IISER BERHAMPUR

NUMERICAL PROJECT REPORT

Simulation of Absorption of EM Wave of Different Frequencies in Human Tissues using FDTD Method

---

Submitted by :
Arindam Bhunia[1] (17021)
Rakesh Naik[1] (17076)
Rishav Sagar[1] (17080)
Sachidananda Sahu[1] (17084)
Shubham Barua[1] (17103)

Submitted to:
Prof. Venugopal Achanta
Dept. of Condensed Matter Physics and Materials Science
Tata Institute of Fundamental Research

[1]-4th Year BS-MS Student
(Physics Major)
IISER Berhampur

Contents

Abstract

For biological materials, dielectric properties highly depend on given environment such as frequency and temperature, the dielectric properties of biological materials are considered to be frequency variations of the relative permittivity and conductivity. In this project, data from 7 human tissues at the frequency ranges from 100 MHz to 800 MHz is taken into consideration. Tackling such problems requires the use of numerical techniques to solve Maxwell equations under the appropriate boundary conditions. Among the different numerical techniques, the FDTD is extremely useful in calculating the induced currents and fields in the human body exposed to electromagnetic radiations. Different formulations have been used to represent dispersive media for numerical analysis, such as Debye, Lorentz or Cole-Cole models. The commonly used FDTD formulation is usually modified using different approaches to accommodate the dispersive characteristics. In this project the main motivation is to simulate a pulse of different frequencies striking the tissues using 1-D FDTD simulation using the simple lossy dielectric model.In this project we have shown the result for tissues: **Skin, Fat, Muscle, Cartilage, Lung and Brain**

1. Introduction

FDTD is an algorithm suitable for use with a computer and can be used as a real-time solver of electromagnetic problems.
It is necessary that one define some basic elements in an analytical electromagnetic problem; in the FDTD method one should do the same, too. These elements are: 1) the basic equations which are Maxwell's equations, 2) spatial and temporal grids, 3) constitutive parameters which include permittivity, electric conductivities, etc., and 4) sources. The first item is split to extract basic equations used in the Yee algorithm. The second item is used to separate time and space in order to interleave Maxwell's equations in space and time. The third item defines the medium and boundary conditions.

The last one, which we are going to discuss, is used to simulate a physical source or excite a structure to obtain the desired response.To unify the results, we use the same basic Parameters necessary for the FDTD method in the scripts. This Parameters are available in section 4.1.

For frequency-dependent lossy dielectric media, the dielectric constant and conductivity vary at different frequencies. For this kind of materials, the complex dielectric constant is of the form

$$\epsilon_r^*(\omega) = \epsilon_r + \frac{\sigma}{j\omega \cdot \epsilon_0} \tag{1}$$

where ϵ_r is the dielectric constant of the medium and σ is the conductivity of the medium.
The frequency-dependent displacement vector will be

$$\mathbf{D}(\omega) = \epsilon_r^*(\omega)\mathbf{E}(\omega)$$

Using eqn. (1) the above equation can be written as

$$\mathbf{D}(\omega) = \epsilon_r \mathbf{E}(\omega) + \frac{\sigma}{j\omega \cdot \epsilon_0}\mathbf{E}(\omega)$$

Using the inverse Fourier transform of the above equation we get,

$$\mathbf{D}(t) = \epsilon_r \mathbf{E}(t) + \frac{\sigma}{\epsilon_0}\int_0^t \mathbf{E}\left(t'\right) dt' \tag{2}$$

Define,

$$\mathbf{I}(t) = \frac{\sigma}{\epsilon_0} \int_0^t \mathbf{E}\left(t'\right) dt' \tag{3}$$

Therefore, the eqn. (2) becomes

$$\mathbf{D}(t) = \epsilon_r \mathbf{E}(t) + \mathbf{I}(t) \tag{4}$$

Approximating the above integral as a summation in the sampled time domain, we get eqn. (3) as

$$\mathbf{I}^n = \frac{\sigma \cdot \Delta t}{\epsilon_0} \sum_{i=0}^{n} \mathbf{E}^i \tag{5}$$

From eqn. (5) we get,

$$\mathbf{I}^n = \frac{\sigma \cdot \Delta t}{\epsilon_0} \mathbf{E}^n + \frac{\sigma \cdot \Delta t}{\epsilon_0} \sum_{i=0}^{n-1} \mathbf{E}^i \tag{6}$$

$$= \frac{\sigma \cdot \Delta t}{\epsilon_0} \mathbf{E}^n + \mathbf{I}^{n-1} \tag{7}$$

