (radians)');
                % ylabel('phi (radians)');
                % title('Franks equation')
                % the while loop is for multi-resolution (not going to use at this point)
                %             while (ymax<xmax)% the current potential-previous > 1*10^(-3)
                %
                %                 % calculate potential
                %                 if currentpotential-previous>1*10^(-3)
                %                 % increase resolution
                %                 end
                %
                %             end

                results = [results ; max(max(Vr)) min(min(Vr)) ndp psi_r k a b MaxR xmax ymax zmax d numbsections desiredcurves center(1,3)];
            end
        end
        iter=1;
        for qw = 1:length(results(:,1))
            if results(qw,3)==ndp
                tmpresults(iter,:) = results(qw,:);
                iter = iter+1;
            end
        end
        [CC,II] = max(tmpresults(:,1));
        MaxAtPsi(ndp-3,t) = results(II,4)
        tmpresults = [];
    end

path1 = ['C:\Documents and Settings\bwessel\Desktop\PsiTest\Trial1\results',num2str(zvals(t)),'.dat']
save(['C:\Documents and Settings\bwessel\Desktop\PsiTest\Trial1\MaxAtPsi.dat'],'MaxAtPsi','-ASCII')
save(path1,'results','-ASCII')
%     save(['/net/bwessel-lh/usr3/users/bwessel/matlab/Results/PsiTest/results',num2str(zvals(t)),'.dat'],'results','-ASCII')
    % save resultsX_parab_ell_hyp.dat results -ascii
    % load results.dat
    results = [];
end % end loop to iterate center position

for t = 1:length(zvals)
    ['results',num2str(zvals(t)),'.dat']
    results = load(path1,'.dat'])

    for j = 1:length(results)
        if results(j,3)==3
            sheetresults = [sheetresults;results(j,:)];
        elseif results(j,3)==4
            circleresults = [circleresults;results(j,:)];
        elseif results(j,3)==5
            parabolaresults = [parabolaresults;results(j,:)];
        elseif results(j,3)==6
            ellipseresults = [ellipseresults;results(j,:)];
        elseif results(j,3)==7
            hyperbolaresults = [hyperbolaresults;results(j,:)];
        end
    end
    for j = 4:7
        if j==3
            shaperesults = sheetresults;
        elseif j==4
            shaperesults = circleresults;
        elseif j==5
            shaperesults = parabolaresults;
        elseif j==6
            shaperesults = ellipseresults;
        elseif j==7
            shaperesults = hyperbolaresults;
        end

        figure(1)
        subplot(2,2,j-3)
        if j~=3
            xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
            ylin = linspace(min(shaperesults(:,4)),max(shaperesults(:,4)),90/psi_r_inc+1);

            [Y,X] = meshgrid(ylin,xlin);

            Z = griddata(shaperesults(:,5),shaperesults(:,4),shaperesults(:,1)-shaperesults(:,2),X,Y);
            surf(X,Y,Z);
            if j==3
                title('sheet');
            elseif j==4
                title('circular');
            elseif j==5
                title('parabolic');
            elseif j==6
                title('elliptical');
            elseif j==7
                title('hyperbolic');
            end
            hold on;
            xlabel('curvature (index)');
            ylabel('psi (angle)');
            zlabel('potential difference (mV)');
            %colorbar;
            colormap gray;
            rotate3d
        end
        scatter3(shaperesults(:,5),shaperesults(:,4),shaperesults(:,1)-shaperesults(:,2))
        hold off;

    end

    pause

    shaperesults=[];
    circleresults = [];
    parabolaresults = [];
    ellipseresults = [];
    hyperbolaresults = [];

end


figure

normzvals = zvals/7.5*100;

subplot(2,2,1)
plot(normzvals,MaxAtPsi(1,:),'b*')
title('Circular','FontSize',15)
ylabel('Optimum Angle (degrees)','FontSize',14)
xlabel('Percent Radius (%)','FontSize',14)
set(gca,'FontSize',14)
axis([0 100 0 45])


subplot(2,2,2)
plot(normzvals,MaxAtPsi(2,:),'b*')
title('Parabolic','FontSize',15)
ylabel('Optimum Angle (degrees)','FontSize',14)
xlabel('Percent Radius (%)','FontSize',14)
set(gca,'FontSize',14)
axis([0 100 0 45])

subplot(2,2,3)
plot(normzvals,MaxAtPsi(3,:),'b*')
title('Elliptical','FontSize',15)
ylabel('Optimum Angle (degrees)','FontSize',14)
xlabel('Percent Radius (%)','FontSize',14)
set(gca,'FontSize',14)
axis([0 100 0 45])

subplot(2,2,4)
plot(normzvals,MaxAtPsi(4,:),'b*')
title('Hyperbolic','FontSize',15)
ylabel('Optimum Angle (degrees)','FontSize',14)
xlabel('Percent Radius (%)','FontSize',14)
set(gca,'FontSize',14)
axis([0 100 0 45])

set(gcf,'Color',[1 1 1])
```

## .10 EXTRACTING POINTS FROM SLICES IN 3D

Contains the following file:

- slicedata.m

```
% SliceData.m
% Brian Wessel
% March 20, 2004
% extracts points from slices that were obtained
% as output from FEMLab

clear all

ind = menu('Slice Plane?','X=0','Z=6.5e-2')

maxR = [7.5e-2 ((7.5e-2)^2-(6.5e-2)^2)^(0.5)]

set(gcf,'Color',[1 1 1]);
colormap cool
title('')
title('boundary voltage (mV)')
axis equal

xlabel('x (m)'),ylabel('y (m)'),zlabel('z (m)')


X = get(gco,'XData');
Y = get(gco,'YData');
Z = get(gco,'ZData');
C = get(gco,'CData');

p = [X(:)'; Y(:)'; Z(:)'];

if ind==2
    u = C(:)';
else
    u = C(:)'-mean(C(:));
end

% finding boundary points and values
m=1;
for j = 1:length(p)
    bpts = norm(p(:,j));

%     display('program paused')
%     pause
    if (maxR(ind)-bpts)<1e-3

%         (0.075-bpts)
%         display('true')
%         pause

        boundv(m) = u(j);
        boundp(1,m) = p(1,j);
        boundp(2,m) = p(2,j);

%             display('program paused - inside loop')
%             pause

        x=p(1,j);
        y=p(2,j);

        theta(m) = atan(y/x);

        % fixing atan function so it goes from 0 to 2*pi rather than -pi/2 to pi/2
        if x<0 & y>0
            theta(m) = theta(m)+pi;
        elseif x<0 & y<0
            theta(m) = theta(m)-pi;
        end

        m = m+1;
    end

end
%%%%%%%%%%%%%%%%%%%% GRAPHING
figure(2)
subplot(2,2,1)
plot(abs(boundv))
title('boundary voltage')

subplot(2,2,2)
% polar(theta,boundv+1.1*min(boundv),'r.')
polar(theta,abs(boundv),'r.')
title('boundary voltage')

subplot(2,2,3)
stem3(boundp(1,:),boundp(2,:),boundv,'ro')
title('boundary voltage')

figure(3)
polar(theta,abs(boundv),'r.')
title('boundary voltage (mV)')
set(gcf,'Color',[1 1 1]);

figure(4)
polar(theta,abs(boundv))
title('boundary voltage')
set(gcf,'Color',[1 1 1]);
%%%%%%%%%%%%%%%%%%%%
```

## .11 PDE TOOL CODE - EPOXY

Contains the following files:

- create_antennas_noReflector.m
- commandline_v9.m
- shapesfundersq.m (refer to .1)
- shapestotarclength.m (refer to .2)
- shapesptsopt_v2.m (refer to .2)
- normalpointsv2.m (refer to .3)

```
% create_antennas_noReflector.m
% Brian Wessel
% creates geometry for input to MatLab PDE solver
% where the antenna only has epoxy between the elements

clear all
close all

% xmax = 0.2;
% ymax = 0.1;
% zmax = 0.9;
% d = 0.9;
% ant_thick = 0.03;

results = [];

%xmax = 2e-3;
ymax = 1e-3;
zmax = 4.5e-3;
d = 9e-3;
ant_thick = 7.5e-4;
% ant_thick = 2.5e-4;
numangles = 10;

epoxyheight=4e-3; % which is two times the old xmax ...reference 1/8/04 why I made this change
xmax = epoxyheight/2-ant_thick;


if xmax > d/2
    error('xmax is greater than d/2 so the antenna elements will criss-cross at large angles!')
end

if ymax > 2*xmax
    error('ymax is greater than 2*xmax, so the epoxy will not cover the side of the antenna when the antenna is turned at a sharp angle (psi)!')
end

% centerchoice = menu('Centering on or off?','on','off');
% shapechoice = menu('Shape?','circle','parabola','ellipse','hyperbola');
% shapechoice = shapechoice+3;
% kchoice = menu('Curvature (k) ?','1','2','3','4','5','6','7','8','9','10');
% psichoice = menu('Angle ?','0','10','20','30','40','50','60','70','80','90');
% psichoice = (psichoice-1)*10;

centerchoice=1;
shapechoice=4;
kchoice=1;
psichoice=0;


% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 20; % MUST BE EVEN b/c linc = 2*totsum/numbsections.  It seems weird but having an even
% number of sections, will put an odd number of points because there is no middle section
% NOTE THAT THE NUBMER OF POINTS WILL BE NUMBSECTIONS+1
desiredcurves = 10; %if k only goes from 1:1 rather than 1:totcurves(ndp) than desiredcurves is overidden and
% therefore, only 1 curve will be created

if numangles==1
    psi_r_inc = 91; % just enough that only zero degrees will be calculated
else
    psi_r_inc = 90/(numangles-1); % increment value for the rotation angle in degrees
end

index = 0; % keeps track of the saving of epoxy and ae's for later use

for ndp = shapechoice:7  % alters shapes
    ndp

    % outputs points on box and the total number of curves that will be used
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    totcurves(ndp)
    %     display('program paused')
    %     pause

    for k= kchoice:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k

        a=1;
        b=1;

        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            b = d/2;
            a = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            clear ypoints
            xpoints = [];


            delxp = 1*10^(-6);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;

                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end
            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k) & i < (numbsections-3)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end

        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end

        % creates negative values and adds a zero point for the xpoints
        %nxpoints = -xpoints;
        xpoints = [fliplr(-xpoints) 0  xpoints];


        % end calculation of xpoints using arc length
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

        % calculates ypoints according to which shape the user specifies.
        % all shapes intersect through the origin
        if ndp == 3

            ypoints = zeros(size(xpoints));

        elseif ndp == 4     %circle points

            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 5 %parabola pts

            ypoints = a*(xpoints.*xpoints);

        elseif ndp == 6  %ellipse points

            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 7   %hyperbola points

            ypoints = (b^2+b^2/a^2*(xpoints.*xpoints)).^(0.5)-b;

        end

        %                 subplot(2,2,ndp-3)
        %                 scatter(xpoints,ypoints,'.')
        %                 axis([(-xmax-xmax/5) (xmax+xmax/5) -ymax/10 (ymax+ymax/5)]);
        %                 axis equal
        %                 if ndp==4
        %                     title('semicircular')
        %                 elseif ndp==5
        %                         title('parabolic')
        %                     elseif ndp==6
        %                             title('elliptical')
        %                         else
        %                             title('hyperbolic')
        %                         end
        %                     end
        %                 end
        %
        %                 pause

        %title(['shape',num2str(ndp),'   ','curve number',num2str(k)])

        [nxpoints,nypoints] = normalpointsv2(ndp,a,b,ant_thick,xpoints,ypoints);

        % %         close(gcf)
        % %         scatter(xpoints,ypoints,'r.');
        % %         axis equal
        % %         hold
        % %         scatter(nxpoints,nypoints,'bx');
        % %         axis equal
        % %
        % %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
        % %         pause


        %%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%% MAKING ANTENNA ELEMENT USING FEMLab FUNCTIONS
        % Note that the I am keeping the enclosing box x and y axes, so it is switched from normal

        for psi_r = psichoice:psi_r_inc:90   % rotates psi_r for a fixed shape and curvature

            index = index+1;

            psi_r


            %             aex=[fliplr(nxpoints),xpoints];
            %             aey=[fliplr(nypoints),ypoints];

            aex=[xpoints,fliplr(nxpoints)];
            aey=[ypoints,fliplr(nypoints)];

            % rotating neg. ae
            phi = psi_r*pi/180;

            rot_mat = [cos(phi) -sin(phi); sin(phi) cos(phi)];
            ae = [aex;aey];
            rae = rot_mat*ae;
            raex = rae(1,:);
            raey = rae(2,:)+d/2;


            %            %%%%%%%%%%%%% Plot which checks for correct rotation
            %             plot(aex,aey,'b',raex,raey,'r')
            %             axis equal
            %             title('check for correct rotation');
            %             pause
            %%%%%%%%%%%%%%
```

```
%             % step 1 -- create antenna elements
%             aer=line2(raex,raey);
%             ael=line2(raex,-raey);