We can write the Displacement vector as

$$\mathbf{D}^n = \epsilon_r \mathbf{E}^n + \mathbf{I}^n \tag{8}$$

Putting

$$\mathbf{I}^n = \mathbf{I}^{n-1} + \frac{\sigma \cdot \Delta t}{\epsilon_0} \mathbf{E}^n$$

in eqn. (8) we get,

$$\mathbf{D}^n = \epsilon_r \mathbf{E}^n + \left(\frac{\sigma \cdot \Delta t}{\epsilon_0} \mathbf{E}^n + \mathbf{I}^{n-1} \right)$$

From the above equation we get,

$$\mathbf{E}^n = \frac{\mathbf{D}^n - \mathbf{I}^{n-1}}{\epsilon_r + \frac{\sigma \cdot \Delta t}{\epsilon_0}}$$

For the simulation using FDTD, we are going to use the following equations.

$$\mathbf{E}^n = \frac{\mathbf{D}^n - \mathbf{I}^{n-1}}{\epsilon_r + \frac{\sigma \cdot \Delta t}{\epsilon_0}} \tag{9}$$

$$\mathbf{I}^n = \mathbf{I}^{n-1} + \frac{\sigma \cdot \Delta t}{\epsilon_0} \mathbf{E}^n \tag{10}$$

1.1 Implementation of FDTD in 1D

The popular FDTD method belong to the general class of grid-based differential numerical modeling methods (finite difference methods). We discretized time-dependent maxwell's equation using central difference approximation applied to the spatial and time derivative part. The resulting finite-difference equations are solved in computer(here we used python): the electric field vector components in a space volume are solved at a given instant in time; then the magnetic field vector components within the same spatial volume are solved at the next instant in time; and therefore the process is repeated over and yet again until the specified transient or steady-state electromagnetic field behavior is fully evolved.

When Maxwell's equations are analysed, it is seen that the change within the E-field in time (the time derivative) relies on the change within the H-field across space (the curl). This results in the usual FDTD time-stepping relation that, at any spatial point, the updated time dependent value of the E-field is dependent on the stored value of the E-field and the curl of the local spatial distribution of the magnetic field also usually referred as update equations.

The H-field is time-stepped in likewise manner. At any point in space, the updated value of the H -field in time depends on the stored value of the H-field and therefore the curl of the spatial distribution of the E-field locally . Iterating the E-field and H-field update results in a real time process wherein sampled-data analogs of the continuous electromagnetic waves under consideration propagate under a numerical grid stored within the computer memory.

This description holds true for $1-\mathrm{D}, 2-\mathrm{D}$, and also for three dimensional FDTD techniques. For simplicity we used 1-D techniques only. It has already been proposed in Yee's paper(1966) "spatially staggering the vector components of the E-field and H-field about rectangular unit cells of a Cartesian computational grid in order that each E-field vector component is located midway between a pair of H -field vector components, and conversely." This scheme, is well known by the name of Yee lattice, has proven to be very robust, has been used extensively, and remains at the core of many of the current FDTD commercial software constructs.

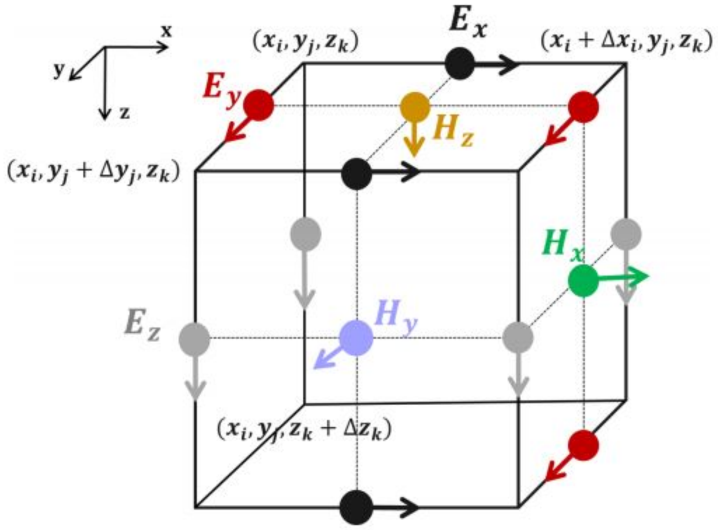


Figure 1: The staggered Yee grid used to define the positions of the electric and magnetic field nodes. The electric and magnetic field components are allocated respectively to cell edges and faces. Image reproduced from doi: 10.1093/gji/ggv377