% %             %%%%%%%%%% Plotting
% %                        close(gcf)
% %                        scatter(raex,raey,'r.');
% %                        axis equal
% %                        hold
% %                        scatter(raex,-raey,'b.');
% %                        axis equal
% %                        %pause
% %                        geomplot(aer)
% %                        axis equal
% %                        hold
% %                        geomplot(ael)
% %                        axis equal
% %                       % pause
% %             %%%%%%%%%%%%%%%%%%%%

%step2 -- create epoxy
%close(gcf)


%%%%%%%%%%%%%%%%%%%%%%%% code for creating outside antenna first...uses output of normalptsv2.m %%%%%%%%%%%%%%%%%%%%%%%%%

halfx = raex(numbsections+2:length(raex));
halfy = raey(numbsections+2:length(raey));
if centerchoice==1 % This choice will center the antenna element within the epoxy
    % This regime is fundamentally different from noncentering because it should expose
    % the difference between having a shield of epoxy vs free space and will give insight
    % into how much a shield is effective at pushing the current to the far-field

    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line...however, I should add about half the antenna thickness
    % to each side so that it doesn't make a sharp point near the tip of the antenna as in the case of the
    % elliptically shaped antenna elements
    if psi_r == 0
        xtra = ant_thick/2;
        %                    xtra = ant_thick*4/5;
        totx = [halfx (halfx(length(halfx))-xtra) (halfx(length(halfx))-xtra) fliplr(halfx) (halfx(1)+xtra) (halfx(1)+xtra) ];
        toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
        %                    epoxy = line2(totx,toty);
    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        halfx = fliplr(halfx);
        halfy = fliplr(halfy);
        widthant = max(halfx)-halfx(1);
        %                    xout = ant_thick*cos(phi);
        xout = ant_thick*cos(phi)*3/5;
        %                    xtra = (epoxyheight-widthant)/2;
        xtra = (epoxyheight-widthant)*4/5;
        %                    yout = ant_thick*sin(phi);
        yout = ant_thick*sin(phi)*3/5;

        % %                     totx = [halfx max(halfx) (max(halfx)+xtra) (max(halfx)+xtra) max(halfx) fliplr(halfx) halfx(1) (halfx(1)-xtra) (halfx(1)-xtra) halfx(1)];
        % %
        % %                    toty = [halfy (halfy(length(halfy))+ant_thick/2) (halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -fli
        % %

        %%%Sophisticated Modeling which is memory intensive
        totx = halfx;
        toty = halfy;

        %                   totx = [totx (max(halfx)+xout)];
        totx = [totx (halfx(length(halfx))+xout)];
        toty = [toty (halfy(length(halfy))+yout)];

        totx = [totx (max(halfx)+xout+xtra)];
        toty = [toty (halfy(length(halfy))+yout)];

        totx = [totx (max(halfx)+xout+xtra)];
        toty = [toty -(halfy(length(halfy))+yout)];

        %                   totx = [totx (max(halfx)+xout)];
        totx = [totx (halfx(length(halfx))+xout)];
        toty = [toty -(halfy(length(halfy))+yout)];

        totx = [totx fliplr(halfx)];
        toty = [toty -fliplr(halfy)];

        totx = [totx (halfx(1)-xout)];
        toty = [toty -(halfy(1)-yout)];

        totx = [totx (halfx(1)-xtra-xout)];
        toty = [toty -(halfy(1)-yout)];

        totx = [totx (halfx(1)-xtra-xout)];
        toty = [toty (halfy(1)-yout)];

        totx = [totx (halfx(1)-xout)];
        toty = [toty (halfy(1)-yout)];


        % % %                 Primitive Modeling
        % %
        % %                   totx = [(max(halfx)+xtra)];
        % %                   toty = [min(halfy)-ant_thick/2];
        % %
        % %                   totx = [totx (max(halfx)+xtra)];
        % %                   toty = [toty max(halfy)];
        % %
        % %                   totx = [totx (max(halfx)+xtra+ant_thick)];
        % %                   toty = [toty max(halfy)];
        % %
        % %                   totx = [totx (max(halfx)+xtra+ant_thick)];
        % %                   toty = [toty -max(halfy)];
        % %
        % %                   totx = [totx (max(halfx)+xtra)];
        % %                   toty = [toty -max(halfy)];
        % %
        % %                   totx = [totx (max(halfx)+xtra)];
        % %                   toty = [toty -(min(halfy)-ant_thick/2)];
        % %
        % %
        % %                   totx = [totx (halfx(1)-xtra)];
        % %                   toty = [toty -(min(halfy)-ant_thick/2)];
        % %
        % %
        % %                   totx = [totx (halfx(1)-xtra)];
        % %                   toty = [toty (min(halfy)-ant_thick/2)];


    end
```

```
else

    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line
    if psi_r == 0
        totx = [fliplr(halfx) halfx];
        toty = [-fliplr(halfy) halfy];

    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        xtra = min(raex)+2*xmax;
        totx = [fliplr(halfx) halfx  xtra xtra];
        toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];

    end

end
%%%%%%%%%%%%%%%%%%%%%%%% end create outside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

% % %%%%%%%%%%%%%%%%%%%%%%%% code for creating inside antenna first...uses output of normalptsv4.m %%%%%%%%%%%%%%%%%%%%%%%%%
% %
% %             halfx = raex(1:numbsections+1);
% %             halfy = raey(1:numbsections+1);
% %             if centerchoice==1 % This choice will center the antenna element within the epoxy
% %                 % This regime is fundamentally different from noncentering because it should expose
% %                 % the difference between having a shield of epoxy vs free space and will give insight
% %                 % into how much a shield is effective at pushing the current to the far-field
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     widthant = max(halfx)-halfx(1);
% %                     xtra = (2*xmax-widthant)/2;
% %                     totx = [halfx (max(halfx)+xtra) (max(halfx)+xtra) fliplr(halfx) (halfx(1)-xtra) (halfx(1)-xtra)];
% %                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %
% %
% %             else
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     xtra = min(raex)+2*xmax;
% %                     totx = [fliplr(halfx) halfx  xtra xtra];
% %                     toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %             end
% % %%%%%%%%%%%%%%%%%%%%%%%% end create inside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

% %         %%%%%%%% GOOD GRAPHING CODE
% %         close(gcf)
% %         subplot(2,2,1)
% %         scatter(xpoints,ypoints,'r.');
% %         axis equal
% %         hold
% %         scatter(nxpoints,nypoints,'bx');
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %
% %         subplot(2,2,2)
% %         plot(totx,toty ,'r'),axis equal
% %
% %         subplot(2,2,3)
% %         scatter(-raey,-raex,'r.');
% %         axis equal
% %         hold
% %         scatter(raey,-raex,'b.');
% %         plot(toty,-totx,'r')
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %         %%%%%%%%%%%%%%%%%%%%%%


% CALCULATE THE AREA FOR EACH ANTENNA
% step 1 -- create antenna elements
aer=line2(raex,raey);
clear fem
% Geometry
fem.geom = aer;
fem.mesh = meshinit(fem);
% Integrate on subdomains
ae_vol = postint(fem,'1');



%%%%%%%%%%%BEGIN 2D FEA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calling file rather than inserting code

commandline_v9



%%%%%%%%%%%END 2D FEA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        end
    end
end

save resultsXIX.dat results -ascii

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%  GRAPHING  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];

for j = 1:length(results)
    if results(j,4)==3
        sheetresults = [sheetresults;results(j,:)];
    elseif results(j,4)==4
        circleresults = [circleresults;results(j,:)];
    elseif results(j,4)==5
        parabolaresults = [parabolaresults;results(j,:)];
    elseif results(j,4)==6
        ellipseresults = [ellipseresults;results(j,:)];
    elseif results(j,4)==7
        hyperbolaresults = [hyperbolaresults;results(j,:)];
    end
end

for j = 4:7
    if j==3
        shaperesults = sheetresults;
    elseif j==4
        shaperesults = circleresults;
    elseif j==5
        shaperesults = parabolaresults;
    elseif j==6
        shaperesults = ellipseresults;
    elseif j==7
        shaperesults = hyperbolaresults;
    end

    figure(1)
    subplot(2,2,j-3)
    if j~=3
    xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
    ylin = linspace(min(shaperesults(:,6)),max(shaperesults(:,6)),90/psi_r_inc+1);

    [Y,X] = meshgrid(ylin,xlin);

    Z = griddata(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),X,Y);
    surf(X,Y,Z);
    if j==3
        title('sheet');
    elseif j==4
        title('circular','FontSize',15);
    elseif j==5
        title('parabolic','FontSize',15);
    elseif j==6
        title('elliptical','FontSize',15);
    elseif j==7
        title('hyperbolic','FontSize',15);
    end
    hold on;
    xlabel('curvature (index)','FontSize',13);
    ylabel('psi (angle)','FontSize',13);
    zlabel('potential difference (V)','FontSize',13);
    set(gca,'FontSize',13)
    axis([0 10 0 90 6e-4 7.5e-4])
    %colorbar;
    colormap gray;
    rotate3d
    end
    plot3(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),'o')
    hold off;

end

set(gcf,'Color',[1 1 1])
```

```
% commandline_v9.m
% Brian Wessel
% Feb. 18, 2004


% Uses a better way to pull boundary
% voltages from the resultant data

% clear all
% close all
%
% load raex.dat
% load raey.dat
% load totx.dat
% load toty.dat

% % %%%%%% creates shield
% % totx(45) = 1.9e-3;
% % totx(46) = 1.9e-3;
% % totx(47) = 4e-3;
% % totx(48) = 4e-3;
% % totx(49) = 1.9e-3;
% % totx(50) = 1.9e-3;
% %
% % toty(46) = -4e-3;
% % toty(46) = -8e-3;
% % toty(47) = -8e-3;
% % toty(48) = 8e-3;
% % toty(49) = 8e-3;
% % toty(50) = 4e-3;
% % %%%%% end create shield
%
% scatter(raey,-raex,'b.');
% hold
% plot(toty,-totx,'r')
% axis equal
% scatter(-raey,-raex,'r.');
% axis equal
% pause

% close(gcf)


%Constants
s_s  = 100/222;
s_e  = 1e-2;
s_ae = 1e2;
curr = 1e-3;

% creating the positive ant. element
% x = [3.3e-2 3.5e-2 3.5e-2 3.3e-2];
% y = [-1e-2 -1e-2 1e-2 1e-2];
pae = [2 length(raex) raey (-raex+6.5e-2) zeros(1,2*(length(totx)-length(raey)))]';

% creating the negative ant. element
nae = [2 length(raex) -raey (-raex+6.5e-2) zeros(1,2*(length(totx)-length(raey)))]';

% creating epoxy
% epoxy = [2 4 -3.3e-2 3.3e-2 3.3e-2 -3.3e-2 -1e-2 -1e-2 1e-2 1e-2]';
epoxy = [2 length(totx) toty (-totx+6.5e-2)]';

% creating a circle with radius = 7.5e-2 m
circle = [1 0 0 7.5e-2 zeros(1,length(epoxy)-4)]';

% Testing code
% pdecirc(0, 0 ,7.5e-2)
% pdepoly([3.5e-2, 3.3e-2, 3.3e-2, 3.5e-2],[-1e-2 -1e-2 1e-2 1e-2])
% pdepoly([-3.5e-2, -3.3e-2, -3.3e-2, -3.5e-2],[-1e-2 -1e-2 1e-2 1e-2])

csg = [nae pae epoxy circle];


% dgm = decsg(csg,['circle+pae+nae+epoxy'],[['nae   ']' ['pae   ']' ['epoxy ']' ['circle']'])
dgm = decsg(csg);

% wgeom(dgm,'geomfile')
% break

% subplot(1,2,1)
% pdegplot(dgm)
% axis equal

[p,e,t]=initmesh(dgm,'jiggle','off','hgrad',1.25);
% q=pdetriq(p,t);
% subplot(1,2,1)
% pdeplot(p,e,t,'xydata',q,'colorbar','on','xystyle','flat')
% axis equal
%
% pause
% subplot(1,2,2)
p1=jigglemesh(p,e,t,'opt','minimum','iter',inf);
q=pdetriq(p1,t);
% % figure(1)
% % pdeplot(p1,e,t,'xydata',q,'colorbar','on','xystyle','flat')
% % axis equal

% pause

p=p1;

% sphere | pae | epoxy | nae
% % c = ['100/222!1e2!1e-2!1e2'];
% % a = ['0!0!0!0'];
% % f = ['0!(1e-3/2.0904e-006)!0!(-1e-3/2.0904e-006)'];

c = [num2str(s_s),'!',num2str(s_ae),'!',num2str(s_e),'!',num2str(s_ae)]
a = ['0!0!0!0']
f = ['0!',num2str(curr/ae_vol),'!0!(-',num2str(curr/ae_vol),')']


% Note only the last columns are the exterior boundaries
% The other internal ones must have contrived data
cols = size(dgm,2);
b = double([0 1 1 1 1 1 '0' '0' '1' '0']');
for i=1:(cols-5)
    % This is the contrived data I found from the output of the PDETool
    b = [b double([0 1 1 1 1 1 '0' '0' '1' '0']')];
end

b(:,cols-3) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols-2) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols-1) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols)   = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');

u = assempde(b,p,e,t,c,a,f);

% demean signal (artificial)
u = u-mean(u);

% [u,p,e,t]=adaptmesh(dgm,b,c,a,f,'Ngen',2);

[cgxu,cgyu]=pdecgrad(p,t,c,u);
% [ux,uy] = pdegrad(p,t,u);
% uu = [ux',uy'];
uu = [cgxu',cgyu'];

% demean signal (artificial)
uu(:,1) = uu(:,1)-mean(uu(:,1));
uu(:,2) = uu(:,2)-mean(uu(:,2));
% % figure(2)
% % pdeplot(p,e,t,'xydata',u,'mesh','off','contour','on','levels',30);
% % title('Voltage')
% % set(gcf,'Color',[1 1 1])
% % axis equal
% %
% % figure(3)
% % pdeplot(p,e,t,'xydata',uu,'mesh','off','contour','on','levels',200);
% % title('Current Density');
% % axis equal

% i=pdesde(e);
% finding boundary points and values
m=1;
for j = 1:length(p)
    bpts = norm(p(:,j));
%     display('program paused')
%     pause
    if (0.075-bpts)<1e-3
%         (0.075-bpts)
%         display('true')
%         pause
        boundv(m) = u(j);
        boundp(1,m) = p(1,j);
        boundp(2,m) = p(2,j);
%             display('program paused - inside loop')
%             pause
        x=p(1,j);
        y=p(2,j);
        theta(m) = atan(y/x);
        % fixing atan function so it goes from 0 to 2*pi rather than -pi/2 to pi/2
        if x<0 & y>0
            theta(m) = theta(m)+pi;
        elseif x<0 & y<0
            theta(m) = theta(m)-pi;
        end
        m = m+1;
    end

end
% % %%%%%%%%%%%%%%%%%%% GRAPHING
% % figure(2)
% % subplot(2,2,1)
% % plot(abs(boundv))
% % title('boundary voltage')
% %
% % subplot(2,2,2)
% % % polar(theta,boundv+1.1*min(boundv),'r.')
% % polar(theta,abs(boundv),'r.')
% % title('boundary voltage')
% %
% % subplot(2,2,3)
% % stem3(boundp(1,:),boundp(2,:),boundv,'ro')
% % title('boundary voltage')
% %
% % figure(3)
% % polar(theta,abs(boundv),'r.')
% % title('boundary voltage')
% % set(gcf,'Color',[1 1 1]);
% % %%%%%%%%%%%%%%%%%%%%
results = [results; (max(boundv)-min(boundv)) max(boundv) min(boundv) ndp k psi_r];

clear p p1 q t u uu cgyu cgxu
pack
% display('End of 2D FEA iteration - PROGRAM PAUSED')
% pause
% subplot(2,2,3)
% pdecont(p,t,u,20);
% axis equal
```

## .12 PDE TOOL CODE - NO EPOXY

Contains the following files:

- create_antennas_noEpoxy.m
- commandline_v9.m (refer to .11)
- shapesfundersq.m (refer to .1)
- shapestotarclength.m (refer to .2)
- shapesptsopt_v2.m (refer to .2)
- normalpointsv2.m (refer to .3)

```
% create_antennas_noEpoxy.m
% Brian Wessel
% creates geometry for input to MatLab PDE solver
clear all
close all

% xmax = 0.2;
% ymax = 0.1;
% zmax = 0.9;
% d = 0.9;
% ant_thick = 0.03;

results = [];

%xmax = 2e-3;
ymax = 1e-3;
zmax = 4.5e-3;
d = 9e-3;
ant_thick = 7.5e-4;
% ant_thick = 2.5e-4;
numangles = 10;

epoxyheight=4e-3; % which is two times the old xmax ...reference 1/8/04 why I made this change
xmax = epoxyheight/2-ant_thick;


if xmax > d/2
    error('xmax is greater than d/2 so the antenna elements will criss-cross at large angles!')
end

if ymax > 2*xmax
    error('ymax is greater than 2*xmax, so the epoxy will not cover the side of the antenna when the antenna is turned at a sharp angle (psi)!')
end

% centerchoice = menu('Centering on or off?','on','off');
% shapechoice = menu('Shape?','circle','parabola','ellipse','hyperbola');
% shapechoice = shapechoice+3

% kchoice = menu('Curvature (k) ?','1','2','3','4','5','6','7','8','9','10');
% psichoice = menu('Angle ?','0','10','20','30','40','50','60','70','80','90');
% psichoice = (psichoice-1)*10;

centerchoice=1;
shapechoice=4;
kchoice=1;
psichoice=0;


% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 20; % MUST BE EVEN b/c linc = 2*totsum/numbsections.  It seems weird but having an even
% number of sections, will put an odd number of points because there is no middle section
% NOTE THAT THE NUBMER OF POINTS WILL BE NUMBSECTIONS+1
desiredcurves = 10; %if k only goes from 1:1 rather than 1:totcurves(ndp) than desiredcurves is overidden and
% therefore, only 1 curve will be created

if numangles==1
    psi_r_inc = 91; % just enough that only zero degrees will be calculated
else
    psi_r_inc = 90/(numangles-1); % increment value for the rotation angle in degrees
end

index = 0; % keeps track of the saving of epoxy and ae's for later use

for ndp = shapechoice:7  % alters shapes
    ndp

    % outputs points on box and the total number of curves that will be used
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    totcurves(ndp)
    %     display('program paused')
    %     pause


    for k= kchoice:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k


        a=1;
        b=1;

        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            b = d/2;
            a = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            clear ypoints
            xpoints = [];


            delxp = 1*10^(-6);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;
                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end
            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k) & i < (numbsections-3)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end
        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end

        % creates negative values and adds a zero point for the xpoints
        %nxpoints = -xpoints;
        xpoints = [fliplr(-xpoints) 0  xpoints];


        % end calculation of xpoints using arc length
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



        % calculates ypoints according to which shape the user specifies.
        % all shapes intersect through the origin
        if ndp == 3

            ypoints = zeros(size(xpoints));

        elseif ndp == 4     %circle points

            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 5 %parabola pts

            ypoints = a*(xpoints.*xpoints);

        elseif ndp == 6  %ellipse points

            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 7   %hyperbola points

            ypoints = (b^2+b^2/a^2*(xpoints.*xpoints)).^(0.5)-b;

        end

        %                 subplot(2,2,ndp-3)
        %                 scatter(xpoints,ypoints,'.')
        %                 axis([(-xmax-xmax/5) (xmax+xmax/5) -ymax/10 (ymax+ymax/5)]);
        %                 axis equal
        %                 if ndp==4
        %                     title('semicircular')
        %                 elseif ndp==5
        %                         title('parabolic')
        %                     elseif ndp==6
        %                             title('elliptical')
        %                         else
        %                             title('hyperbolic')
        %                         end
        %                     end
        %                 end
        %
        %                 pause

        %title(['shape',num2str(ndp),'   ','curve number',num2str(k)])

        [nxpoints,nypoints] = normalpointsv2(ndp,a,b,ant_thick,xpoints,ypoints);

        % %         close(gcf)
        % %         scatter(xpoints,ypoints,'r.');
        % %         axis equal
        % %         hold
        % %         scatter(nxpoints,nypoints,'bx');
        % %         axis equal
        % %
        % %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
        % %         pause


        %%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%% MAKING ANTENNA ELEMENT USING FEMLab FUNCTIONS
        % Note that the I am keeping the enclosing box x and y axes, so it is switched from normal


        for psi_r = psichoice:psi_r_inc:90   % rotates psi_r for a fixed shape and curvature

            index = index+1;

            psi_r


            %             aex=[fliplr(nxpoints),xpoints];
            %             aey=[fliplr(nypoints),ypoints];

            aex=[xpoints,fliplr(nxpoints)];
            aey=[ypoints,fliplr(nypoints)];

            % rotating neg. ae
            phi = psi_r*pi/180;

            rot_mat = [cos(phi) -sin(phi); sin(phi) cos(phi)];
            ae = [aex;aey];
            rae = rot_mat*ae;
            raex = rae(1,:);
            raey = rae(2,:)+d/2;


            %            %%%%%%%%%%%% Plot which checks for correct rotation
            %             plot(aex,aey,'b',raex,raey,'r')
            %             axis equal
            %             title('check for correct rotation');
            %             pause
            %%%%%%%%%%%%%%
            %             % step 1 -- create antenna elements
            %             aer=line2(raex,raey);
            %             ael=line2(raex,-raey);
            % %             %%%%%%%%%% Plotting
            % %                         close(gcf)
            % %                         scatter(raex,raey,'r.');
            % %                         axis equal
            % %                         hold
            % %                         scatter(raex,-raey,'b.');
            % %                         axis equal
            % %                         %pause
            % %                         geomplot(aer)
            % %                         axis equal
            % %                         hold
            % %                         geomplot(ael)
            % %                         axis equal
            % %                        % pause
            % %             %%%%%%%%%%%%%%%%%%%%

            %step2 -- create epoxy
            %close(gcf)


            %%%%%%%%%%%%%%%%%%%%%%%% code for creating outside antenna first...uses output of normalptsv2.m %%%%%%%%%%%%%%%%%%%%%%%%%%
            halfx = raex(numbsections+2:length(raex));
            halfy = raey(numbsections+2:length(raey));
            if centerchoice==1 % This choice will center the antenna element within the epoxy
                % This regime is fundamentally different from noncentering because it should expose
                % the difference between having a shield of epoxy vs free space and will give insight
                % into how much a shield is effective at pushing the current to the far-field

                %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
                % will be too close to connect with a line...however, I should add about half the antenna thickness
                % to each side so that it doesn't make a sharp point near the tip of the antenna as in the case of the
                % elliptically shaped antenna elements
                if psi_r == 0
                    xtra = ant_thick/2;
                    %                     xtra = ant_thick*4/5;
                    totx = [halfx (halfx(length(halfx))-xtra) (halfx(length(halfx))-xtra) fliplr(halfx) (halfx(1)+xtra) (halfx(1)+xtra) ];
                    toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
                    %                     epoxy = line2(totx,toty);
                else
                    % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
                    % therefore cause a problem.
                    % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
                    % old way #2 -- xtra = max(raex)+1/5*xmax;

                    %xtra = min(raex)+2*xmax+ant_thick;
                    halfx = fliplr(halfx);
                    halfy = fliplr(halfy);
                    widthant = max(halfx)-halfx(1);
                    %                     xout = ant_thick*cos(phi);
                    xout = ant_thick*cos(phi)*3/5;
                    %                     xtra = (epoxyheight-widthant)/2;
                    xtra = (epoxyheight-widthant)*4/5;
                    %                     yout = ant_thick*sin(phi);
                    yout = ant_thick*sin(phi)*3/5;

                    % %                      totx = [halfx max(halfx) (max(halfx)+xtra) (max(halfx)+xtra) max(halfx) fliplr(halfx) halfx(1) (halfx(1)-xtra) (halfx(1)-xtra) halfx(1)];
                    % %
                    % %                     toty = [halfy (halfy(length(halfy))+ant_thick/2) (halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -fli
                    % %

                    %%%Sophisticated Modeling which is memory intensive
                    totx = halfx;
                    toty = halfy;

                    %                    totx = [totx (max(halfx)+xout)];
                    totx = [totx (halfx(length(halfx))+xout)];
                    toty = [toty (halfy(length(halfy))+yout)];

                    totx = [totx (max(halfx)+xout+xtra)];
                    toty = [toty (halfy(length(halfy))+yout)];

                    totx = [totx (max(halfx)+xout+xtra)];
                    toty = [toty -(halfy(length(halfy))+yout)];

                    %                    totx = [totx (max(halfx)+xout)];
                    totx = [totx (halfx(length(halfx))+xout)];
                    toty = [toty -(halfy(length(halfy))+yout)];

                    totx = [totx fliplr(halfx)];
                    toty = [toty -fliplr(halfy)];

                    totx = [totx (halfx(1)-xout)];
                    toty = [toty -(halfy(1)-yout)];

                    totx = [totx (halfx(1)-xtra-xout)];
                    toty = [toty -(halfy(1)-yout)];

                    totx = [totx (halfx(1)-xtra-xout)];
                    toty = [toty (halfy(1)-yout)];

                    totx = [totx (halfx(1)-xout)];
                    toty = [toty (halfy(1)-yout)];


                    % % %                  Primitive Modeling
                    % %
                    % %                    totx = [(max(halfx)+xtra)];
                    % %                    toty = [min(halfy)-ant_thick/2];
                    % %
                    % %                    totx = [totx (max(halfx)+xtra)];
                    % %                    toty = [toty max(halfy)];
                    % %
                    % %                    totx = [totx (max(halfx)+xtra+ant_thick)];
                    % %                    toty = [toty max(halfy)];
                    % %
                    % %                    totx = [totx (max(halfx)+xtra+ant_thick)];
                    % %                    toty = [toty -max(halfy)];
                    % %
                    % %                    totx = [totx (max(halfx)+xtra)];
                    % %                    toty = [toty -max(halfy)];
                    % %
                    % %                    totx = [totx (max(halfx)+xtra)];
                    % %                    toty = [toty -(min(halfy)-ant_thick/2)];
                    % %
                    % %
                    % %                    totx = [totx (halfx(1)-xtra)];
                    % %                    toty = [toty -(min(halfy)-ant_thick/2)];
                    % %
                    % %
                    % %                    totx = [totx (halfx(1)-xtra)];
                    % %                    toty = [toty (min(halfy)-ant_thick/2)];


                end
```

```
else

    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line
    if psi_r == 0
        totx = [fliplr(halfx) halfx];
        toty = [-fliplr(halfy) halfy];

    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        xtra = min(raex)+2*xmax;
        totx = [fliplr(halfx) halfx  xtra xtra];
        toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];

    end

end
%%%%%%%%%%%%%%%%%%%%%%%% end create outside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%%

% % %%%%%%%%%%%%%%%%%%%%%%%% code for creating inside antenna first...uses output of normalptsv4.m %%%%%%%%%%%%%%%%%%%%%%%%%%
% %
% %             halfx = raex(1:numbsections+1);
% %             halfy = raey(1:numbsections+1);
% %             if centerchoice==1 % This choice will center the antenna element within the epoxy
% %                 % This regime is fundamentally different from noncentering because it should expose
% %                 % the difference between having a shield of epoxy vs free space and will give insight
% %                 % into how much a shield is effective at pushing the current to the far-field
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     widthant = max(halfx)-halfx(1);
% %                     xtra = (2*xmax-widthant)/2;
% %                     totx = [halfx (max(halfx)+xtra) (max(halfx)+xtra) fliplr(halfx) (halfx(1)-xtra) (halfx(1)-xtra)];
% %                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %
% %
% %             else
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     xtra = min(raex)+2*xmax;
% %                     totx = [fliplr(halfx) halfx  xtra xtra];
% %                     toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %             end
% % %%%%%%%%%%%%%%%%%%%%%%%% end create inside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%%

% %         %%%%%%% GOOD GRAPHING CODE
% %         close(gcf)
% %         subplot(2,2,1)
% %         scatter(xpoints,ypoints,'r.');
% %         axis equal
% %         hold
% %         scatter(nxpoints,nypoints,'bx');
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %
% %         subplot(2,2,2)
% %         plot(totx,toty ,'r'),axis equal
% %
% %         subplot(2,2,3)
% %         scatter(-raey,-raex,'r.');
% %         axis equal
% %         hold
% %         scatter(raey,-raex,'b.');
% %         plot(toty,-totx,'r')
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %         %%%%%%%%%%%%%%%%%%%%%


% CALCULATE THE AREA FOR EACH ANTENNA
% step 1 -- create antenna elements
aer=line2(raex,raey);
clear fem
% Geometry
fem.geom = aer;
fem.mesh = meshinit(fem);
% Integrate on subdomains
ae_vol = postint(fem,'1');



%%%%%%%%%%%BEGIN 2D FEA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calling file rather than inserting code

commandline_v9_noepoxy



%%%%%%%%%%%END 2D FEA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
        end
    end
end

save resultsXIX.dat results -ascii

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%  GRAPHING  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];

for j = 1:length(results)
    if results(j,4)==3
        sheetresults = [sheetresults;results(j,:)];
    elseif results(j,4)==4
        circleresults = [circleresults;results(j,:)];
    elseif results(j,4)==5
        parabolaresults = [parabolaresults;results(j,:)];
    elseif results(j,4)==6
        ellipseresults = [ellipseresults;results(j,:)];
    elseif results(j,4)==7
        hyperbolaresults = [hyperbolaresults;results(j,:)];
    end
end

for j = 4:7
    if j==3
        shaperesults = sheetresults;
    elseif j==4
        shaperesults = circleresults;
    elseif j==5
        shaperesults = parabolaresults;
    elseif j==6
        shaperesults = ellipseresults;
    elseif j==7
        shaperesults = hyperbolaresults;
    end

    figure(1)
    subplot(2,2,j-3)
    if j~=3
    xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
    ylin = linspace(min(shaperesults(:,6)),max(shaperesults(:,6)),90/psi_r_inc+1);

    [Y,X] = meshgrid(ylin,xlin);

    Z = griddata(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),X,Y);
    surf(X,Y,Z);
    if j==3
        title('sheet');
    elseif j==4
        title('circular','FontSize',15);
    elseif j==5
        title('parabolic','FontSize',15);
    elseif j==6
        title('elliptical','FontSize',15);
    elseif j==7
        title('hyperbolic','FontSize',15);
    end
    hold on;
    xlabel('curvature (index)','FontSize',13);
    ylabel('psi (angle)','FontSize',13);
    zlabel('potential difference (V)','FontSize',13);
    set(gca,'FontSize',13)
    axis([0 10 0 90 6e-4 7.5e-4])
    %colorbar;
    colormap gray;
    rotate3d
    end
    plot3(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),'o'
    hold off;

end
set(gcf,'Color',[1 1 1])
```

## .13 PDE TOOL CODE - REFLECTOR

Contains the following files:

- create_antennas_hasReflector.m
- commandline_v9.m (refer to .11)
- shapesfundersq.m (refer to .1)
- shapestotarclength.m (refer to .2)
- shapesptsopt_v2.m (refer to .2)
- normalpointsv2.m (refer to .3)

```
% create_antennas_hasReflector.m
% Brian Wessel
% creates geometry for input to MatLab PDE solver
% This code creates a reflector at the bottom of the epoxy

clear all
close all

% xmax = 0.2;
% ymax = 0.1;
% zmax = 0.9;
% d = 0.9;
% ant_thick = 0.03;

results = [];

%xmax = 2e-3;
ymax = 1e-3;
zmax = 4.5e-3;
d = 9e-3;
ant_thick = 7.5e-4;
% ant_thick = 2.5e-4;
numangles = 10;

epoxyheight=4e-3; % which is two times the old xmax ...reference 1/8/04 why I made this change
xmax = epoxyheight/2-ant_thick;


if xmax > d/2
    error('xmax is greater than d/2 so the antenna elements will criss-cross at large angles!')
end

if ymax > 2*xmax
    error('ymax is greater than 2*xmax, so the epoxy will not cover the side of the antenna when the antenna is turned at a sharp angle (psi)!')
end

% centerchoice = menu('Centering on or off?','on','off');
% shapechoice = menu('Shape?','circle','parabola','ellipse','hyperbola');
% shapechoice = shapechoice+3;
% kchoice = menu('Curvature (k) ?','1','2','3','4','5','6','7','8','9','10');
% psichoice = menu('Angle ?','0','10','20','30','40','50','60','70','80','90');
% psichoice = (psichoice-1)*10;

centerchoice=1;
shapechoice=4;
kchoice=1;
psichoice=0;


% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 20; % MUST BE EVEN b/c linc = 2*totsum/numbsections.  It seems weird but having an even
% number of sections, will put an odd number of points because there is no middle section
% NOTE THAT THE NUBMER OF POINTS WILL BE NUMBSECTIONS+1
desiredcurves = 10; %if k only goes from 1:1 rather than 1:totcurves(ndp) than desiredcurves is overidden and
% therefore, only 1 curve will be created

if numangles==1
    psi_r_inc = 91; % just enough that only zero degrees will be calculated
else
    psi_r_inc = 90/(numangles-1); % increment value for the rotation angle in degrees
end

index = 0; % keeps track of the saving of epoxy and ae's for later use

progress = 0;
waitbar(progress/360,'2D FE Progress')

for ndp = shapechoice:7  % alters shapes
    ndp

    % outputs points on box and the total number of curves that will be used
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    totcurves(ndp)
    %     display('program paused')
    %     pause

    for k= kchoice:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k

        a=1;
        b=1;

        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            b = d/2;
            a = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            clear ypoints
            xpoints = [];


            delxp = 1*10^(-6);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;
                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end


            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k) & i < (numbsections-3)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end

        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end

        % creates negative values and adds a zero point for the xpoints
        %nxpoints = -xpoints;
        xpoints = [fliplr(-xpoints) 0  xpoints];


        % end calculation of xpoints using arc length
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



        % calculates ypoints according to which shape the user specifies.
        % all shapes intersect through the origin
        if ndp == 3

            ypoints = zeros(size(xpoints));

        elseif ndp == 4     %circle points

            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 5 %parabola pts

            ypoints = a*(xpoints.*xpoints);

        elseif ndp == 6  %ellipse points

            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 7   %hyperbola points

            ypoints = (b^2+b^2/a^2*(xpoints.*xpoints)).^(0.5)-b;

        end

        %                 subplot(2,2,ndp-3)
        %                 scatter(xpoints,ypoints,'.')
        %                 axis([(-xmax-xmax/5) (xmax+xmax/5) -ymax/10 (ymax+ymax/5)]);
        %                 axis equal
        %                 if ndp==4
        %                     title('semicircular')
        %                 elseif ndp==5
        %                         title('parabolic')
        %                     elseif ndp==6
        %                             title('elliptical')
        %                         else
        %                             title('hyperbolic')
        %                         end
        %                     end
        %                 end
        %
        %                 pause

        %title(['shape',num2str(ndp),'   ','curve number',num2str(k)])

        [nxpoints,nypoints] = normalpointsv2(ndp,a,b,ant_thick,xpoints,ypoints);

        % %         close(gcf)
        % %         scatter(xpoints,ypoints,'r.');
        % %         axis equal
        % %         hold
        % %         scatter(nxpoints,nypoints,'bx');
        % %         axis equal
        % %
        % %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
        % %         pause


        %%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%% MAKING ANTENNA ELEMENT USING FEMLab FUNCTIONS
        % Note that the I am keeping the enclosing box x and y axes, so it is switched from normal


        for psi_r = psichoice:psi_r_inc:90   % rotates psi_r for a fixed shape and curvature

            index = index+1;

            psi_r


            %             aex=[fliplr(nxpoints),xpoints];
            %             aey=[fliplr(nypoints),ypoints];

            aex=[xpoints,fliplr(nxpoints)];
            aey=[ypoints,fliplr(nypoints)];

            % rotating neg. ae
            phi = psi_r*pi/180;

            rot_mat = [cos(phi) -sin(phi); sin(phi) cos(phi)];
            ae = [aex;aey];
            rae = rot_mat*ae;
            raex = rae(1,:);
            raey = rae(2,:)+d/2;


            %            %%%%%%%%%%%%% Plot which checks for correct rotation
            %             plot(aex,aey,'b',raex,raey,'r')
```

```
            %             axis equal
            %             title('check for correct rotation');
            %             pause
            %%%%%%%%%%%%%%
            %             % step 1 -- create antenna elements
            %             aer=line2(raex,raey);
            %             ael=line2(raex,-raey);
% %                         %%%%%%%%%% Plotting
% %                                     close(gcf)
% %                                     scatter(raex,raey,'r.');
% %                                     axis equal
% %                                     hold
% %                                     scatter(raex,-raey,'b.');
% %                                     axis equal
% % %                                     %pause
% % %                                     geomplot(aer)
% % %                                     axis equal
% % %                                     hold
% % %                                     geomplot(ael)
% % %                                     axis equal
% % %                                    % pause
% %                         %%%%%%%%%%%%%%%%%%%%
            %step2 -- create epoxy
            %close(gcf)


            %%%%%%%%%%%%%%%%%%%%%%%% code for creating outside antenna first...uses output of normalptsv2.m %%%%%%%%%%%%%%%%%%%%%%%%%
            halfx = raex(numbsections+2:length(raex));
            halfy = raey(numbsections+2:length(raey));
            if centerchoice==1 % This choice will center the antenna element within the epoxy
                % This regime is fundamentally different from noncentering because it should expose
                % the difference between having a shield of epoxy vs free space and will give insight
                % into how much a shield is effective at pushing the current to the far-field
                %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
                % will be too close to connect with a line...however, I should add about half the antenna thickness
                % to each side so that it doesn't make a sharp point near the tip of the antenna as in the case of the
                % elliptically shaped antenna elements
%                 if psi_r == 0
%                     xtra = ant_thick/2;
%                     %                    xtra = ant_thick*4/5;
%                     totx = [halfx (halfx(length(halfx))-xtra) (halfx(length(halfx))-xtra) fliplr(halfx) (halfx(1)+xtra) (halfx(1)+xtra) ];
%                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
%                     %                    epoxy = line2(totx,toty);
%                 else
                    % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
                    % therefore cause a problem.
                    % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
                    % old way #2 -- xtra = max(raex)+1/5*xmax;
                    %xtra = min(raex)+2*xmax+ant_thick;
                    halfx = fliplr(halfx);
                    halfy = fliplr(halfy);
                    widthant = max(halfx)-halfx(1);
                    %                   xout = ant_thick*cos(phi);
                    xout = ant_thick*cos(phi)*3/5;
                    %                   xtra = (epoxyheight-widthant)/2;
                    xtra = (epoxyheight-widthant)*4/5;
                    %                   yout = ant_thick*sin(phi);
                    yout = ant_thick*sin(phi)*3/5;
                    %reflector thickness and width
                    rthick = 2e-3;
                    rwidth = 1e-2;
                    % %                    totx = [halfx max(halfx) (max(halfx)+xtra) (max(halfx)+xtra) max(halfx) fliplr(halfx) halfx(1) (halfx(1)-xtra) (halfx(1)-xtra) halfx(1)];
                    % %
                    % %                   toty = [halfy (halfy(length(halfy))+ant_thick/2) (halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -fli
                    % %
                    %%%Sophisticated Modeling which is memory intensive
                    totx = halfx;
                    toty = halfy;
                    %                  totx = [totx (max(halfx)+xout)];
                    totx = [totx (halfx(length(halfx))+xout)];
                    toty = [toty (halfy(length(halfy))+yout)];
                    totx = [totx (max(halfx)+xout)];
                    toty = [toty (halfy(length(halfy))+rwidth)];
                    totx = [totx (max(halfx)+xout+rthick)];
                    toty = [toty (halfy(length(halfy))+rwidth)];
                    totx = [totx (max(halfx)+xout+rthick)];
                    toty = [toty -(halfy(length(halfy))+rwidth)];
                    %                  totx = [totx (max(halfx)+xout)];
                    totx = [totx (max(halfx)+xout)];
                    toty = [toty -(halfy(length(halfy))+rwidth)];
                    totx = [totx (halfx(length(halfx))+xout)];
                    toty = [toty -(halfy(length(halfy))+yout)];
                    totx = [totx fliplr(halfx)];
                    toty = [toty -fliplr(halfy)];
                    totx = [totx (halfx(1)-xout)];
                    toty = [toty -(halfy(1)-yout)];
                    totx = [totx (halfx(1)-xtra-xout)];
                    toty = [toty -(halfy(1)-yout)];
                    totx = [totx (halfx(1)-xtra-xout)];
                    toty = [toty (halfy(1)-yout)];
                    totx = [totx (halfx(1)-xout)];
                    toty = [toty (halfy(1)-yout)];

                    % % %                   Primitive Modeling
                    % %
                    % %                   totx = [(max(halfx)+xtra)];
                    % %                   toty = [min(halfy)-ant_thick/2];
                    % %
                    % %                   totx = [totx (max(halfx)+xtra)];
                    % %                   toty = [toty max(halfy)];
                    % %
                    % %                   totx = [totx (max(halfx)+xtra+ant_thick)];
                    % %                   toty = [toty max(halfy)];
                    % %
                    % %                   totx = [totx (max(halfx)+xtra+ant_thick)];
                    % %                   toty = [toty -max(halfy)];
                    % %
                    % %                   totx = [totx (max(halfx)+xtra)];
                    % %                   toty = [toty -max(halfy)];
                    % %
                    % %                   totx = [totx (max(halfx)+xtra)];
                    % %                   toty = [toty -(min(halfy)-ant_thick/2)];
                    % %
                    % %                     totx = [totx (halfx(1)-xtra)];
                    % %                     toty = [toty -(min(halfy)-ant_thick/2)];
                    % %
                    % %                     totx = [totx (halfx(1)-xtra)];
                    % %                     toty = [toty (min(halfy)-ant_thick/2)];


%                 end
%

            else
                %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
                % will be too close to connect with a line
                if psi_r == 0
                    totx = [fliplr(halfx) halfx];
                    toty = [-fliplr(halfy) halfy];
                else
                    % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
                    % therefore cause a problem.
                    % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
                    % old way #2 -- xtra = max(raex)+1/5*xmax;
                    %xtra = min(raex)+2*xmax+ant_thick;
                    xtra = min(raex)+2*xmax;
                    totx = [fliplr(halfx) halfx  xtra xtra];
                    toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
                end
            end
            %%%%%%%%%%%%%%%%%%%%%%%%%% end create outside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%%
            % % %%%%%%%%%%%%%%%%%%%%%%%%%% code for creating inside antenna first...uses output of normalptsv4.m %%%%%%%%%%%%%%%%%%%%%%%%%%
            % %
            % %             halfx = raex(1:numbsections+1);
            % %             halfy = raey(1:numbsections+1);
            % %             if centerchoice==1 % This choice will center the antenna element within the epoxy
            % %                 % This regime is fundamentally different from noncentering because it should expose
            % %                 % the difference between having a shield of epoxy vs free space and will give insight
            % %                 % into how much a shield is effective at pushing the current to the far-field
            % %
            % %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
            % %                 % will be too close to connect with a line
            % %                 if psi_r == 0
            % %                     totx = [fliplr(halfx) halfx];
            % %                     toty = [-fliplr(halfy) halfy];
            % %                     epoxy = line2(totx,toty);
            % %                 else
            % %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
            % %                     % therefore cause a problem.
            % %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
            % %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
            % %
            % %                     %xtra = min(raex)+2*xmax+ant_thick;
            % %                     widthant = max(halfx)-halfx(1);
            % %                     xtra = (2*xmax-widthant)/2;
            % %                     totx = [halfx (max(halfx)+xtra) (max(halfx)+xtra) fliplr(halfx) (halfx(1)-xtra) (halfx(1)-xtra)];
            % %                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
            % %                     epoxy = line2(totx,toty);
            % %                 end
            % %
            % %
            % %
            % %             else
            % %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
            % %                 % will be too close to connect with a line
            % %                 if psi_r == 0
            % %                     totx = [fliplr(halfx) halfx];
            % %                     toty = [-fliplr(halfy) halfy];
            % %                     epoxy = line2(totx,toty);
            % %                 else
            % %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
            % %                     % therefore cause a problem.
            % %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
            % %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
            % %
            % %                     %xtra = min(raex)+2*xmax+ant_thick;
            % %                     xtra = min(raex)+2*xmax;
            % %                     totx = [fliplr(halfx) halfx  xtra xtra];
            % %                     toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
            % %                     epoxy = line2(totx,toty);
            % %                 end
            % %
            % %             end
            % % %%%%%%%%%%%%%%%%%%%%%%%%%% end create inside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%%
% %                     %%%%%%%% GOOD GRAPHING CODE
% %                     close(gcf)
% %                     subplot(2,2,1)
% %                     scatter(xpoints,ypoints,'r.');
% %                     axis equal
% %                     hold
% %                     scatter(nxpoints,nypoints,'bx');
% %                     axis equal
% %                     title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% % %                     pause
% %
% %                     subplot(2,2,2)
% %                     plot(totx,toty ,'r'),axis equal
% %
% %                     subplot(2,2,3)
% %                     scatter(-raey,-raex,'r.');
% %                     axis equal
% %                     hold
% %                     scatter(raey,-raex,'b.');
% %                     plot(toty,-totx,'r')
% %                     axis equal
% %                     title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %                     pause
% %                     %%%%%%%%%%%%%%%%%%%%%%


            % CALCULATE THE AREA FOR EACH ANTENNA
            % step 1 -- create antenna elements
            aer=line2(raex,raey);
            clear fem
            % Geometry
            fem.geom = aer;
            fem.mesh = meshinit(fem);
            % Integrate on subdomains
            ae_vol = postint(fem,'1');
            %%%%%%%%%%%%BEGIN 2D FEA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
            %calling file rather than inserting code
            commandline_v9
            %%%%%%%%%%%END 2D FEA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


            progress = progress+1;
            waitbar(progress/360)

        end
    end
end

save resultsII.dat results -ascii

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% GRAPHING  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];

for j = 1:length(results)
    if results(j,4)==3
        sheetresults = [sheetresults;results(j,:)];
    elseif results(j,4)==4
        circleresults = [circleresults;results(j,:)];
    elseif results(j,4)==5
        parabolaresults = [parabolaresults;results(j,:)];
    elseif results(j,4)==6
        ellipseresults = [ellipseresults;results(j,:)];
    elseif results(j,4)==7
        hyperbolaresults = [hyperbolaresults;results(j,:)];
    end
end

for j = 4:7
    if j==3
        shaperesults = sheetresults;
    elseif j==4
        shaperesults = circleresults;
    elseif j==5
        shaperesults = parabolaresults;
    elseif j==6
        shaperesults = ellipseresults;
    elseif j==7
        shaperesults = hyperbolaresults;
    end

    figure(1)
    subplot(2,2,j-3)
    if j~=3
    xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
    ylin = linspace(min(shaperesults(:,6)),max(shaperesults(:,6)),90/psi_r_inc+1);

    [Y,X] = meshgrid(ylin,xlin);

    Z = griddata(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),X,Y);
    surf(X,Y,Z);
    if j==3
        title('sheet');
    elseif j==4
        title('circular','FontSize',15);
    elseif j==5
        title('parabolic','FontSize',15);
    elseif j==6
        title('elliptical','FontSize',15);
    elseif j==7
        title('hyperbolic','FontSize',15);
    end
    hold on;
    xlabel('curvature (index)','FontSize',13);
    ylabel('psi (angle)','FontSize',13);
    zlabel('potential difference (V)','FontSize',13);
    set(gca,'FontSize',13)
    %colorbar;
    colormap gray;
    rotate3d
    end
    plot3(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),'o')
    hold off;

end

set(gcf,'Color',[1 1 1])
```

## .14 PDE MULTISHELL (WITH EPOXY)

Contains the following files:

- create_antennas_noReflector.m
- commandline_v9.m (refer to .11)
- shapesfundersq.m (refer to .1)
- shapestotarclength.m (refer to .2)
- shapesptsopt_v2.m (refer to .2)
- normalpointsv2.m (refer to .3)

```
% Brian Wessel
% creates geometry for input to MatLab PDE solver

clear all
close all

% xmax = 0.2;
% ymax = 0.1;
% zmax = 0.9;
% d = 0.9;
% ant_thick = 0.03;

results = [];

progress=0;

%xmax = 2e-3;
ymax = 1e-3;
zmax = 4.5e-3;
d = 9e-3;
ant_thick = 7.5e-4;
% ant_thick = 2.5e-4;
numangles = 10;

epoxyheight=4e-3; % which is two times the old xmax ...reference 1/8/04 why I made this change
xmax = epoxyheight/2-ant_thick;


if xmax > d/2
    error('xmax is greater than d/2 so the antenna elements will criss-cross at large angles!')
end

if ymax > 2*xmax
    error('ymax is greater than 2*xmax, so the epoxy will not cover the side of the antenna when the antenna is turned at a sharp angle (psi)!')
end

% centerchoice = menu('Centering on or off?','on','off');
% shapechoice = menu('Shape?','circle','parabola','ellipse','hyperbola');
% shapechoice = shapechoice+3;
% kchoice = menu('Curvature (k) ?','1','2','3','4','5','6','7','8','9','10');
% psichoice = menu('Angle ?','0','10','20','30','40','50','60','70','80','90');
% psichoice = (psichoice-1)*10;

centerchoice=1;
shapechoice=4;
kchoice=1;
psichoice=0;


% number of monopoles on each antenna and the number of curves to be simulated per shape (except the sheet of course)
numbsections = 20; % MUST BE EVEN b/c linc = 2*totsum/numbsections.  It seems weird but having an even
% number of sections, will put an odd number of points because there is no middle section
% NOTE THAT THE NUBMER OF POINTS WILL BE NUMBSECTIONS+1
desiredcurves = 10; %if k only goes from 1:1 rather than 1:totcurves(ndp) than desiredcurves is overidden and
% therefore, only 1 curve will be created

if numangles==1
    psi_r_inc = 91; % just enough that only zero degrees will be calculated
else
    psi_r_inc = 90/(numangles-1); % increment value for the rotation angle in degrees
end

index = 0; % keeps track of the saving of epoxy and ae's for later use

for ndp = shapechoice:7  % alters shapes
    ndp

    % outputs points on box and the total number of curves that will be used
    [totcurves(ndp) xpt ypt] = shapesptsopt_v2(xmax,ymax,ndp,desiredcurves);

    totcurves(ndp)
    %     display('program paused')
    %     pause

    for k= kchoice:totcurves(ndp)  % calculates a point on the box for a fixed shape (the total number of curves will be numcurves-1)
        k

        a=1;
        b=1;

        if ndp==4
            b = ypt(k)/2*(1+xpt(k)^2/ypt(k)^2); % b is the radius of the sphere at the pts (xpt,ypt) and (0,0)
        elseif ndp==5
            a = ypt(k)/xpt(k)^2;
        elseif ndp==6
            a = xpt(k);
            b = ypt(k);
        elseif ndp==7
            b = d/2;
            a = ((d^2/4*xpt(k)^2)/(ypt(k)^2+ypt(k)*d))^(0.5);
        end

        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % uses the arclength to calculate the placement of the number of dipoles
        % otherwise it uses a much simpler way for the sheet points
        % also an easy way to calculate the total arclength of a circle is used

        if ndp~=3

            % calculates the total arc length in a curve from 0 to xmax then
            % multiplies by 2 to get the total arclength from -xmax to xmax
            % due to symmetry of the curves

            clear xpoints
            clear ypoints
            xpoints = [];


            delxp = 1*10^(-6);

            if ndp==6
                delxp = 1*10^(-6);
            end

            if ndp ~=4

                clear xpoints
                xpoints = [];

                totsum = shapestotarclength(ndp,0,xpt(k),(a)^2,(b)^2);
                linc = 2*totsum/numbsections;

                % calculates the circle arclength increment quickly
            elseif ndp==4

                xy = [xpt(k) ypt(k)];
                rad = [0 b];
                origin = [0 0];
                cosgamma = dot(xy-rad,origin-rad)/(norm(xy-rad)*norm(origin-rad));
                gamma = acos(cosgamma);
                arclength = 2*b*gamma;
                linc = arclength/numbsections;

                for p = 1:numbsections/2
                    xpoints(p) = b*sin((linc*p)/b);
                end
            end
            xp = 0;
            lsum = 0;

            i = 1;
            fpsq = [];

            if ndp ~=4

                % numerically determines the x value for a specified unit of arc length
                while xp < xpt(k) & i < (numbsections-3)

                    while lsum<linc
                        fpsq = shapesfundersq(xp,ndp,a,b);
                        lsum = lsum+delxp*(1+fpsq)^0.5;
                        xp = xp + delxp;
                        if xp > xpt(k)
                            break
                        end
                    end

                    lsum = 0;

                    if xp< xpt(k)
                        xpoints(i) = xp;
                    end

                    i = i+1;
                end
                xpoints = [xpoints, xpt(k)];
            end

        else ndp==3
            linc = xpt(k)*2/numbsections;
            xpoints = linc:linc:xpt(k);
        end

        % creates negative values and adds a zero point for the xpoints
        %nxpoints = -xpoints;
        xpoints = [fliplr(-xpoints) 0  xpoints];


        % end calculation of xpoints using arc length
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




        % calculates ypoints according to which shape the user specifies.
        % all shapes intersect through the origin
        if ndp == 3

            ypoints = zeros(size(xpoints));

        elseif ndp == 4     %circle points

            ypoints = abs((b^2-(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 5 %parabola pts

            ypoints = a*(xpoints.*xpoints);

        elseif ndp == 6  %ellipse points

            ypoints = abs((b^2-b^2/a^2*(xpoints.*xpoints)).^(0.5)-b);

        elseif ndp == 7   %hyperbola points

            ypoints = (b^2+b^2/a^2*(xpoints.*xpoints)).^(0.5)-b;

        end

        %                 subplot(2,2,ndp-3)
        %                 scatter(xpoints,ypoints,'.')
        %                 axis([(-xmax-xmax/5) (xmax+xmax/5) -ymax/10 (ymax+ymax/5)]);
        %                 axis equal
        %                 if ndp==4
        %                     title('semicircular')
        %                 elseif ndp==5
        %                         title('parabolic')
        %                     elseif ndp==6
        %                             title('elliptical')
        %                         else
        %                             title('hyperbolic')
        %                         end
        %                     end
        %                 end
        %
        %                 pause

        %title(['shape',num2str(ndp),'   ','curve number',num2str(k)])

        [nxpoints,nypoints] = normalpointsv2(ndp,a,b,ant_thick,xpoints,ypoints);

        % %         close(gcf)
        % %         scatter(xpoints,ypoints,'r.');
        % %         axis equal
        % %         hold
        % %         scatter(nxpoints,nypoints,'bx');
        % %         axis equal
        % %
        % %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
        % %         pause


        %%%%%%%%%%%%%%%%%
        %%%%%%%%%%%%%%%%% MAKING ANTENNA ELEMENT USING FEMLab FUNCTIONS
        % Note that the I am keeping the enclosing box x and y axes, so it is switched from normal

        for psi_r = psichoice:psi_r_inc:90   % rotates psi_r for a fixed shape and curvature

            index = index+1;

            psi_r


            %             aex=[fliplr(nxpoints),xpoints];
            %             aey=[fliplr(nypoints),ypoints];

            aex=[xpoints,fliplr(nxpoints)];
            aey=[ypoints,fliplr(nypoints)];

            % rotating neg. ae
            phi = psi_r*pi/180;

            rot_mat = [cos(phi) -sin(phi); sin(phi) cos(phi)];
            ae = [aex;aey];
            rae = rot_mat*ae;
            raex = rae(1,:);
            raey = rae(2,:)+d/2;


            %            %%%%%%%%%%%% Plot which checks for correct rotation
            %             plot(aex,aey,'b',raex,raey,'r')
            %             axis equal
            %             title('check for correct rotation');
            %             pause
```

```
%%%%%%%%%%%%%%
%             % step 1 -- create antenna elements
%             aer=line2(raex,raey);
%             ael=line2(raex,-raey);

% %             %%%%%%%%%% Plotting
% %                         close(gcf)
% %                         scatter(raex,raey,'r.');
% %                         axis equal
% %                         hold
% %                         scatter(raex,-raey,'b.');
% %                         axis equal
% %                         %pause
% %                         geomplot(aer)
% %                         axis equal
% %                         hold
% %                         geomplot(ael)
% %                         axis equal
% %                        % pause
% %             %%%%%%%%%%%%%%%%%%%%

%step2 -- create epoxy
%close(gcf)


%%%%%%%%%%%%%%%%%%%%%%%% code for creating outside antenna first...uses output of normalptsv2.m %%%%%%%%%%%%%%%%%%%%%%%%%%

halfx = raex(numbsections+2:length(raex));
halfy = raey(numbsections+2:length(raey));
if centerchoice==1 % This choice will center the antenna element within the epoxy
    % This regime is fundamentally different from noncentering because it should expose
    % the difference between having a shield of epoxy vs free space and will give insight
    % into how much a shield is effective at pushing the current to the far-field

    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line...however, I should add about half the antenna thickness
    % to each side so that it doesn't make a sharp point near the tip of the antenna as in the case of the
    % elliptically shaped antenna elements
    if psi_r == 0
        xtra = ant_thick/2;
        %                    xtra = ant_thick*4/5;
        totx = [halfx (halfx(length(halfx))-xtra) (halfx(length(halfx))-xtra) fliplr(halfx) (halfx(1)+xtra) (halfx(1)+xtra) ];
        toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
        %                    epoxy = line2(totx,toty);
    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        halfx = fliplr(halfx);
        halfy = fliplr(halfy);
        widthant = max(halfx)-halfx(1);
        %                    xout = ant_thick*cos(phi);
        xout = ant_thick*cos(phi)*3/5;
        %                    xtra = (epoxyheight-widthant)/2;
        xtra = (epoxyheight-widthant)*4/5;
        %                    yout = ant_thick*sin(phi);
        yout = ant_thick*sin(phi)*3/5;

        % %                     totx = [halfx max(halfx) (max(halfx)+xtra) (max(halfx)+xtra) max(halfx) fliplr(halfx) halfx(1) (halfx(1)-xtra) (halfx(1)-xtra) halfx(1)];
        % %
        % %                    toty = [halfy (halfy(length(halfy))+ant_thick/2) (halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -(halfy(length(halfy))+ant_thick/2) -fli
        % %

        %%%Sophisticated Modeling which is memory intensive
        totx = halfx;
        toty = halfy;

        %                   totx = [totx (max(halfx)+xout)];
        totx = [totx (halfx(length(halfx))+xout)];
        toty = [toty (halfy(length(halfy))+yout)];

        totx = [totx (max(halfx)+xout+xtra)];
        toty = [toty (halfy(length(halfy))+yout)];

        totx = [totx (max(halfx)+xout+xtra)];
        toty = [toty -(halfy(length(halfy))+yout)];

        %                   totx = [totx (max(halfx)+xout)];
        totx = [totx (halfx(length(halfx))+xout)];
        toty = [toty -(halfy(length(halfy))+yout)];

        totx = [totx fliplr(halfx)];
        toty = [toty -fliplr(halfy)];

        totx = [totx (halfx(1)-xout)];
        toty = [toty -(halfy(1)-yout)];

        totx = [totx (halfx(1)-xtra-xout)];
        toty = [toty -(halfy(1)-yout)];

        totx = [totx (halfx(1)-xtra-xout)];
        toty = [toty (halfy(1)-yout)];

        totx = [totx (halfx(1)-xout)];
        toty = [toty (halfy(1)-yout)];


        % % %                  Primitive Modeling
        % %
        % %                    totx = [(max(halfx)+xtra)];
        % %                    toty = [min(halfy)-ant_thick/2];
        % %
        % %                    totx = [totx (max(halfx)+xtra)];
        % %                    toty = [toty max(halfy)];
        % %
        % %                    totx = [totx (max(halfx)+xtra+ant_thick)];
        % %                    toty = [toty max(halfy)];
        % %
        % %                    totx = [totx (max(halfx)+xtra+ant_thick)];
        % %                    toty = [toty -max(halfy)];
        % %
        % %                    totx = [totx (max(halfx)+xtra)];
        % %                    toty = [toty -max(halfy)];
        % %
        % %                    totx = [totx (max(halfx)+xtra)];
        % %                    toty = [toty -(min(halfy)-ant_thick/2)];
        % %
        % %
        % %                    totx = [totx (halfx(1)-xtra)];
        % %                    toty = [toty -(min(halfy)-ant_thick/2)];
        % %
        % %
        % %                    totx = [totx (halfx(1)-xtra)];
        % %                    toty = [toty (min(halfy)-ant_thick/2)];


    end
```

```
else

    %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
    % will be too close to connect with a line
    if psi_r == 0
        totx = [fliplr(halfx) halfx];
        toty = [-fliplr(halfy) halfy];

    else
        % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
        % therefore cause a problem.
        % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
        % old way #2 -- xtra = max(raex)+1/5*xmax;

        %xtra = min(raex)+2*xmax+ant_thick;
        xtra = min(raex)+2*xmax;
        totx = [fliplr(halfx) halfx  xtra xtra];
        toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];

    end

end
%%%%%%%%%%%%%%%%%%%%%%%% end create outside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

% % %%%%%%%%%%%%%%%%%%%%%%%% code for creating inside antenna first...uses output of normalptsv4.m %%%%%%%%%%%%%%%%%%%%%%%%%
% %
% %             halfx = raex(1:numbsections+1);
% %             halfy = raey(1:numbsections+1);
% %             if centerchoice==1 % This choice will center the antenna element within the epoxy
% %                 % This regime is fundamentally different from noncentering because it should expose
% %                 % the difference between having a shield of epoxy vs free space and will give insight
% %                 % into how much a shield is effective at pushing the current to the far-field
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     widthant = max(halfx)-halfx(1);
% %                     xtra = (2*xmax-widthant)/2;
% %                     totx = [halfx (max(halfx)+xtra) (max(halfx)+xtra) fliplr(halfx) (halfx(1)-xtra) (halfx(1)-xtra)];
% %                     toty = [halfy halfy(length(halfy)) -halfy(length(halfy)) -fliplr(halfy) -halfy(1) halfy(1)];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %
% %
% %             else
% %
% %                 %if psi_r = 0, we don't want to add extra epoxy or it will cause an error because the points
% %                 % will be too close to connect with a line
% %                 if psi_r == 0
% %                     totx = [fliplr(halfx) halfx];
% %                     toty = [-fliplr(halfy) halfy];
% %                     epoxy = line2(totx,toty);
% %                 else
% %                     % (for old way #1)this strange sin*blah is so that at large angles the epoxy does not overlap itself and
% %                     % therefore cause a problem.
% %                     % old way #1 -- xtra = halfx(1,1)+sin(psi_r*pi/180)*(ymax+ant_thick);
% %                     % old way #2 -- xtra = max(raex)+1/5*xmax;
% %
% %                     %xtra = min(raex)+2*xmax+ant_thick;
% %                     xtra = min(raex)+2*xmax;
% %                     totx = [fliplr(halfx) halfx  xtra xtra];
% %                     toty = [fliplr(halfy) -halfy -halfy(1,length(halfy)) halfy(1,length(halfy))];
% %                     epoxy = line2(totx,toty);
% %                 end
% %
% %             end
% % %%%%%%%%%%%%%%%%%%%%%%%% end create inside antenna first %%%%%%%%%%%%%%%%%%%%%%%%%

% %         %%%%%%%% GOOD GRAPHING CODE
% %         close(gcf)
% %         subplot(2,2,1)
% %         scatter(xpoints,ypoints,'r.');
% %         axis equal
% %         hold
% %         scatter(nxpoints,nypoints,'bx');
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %
% %         subplot(2,2,2)
% %         plot(totx,toty ,'r'),axis equal
% %
% %         subplot(2,2,3)
% %         scatter(-raey,-raex,'r.');
% %         axis equal
% %         hold
% %         scatter(raey,-raex,'b.');
% %         plot(toty,-totx,'r')
% %         axis equal
% %         title(['shape',num2str(ndp),'   ','curve number',num2str(k)])
% %         pause
% %         %%%%%%%%%%%%%%%%%%%%%%

            % CALCULATE THE AREA FOR EACH ANTENNA
% step 1 -- create antenna elements
aer=line2(raex,raey);
clear fem
% Geometry
fem.geom = aer;
fem.mesh = meshinit(fem);
% Integrate on subdomains
ae_vol = postint(fem,'1');



%%%%%%%%%%%BEGIN 2D FEA%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%calling file rather than inserting code

commandline_v9_Multishell



%%%%%%%%%%END 2D FEA %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


progress = progress+1;
```

```
            waitbar(progress/360)


        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%  GRAPHING  %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
break
save resultsMultishell_II.dat results -ascii

sheetresults = [];
circleresults = [];
parabolaresults = [];
ellipseresults = [];
hyperbolaresults = [];

for j = 1:length(results)
    if results(j,4)==3
        sheetresults = [sheetresults;results(j,:)];
    elseif results(j,4)==4
        circleresults = [circleresults;results(j,:)];
    elseif results(j,4)==5
        parabolaresults = [parabolaresults;results(j,:)];
    elseif results(j,4)==6
        ellipseresults = [ellipseresults;results(j,:)];
    elseif results(j,4)==7
        hyperbolaresults = [hyperbolaresults;results(j,:)];
    end
end

for j = 4:7
    if j==3
        shaperesults = sheetresults;
    elseif j==4
        shaperesults = circleresults;
    elseif j==5
        shaperesults = parabolaresults;
    elseif j==6
        shaperesults = ellipseresults;
    elseif j==7
        shaperesults = hyperbolaresults;
    end

    figure(1)
    subplot(2,2,j-3)
    if j~=3
    xlin = linspace(min(shaperesults(:,5)),max(shaperesults(:,5)),totcurves(j));
    ylin = linspace(min(shaperesults(:,6)),max(shaperesults(:,6)),90/psi_r_inc+1);

    [Y,X] = meshgrid(ylin,xlin);

    Z = griddata(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),X,Y);
    surf(X,Y,Z);
    if j==3
        title('sheet');
    elseif j==4
        title('circular');
    elseif j==5
        title('parabolic');
    elseif j==6
        title('elliptical');
    elseif j==7
        title('hyperbolic');
    end
    hold on;
    xlabel('curvature (index)');
    ylabel('psi (angle)');
    zlabel('potential difference (mV)');
    %colorbar;
    colormap gray;
    rotate3d
    end
    plot3(shaperesults(:,5),shaperesults(:,6),shaperesults(:,1),'o')
    hold off;

end

set(gcf,'Color',[1 1 1])
```

```
% commandline_v9_MultiShell.m
% Brian Wessel
% Feb. 26, 2004


% clear all
% close all
%
% load raex.dat
% load raey.dat
% load totx.dat
% load toty.dat
%
% %%%%%% creates shield
% totx(45) = 1.9e-3;
% totx(46) = 1.9e-3;
% totx(47) = 4e-3;
% totx(48) = 4e-3;
% totx(49) = 1.9e-3;
% totx(50) = 1.9e-3;
%
% toty(46) = -4e-3;
% toty(46) = -8e-3;
% toty(47) = -8e-3;
% toty(48) = 8e-3;
% toty(49) = 8e-3;
% toty(50) = 4e-3;
% %%%%% end create shield
%
% scatter(raey,-raex,'b.');
% hold
% plot(toty,-totx,'r')
% axis equal
% scatter(-raey,-raex,'r.');
% axis equal
% pause
% close(gcf)


%%%% Constants
% conductivities
s_br  = 1/5.8;
s_sk = 1/177;
s_ski = 100/222;
s_e  = 1e-7;
s_ae = 1e13;
s_csf = 1/0.7;

% % radii (actual)
% skin  = 7.5000e-2; %
% skull = 7.1467e-2; %
% csf   = 6.9705e-2; %
% brain = 6.6180e-2; %
% z_ant = 6.7942e-2;

% radii (brain is smaller so the antenna will fit in between)
skin  = 7.5000e-2; %
skull = 7.1467e-2; %
csf   = 6.9705e-2; %
brain = 6.1180e-2; %
z_ant = 6.5942e-2;


% amount of current/area (Therefore, I will need to know the area
% so that I can apply the correct amount of current.)
curr = 1e-3;

% creating the positive ant. element
% x = [3.3e-2 3.5e-2 3.5e-2 3.3e-2];
% y = [-1e-2 -1e-2 1e-2 1e-2];



% adding 5.5e-2 for now just to make sure that I am simulating correctly
pae = [2 length(raex) raey (-raex+z_ant) zeros(1,2*(length(totx)-length(raey)))]';

% creating the negative ant. element
nae = [2 length(raex) -raey (-raex+z_ant) zeros(1,2*(length(totx)-length(raey)))]';

% creating epoxy
% epoxy = [2 4 -3.3e-2 3.3e-2 3.3e-2 -3.3e-2 -1e-2 -1e-2 1e-2 1e-2]';
epoxy = [2 length(totx) toty (-totx+z_ant)]';


br  = [1 0 0 brain zeros(1,length(epoxy)-4)]';
cs  = [1 0 0 csf zeros(1,length(epoxy)-4)]';
sk  = [1 0 0 skull zeros(1,length(epoxy)-4)]';
ski = [1 0 0 skin zeros(1,length(epoxy)-4)]';

% Testing code
% pdecirc(0, 0 ,7.5e-2)
% pdepoly([3.5e-2, 3.3e-2, 3.3e-2, 3.5e-2],[-1e-2 -1e-2 1e-2 1e-2])
% pdepoly([-3.5e-2, -3.3e-2, -3.3e-2, -3.5e-2],[-1e-2 -1e-2 1e-2 1e-2])

csg = [nae pae epoxy br cs sk ski];


% dgm = decsg(csg,['circle+pae+nae+epoxy'],[['nae   ']' ['pae   ']' ['epoxy ']' ['circle']'])
dgm = decsg(csg);

% wgeom(dgm,'geomfile')
% break

% % % subplot(1,2,1)

% % pdegplot(dgm)
% % axis equal
% % display('program pause')
% % pause


[p,e,t]=initmesh(dgm,'jiggle','off','hgrad',1.25);
% q=pdetriq(p,t);
% subplot(1,2,1)
% pdeplot(p,e,t,'xydata',q,'colorbar','on','xystyle','flat')
% axis equal
%
% pause
% subplot(1,2,2)
% % % p1=jigglemesh(p,e,t,'opt','minimum','iter',inf);
% % % q=pdetriq(p1,t);
% % % figure(1)
% % % % pdeplot(p1,e,t,'xydata',q,'colorbar','on','xystyle','flat')
% % % pdeplot(p1,e,t)
% % % axis equal


% pause

% % % p=p1;
%

% br | pae | epoxy | nae | ski | sk | cs
% c = ['1!1e2!1e-10!1e2!1!1!1'];
% a = ['0!0!0!0!0!0!0'];
```

```
%f = ['0!(1e-3/2.0904e-006)!0!(-1e-3/2.0904e-006)!0!0!0'];

% br | pae | epoxy | nae | ski | sk | cs
c = [num2str(s_br),'!',num2str(s_ae),'!',num2str(s_e),'!',num2str(s_ae),'!',num2str(s_ski),'!',num2str(s_sk),'!',num2str(s_csf)];
a = ['0!0!0!0!0!0!0'];
f = ['0!',num2str(curr/ae_vol),'!0!(-',num2str(curr/ae_vol),')!0!0!0'];

% Note only the last columns are the exterior boundaries
% The other internal ones must have contrived data
cols = size(dgm,2);
b = double([0 1 1 1 1 1 '0' '0' '1' '0']');
for i=1:(cols-5)
    % This is the contrived data I found from the output of the PDETool
    b = [b double([0 1 1 1 1 1 '0' '0' '1' '0']')];
end
b(:,cols-3) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols-2) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols-1) = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
b(:,cols)   = double([1 0 1 1 '0' '0' '0' '0' '1' '0']');
u = assempde(b,p,e,t,c,a,f);
% demean signal (artificial)
u = u-mean(u);
% [u,p,e,t]=adaptmesh(dgm,b,c,a,f,'Ngen',2);
[cgxu,cgyu]=pdecgrad(p,t,c,u);
% [ux,uy] = pdegrad(p,t,u);
% uu = [ux',uy'];
uu = [cgxu',cgyu'];
% demean signal (artificial)
uu(:,1) = uu(:,1)-mean(uu(:,1));
uu(:,2) = uu(:,2)-mean(uu(:,2));
% % figure(2)
% % pdeplot(p,e,t,'xydata',u,'mesh','off','contour','on','levels',60);
% % title(['Voltage Conductivities:','brain=',num2str(s_br),' ant ele''s=',num2str(s_ae),' epoxy=',num2str(s_e),' csf=',num2str(s_csf),' skull=',num2str(s_sk),' skin=',num2str(s_ski)]);
% % set(gcf,'Color',[1 1 1])
% % axis equal
% %
% % display('program pause')
% % pause
% %
% % figure(3)
% % pdeplot(p,e,t,'xydata',uu,'mesh','off','contour','on','levels',200);
% % title('Current Density');
% % axis equal

% i=pdesde(e,5);

% finding boundary points and values
m=1;
for j = 1:length(p)
    bpts = norm(p(:,j));
%     display('program paused')
%     pause
    if (0.075-bpts)<1e-3
%         (0.075-bpts)
%         display('true')
%         pause
        boundv(m) = u(j);
        boundp(1,m) = p(1,j);
        boundp(2,m) = p(2,j);
%             display('program paused - inside loop')
%             pause
        x=p(1,j);
        y=p(2,j);
        theta(m) = atan(y/x);
        % fixing atan function so it goes from 0 to 2*pi rather than -pi/2 to pi/2
        if x<0 & y>0
            theta(m) = theta(m)+pi;
        elseif x<0 & y<0
            theta(m) = theta(m)-pi;
        end
        m = m+1;
    end

end
% % %%%%%%%%%%%%%%%%%% GRAPHING
% % figure(2)
% % subplot(2,2,1)
% % plot(abs(boundv))
% % title('boundary voltage')
% %
% % subplot(2,2,2)
% % % polar(theta,boundv+1.1*min(boundv),'r.')
% % polar(theta,abs(boundv),'r.')
% % title('boundary voltage')
% % % % subplot(2,2,3)
% % stem3(boundp(1,:),boundp(2,:),boundv,'ro')
% % title('boundary voltage')
% %
% % figure(3)
% % polar(theta,abs(boundv),'r.')
% % title('boundary voltage')
% % %%%%%%%%%%%%%%%%%%
% display('program pause')
% pause
results = [results; (max(boundv)-min(boundv)) max(boundv) min(boundv) ndp k psi_r];

clear p p1 q t u uu cgyu cgxu
pack
% display('End of 2D FEA iteration - PROGRAM PAUSED')
% pause
% subplot(2,2,3)
% pdecont(p,t,u,20);
% axis equal
```

## .15 PLOTTING - EPOXY WITH % INCREASE FROM ANALYTICAL

Contains the following file:

- EpoxyResults.m

```
% EpoxyResults.m
% Brian Wessel
% plotting results

load resultsX.dat

%circle points

curve = [1 3 5 6 1 3 5 6 1 3 5 6];

angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==4 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [15.9951 15.8749 16.6184 16.4645 15.291 16.2845 17.7529 18.041 17.2755 17.5412 17.9634 18.0391];

c_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 3 5 6];
ylin = [0 40 90 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,1)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('potential difference (mV)','FontSize',13);
title('CIRCULAR','FontSize',15)
axis([0 10 0 90 28 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;

pause
%parabola points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];

angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==5 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [14.8218 16.1019 16.3218 16.059 16.0707 17.5165 18.5119 19.5355 16.8201 17.5545 17.9899 18.6729];

p_per = (maxv-ana_maxv)./ana_maxv*100;



xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,2)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('potential difference (mV)','FontSize',13);
title('PARABOLIC','FontSize',15)
axis([0 10 0 90 28 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;

pause
%ellipse points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];

angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==6 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [15.867 16.5356 16.2539 15.67 16.3824 17.2376 18.3474 19.3977 16.4966 17.5417 18.0732 18.6771];

e_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,3)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('potential difference (mV)','FontSize',13);
title('ELLIPTICAL','FontSize',15)
axis([0 10 0 90 28 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;


pause
```

```
%hyperbola points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];

angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==7 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = [14.9403 16.2585 16.2866 15.7563 16.0993 17.4512 18.5534 19.387 16.9944 17.6567 18.3735 18.0686];

h_per = (maxv-ana_maxv)./ana_maxv*100;


xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,4)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('potential difference (mV)','FontSize',13);
title('HYPERBOLIC','FontSize',15)
axis([0 10 0 90 28 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;
set(gcf,'Color',[1 1 1])

pause

%%%%%%%%%%%%%%%%%%%%%% PERCENT INCREASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)

curve = [1 3 5 6 1 3 5 6 1 3 5 6];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
xlin = [1 3 5 6];
ylin = [0 40 90 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,c_per,X,Y);
subplot(2,2,1)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('Percent Increase (%)','FontSize',13);
title('CIRCULAR','FontSize',15)
axis([0 10 0 90 10 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,c_per)

hold off;

pause

curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,p_per,X,Y);
subplot(2,2,2)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('Percent Increase (%)','FontSize',13);
title('PARABOLIC','FontSize',15)
axis([0 10 0 90 10 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,p_per)

hold off;


pause

Z = griddata(curve,angle,e_per,X,Y);
subplot(2,2,3)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('Percent Increase (%)','FontSize',13);
title('ELLIPTICAL','FontSize',15)
axis([0 10 0 90 10 40])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,e_per)

hold off;

pause


subplot(2,2,4)
Z = griddata(curve,angle,h_per,X,Y);
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',13);
ylabel('psi (angle)','FontSize',13);
zlabel('Percent Increase (%)','FontSize',13);
title('HYPERBOLIC','FontSize',15)
axis([0 10 0 90 10 40])
%colorbar;
```

```
colormap gray;
rotate3d

scatter3(curve,angle,h_per)

hold off;
set(gcf,'Color',[1 1 1])
```

## .16 PLOTTING - NO EPOXY (FE) WITH % INCREASE FROM ANALYTICAL

Contains the following file:

- NoEpoxyResults.m

```
% NoEpoxyResults.m
% Brian Wessel
% February 25, 2004
% Results from 3D FE simulation using FEMLab
% with no epoxy between the elements.  By no
% epoxy I mean that I set the conductivity
% of the epoxy equal to that of the sphere.

clear all
close all

load resultsX.dat


circle = [11.9553 11.0397 11.3654 12.0697...
        11.6078 11.677 12.7763 12.5281 12.2731];

parabola = [11.5227 11.0792 11.33 12.3377 12.0991 11.9801...
        12.7381 12.6129 12.333 12.8672 13.1119 12.7074];

ellipse = [11.8072 11.1914 11.0606 12.6002 12.0718 11.9506...
        12.7601 12.6485 12.3614 12.8284 13.1412 12.7101];

hyperbola = [11.5846 11.1361 11.346 12.3808 12.0797 11.9602...
        12.7433 12.6861 12.438 12.8789 13.078 12.6006];


curve = [1 4 6 1 4 6 1 4 6];


angle = [0 0 0 40 40 40 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==4 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

%maxv = [15.9951 15.8749 16.6184 16.4645 15.291 16.2845 17.7529 18.041 17.2755 17.5412 17.9634 18.0391];
maxv = circle;

c_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 4 6];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,1)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('potential difference (mV)','FontSize',14);
title('CIRCULAR','FontSize',15)
axis([0 10 0 90 22 27])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;


%parabola points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];

% maxv = [14.8218 16.1019 16.3218 16.059 16.0707 17.5165 18.5119 19.5355 16.8201 17.5545 17.9899 18.6729];
for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==5 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = parabola;
p_per = (maxv-ana_maxv)./ana_maxv*100;


xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,2)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('potential difference (mV)','FontSize',14);
title('PARABOLIC','FontSize',15)
axis([0 10 0 90 22 27])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;

%ellipse points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==6 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end
% maxv = [15.867 16.5356 16.2539 15.67 16.3824 17.2376 18.3474 19.3977 16.4966 17.5417 18.0732 18.6771];
maxv = ellipse;

e_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,3)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('potential difference (mV)','FontSize',14);
title('ELLIPTICAL','FontSize',15)
axis([0 10 0 90 22 27])
```

```
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;


%hyperbola points

curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];

for i = 1:length(curve)
    for j = 1:length(resultsX(:,1))
        if(resultsX(j,3)==7 & resultsX(j,4)==angle(i) & resultsX(j,5)==curve(i))
            ana_maxv(i) = resultsX(j,1);
        end
    end
end

maxv = hyperbola;
% maxv = [14.9403 16.2585 16.2866 15.7563 16.0993 17.4512 18.5534 19.387 16.9944 17.6567 18.3735 18.0686];

h_per = (maxv-ana_maxv)./ana_maxv*100;

xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,2*maxv,X,Y);
subplot(2,2,4)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('potential difference (mV)','FontSize',14);
title('HYPERBOLIC','FontSize',15)
axis([0 10 0 90 22 27])
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,2*maxv)

hold off;

set(gcf,'Color',[1 1 1])

%%%%%%%%%%%%%%%%%%%%% PERCENT INCREASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

figure(2)


curve = [1 4 6 1 4 6 1 4 6];
angle = [0 0 0 40 40 40 90 90 90];
xlin = [1 4 6];
ylin = [0 40 90];
[Y,X] = meshgrid(ylin,xlin);
Z = griddata(curve,angle,c_per,X,Y);
subplot(2,2,1)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('Percent Increase (%)','FontSize',14);
title(['CIRCULAR    AVG= ',num2str(mean(mean(c_per))),'%'],'FontSize',15)
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,c_per)
axis([0 10 0 100 -100 0])
hold off;


curve = [1 4 7 10 1 4 7 10 1 4 7 10];
angle = [0 0 0 0 40 40 40 40 90 90 90 90];
xlin = [1 4 7 10];
ylin = [0 40 90];

[Y,X] = meshgrid(ylin,xlin);

Z = griddata(curve,angle,p_per,X,Y);
subplot(2,2,2)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('Percent Increase (%)','FontSize',14);
title(['PARABOLIC    AVG= ',num2str(mean(mean(p_per))),'%'],'FontSize',15)
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,p_per)
axis([0 10 0 100 -100 0])
hold off;


Z = griddata(curve,angle,e_per,X,Y);
subplot(2,2,3)
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('Percent Increase (%)','FontSize',14);
title(['ELLIPTICAL    AVG= ',num2str(mean(mean(e_per))),'%'],'FontSize',15)
%colorbar;
colormap gray;
rotate3d

scatter3(curve,angle,e_per)
axis([0 10 0 100 -100 0])
hold off;



subplot(2,2,4)
Z = griddata(curve,angle,h_per,X,Y);
surf(X,Y,Z);

hold on;
xlabel('curvature (index)','FontSize',14);
ylabel('psi (angle)','FontSize',14);
zlabel('Percent Increase (%)','FontSize',14);
title(['HYPERBOLIC    AVG= ',num2str(mean(mean(h_per))),'%'],'FontSize',15)
%colorbar;
colormap gray;
rotate3d
```

```
set(gca,'FontSize',12)
scatter3(curve,angle,h_per)
axis([0 10 0 100 -100 0])
hold off;

set(gcf,'Color',[1 1 1])
```

## .17 CONDUCTIVITY CONVERGENCE

Contains the following file:

- CondConvergence.m

```
% CondConvergence.m
% Brian Wessel
% plotting results of experiment

%first plot with a spread around sigma_sphere
spread = [1 5 10 50 100 500 1000 5000 10000 50000 1e5 5e5 1e6 5e6 1e7 5e7 1e8 5e8]
maxV = [13.6778 15.3221 15.7715 16.3031 16.3929 16.4507 ...
        16.4306 16.3742 16.3320 16.2570 16.2255 16.1354 ...
        16.1207 16.0613 16.1184 16.1329 15.7711 14.8409]

figure(1)
plot(log10(spread),maxV,'r.')
title('Max Voltage vs Log10(spread)')
xlabel('Log10(spread)')
ylabel('Max Voltage')
axis equal

[log10(spread)',maxV']

%second plot with a constant sigma_ae but decreasing sigma_ins

maxV = [13.4051 15.7409 16.0970 16.4119 ...
        16.4556 16.4875 16.4917 16.4948 ...
        16.4952 16.4955 16.4955]
sig_sphere = 100/222;
sig_ae = sig_sphere*50;
sig_ins = sig_sphere./[1 5 10 50 100 500 1000 5000 1e4 5e4 1e5];

spread = sig_ae./sig_ins;

figure(2)
plot(log10(spread),maxV,'ro','MarkerSize',10)
title('Max Voltage vs Log_{10}(\sigma_{epoxy}/\sigma_{epoxy})','FontSize',20)
xlabel('Log_{10}(\sigma_{metal}/\sigma_{epoxy})','FontSize',20)
ylabel('Max Voltage','FontSize',20)
set(gca,'FontSize',15)
axis equal
%gtext('\sigma_{ae} = \sigma_{sphere}*50')
```

# BIBLIOGRAPHY

[1] Halliday D, Resnick R, and K Krane. *Physics.* Vol. 2, 4th ed., John Wiley & Sons, Inc., NY, 1992.

[2] Hauser WA, Hesdorffer DC. *Epilepsy: Frequency, Causes and Consequences.* New York: Demos Press, 1990.

[3] Engel J Jr, Shewmon DA. Overview: who should be considered a surgical candidate? In: Engel J Jr, ed. *Surgical Treatment of the Epilepsies* 2nd ed. New York: Raven Press, 1993:23-24.

[4] Engel J Jr. A Greater Role for Surgical Treatment of Epilepsy: Why and When?. *Epilepsy Currents.* Vol. 3, No. 2 (March/April) 2003:37-40.

[5] Sun M, Mickle M, Liang W, Liu Q, Sclabassi RJ. Data Communication between Brain Implants and Computer. *IEEE Transactions on Neural Systems and Rehabilitation Engineering.* Vol. 11, No. 2, June 2003.

[6] Sun M, Liu Q, Liang W, Wessel BL, Roche P, Mickle M,Sclabassi RJ. *Application of the Reciprocity Theorem to Volume Conduction Based Data Communication Systems between Implantable Devices and Computers.* IEEE Transactions on Neural Systems and Rehabilitation Engineering, Vol. 11, No. 2, June 2003.

[7] Malmivuo J, Plonsey R. *Bioelectromagnetism.* Oxford University Press, New York, 1995.

[8] Rush S, Driscoll DA. EEG-electrode sensitivity–An application of reciprocity. *IEEE Trans. Biomed. Eng..* BME-16:(1) 15-22.

[9] Barber DC, Brown BH. Applied potential tomography. *J. Phys. E.: Sci. Instrum.* Vol. 17: 723-733.

[10] Plonsey R, Heppner DB. Considerations of quasi-stationarity in electrophsyiological systems. *Bulletin of Mathematical Biophysics* Vol. 29, 1967, 657-664.

[11] Webster, JG ed. *Medical Instrumentation: Application and Design.* 3rd ed.,John Wiley & Sons, Inc. New York, 1998 .

[12] R. D. Sidman, V. Giambalvo, T. Allison and P. Bergey . "A method of localization of sources of human cerebral potentials evoked by sensory stimuli" *Sensory Processes* Vol. 2, 1978, pp. 116-129.

[13] Frank, E. "Electric potential produced by two point current sources in a homogeneous conducting sphere," *J. Appl. Phys.* 23: 1225-1228 (1952).

[14] *FEMLab Reference Manual.*COMSOL, Nov, 2001, pp 3-124,5-287.

[15] Sun M."An Efficient algorithm for computing multishell spherical volume conductor models in EEG dipole source localization," *IEEE Transactions on Biomedical Engineering.* Vol. 44, No. 12, December, 1997.

[16] Chari MVK, Salon SJ. *Numerical Methods in Electromagnetism.* Academic Press, San Diego, 2001.

[17] Gulrajani RM. *Bioelectricity and Biomagnetism.* John Wiley & Sons, Inc. New York, 1998.

[18] http://www.mathworks.com/access/helpdesk/help/toolbox/pde/4fem2.shtml.

[19] Hunter, P and A Pullan. *FEM/BEM Notes.* The University of Auckland, New Zealand. ©1997-2003.

[20] Ahonen, AI, Hamalainen MS, Ilmoniemi RJ, Jajola MJ, Knuutila JET, Simola JT, Vilkman VA. ."Sampling Theory for Neuromagnetic Detector Arrays." *IEEE Transactions on Biomedical Engineering* . Vol. 40, No. 9, September, 1993.

[21] Lindsey DP, McKee EL, Hull ML, Howell SM. ."A New Technique for Transmission of Signals from Implantable Transducers." *IEEE Transactions on Biomedical Engineering* . Vol. 45, No. 5, May, 1998.

[22] Cuffin BN and D Cohen. ."Comparison of the magnetoencephalogram and electroencephalogram." *Electroencephalography and Clinical Neurophysiology* . vol. 47, pp 132-146, 1979.

[23] Lindsey DP, McKee EL, Hull ML, Howell SM. ."A New Technique for Transmission of Signals from Implantable Transducers." *IEEE Transactions on Biomedical Engineering* . Vol. 45, No. 5, May, 1998.