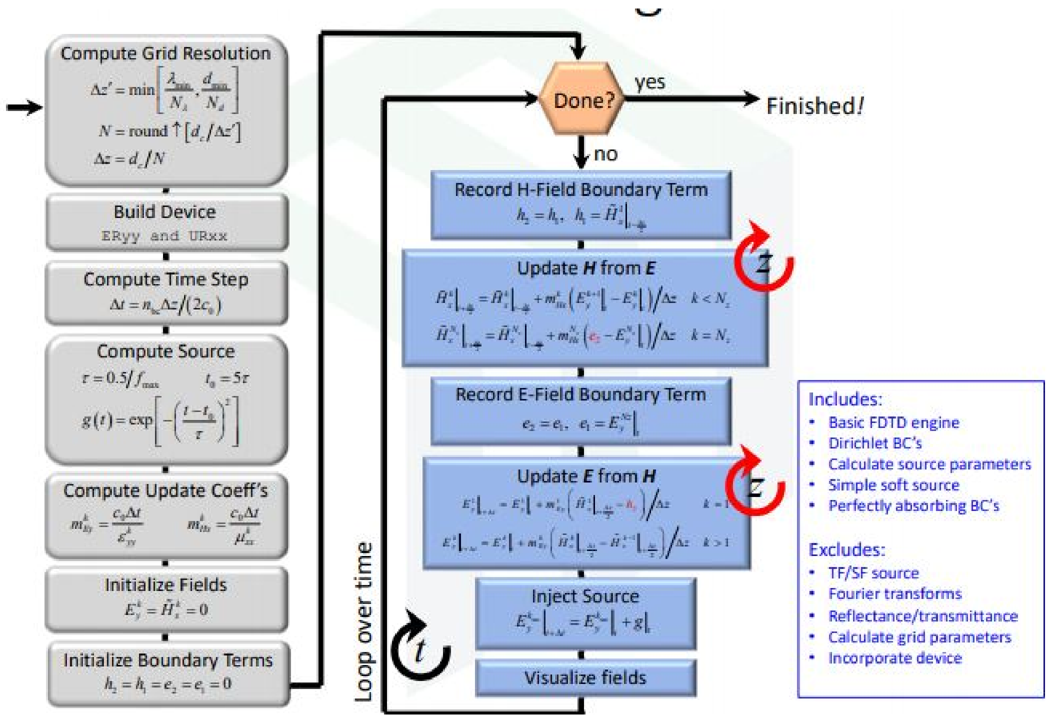


Figure 2: The basic algorithm for implementing FDTD, here source used in soft source. Image reproduced from https://empossible.net/

2. Kind of Sources

There are many kind of sources that are added during FDTD simulation. Some of them are described here

2.0.1 Simple Hard Source

The simple hard source is the easiest to implement. After updating the field across the entire grid, one field component at one point on the grid is replaced with the source. This approach inserts power into the model, but the source point behaves like a perfect electric conductor or perfect magnetic conductor and will scatter waves which is not desired.
We used this source just to test if our code works fine as this source is easiest to implement.

2.0.2 Simple Soft Source

The simple soft source is better than the hard source because it is transparent to scattered waves passing through it. After updating the field across the entire grid, the source function is added to one field component at one point on the grid. This approach administer power into the model in both directions.
It is great for testing boundary conditions.

2.0.3 Total-Field / Scattered-Field

The total-field/scattered-field (TF/SF) is a technique to inject a "one-way" source.
Benefits:

- No backward propagating waves

- Make sure waves at the boundaries are only travelling outward
- Complete power andmisitered by the source is incident on the device or dielectric being simulated.

We have used this as a source in our project.

3. Dielectric Properties of Human Muscle Tissue

When we hit the surface of a material with Radio Frequency(RF) only certain part of it gets absorbed and the rest of the part gets reflected and some part gets transmitted. The energy can be defined in terms of dielectric properties of the materials. For any material dielectric properties are of great use to study the interaction of electromagnetic waves with it constituents.Here complex permittivity helps us to find all these. The real part of the complex permittivity gives the relative permittivity which measures the energy stored on the material while the imaginary part measures the dielectric loss factor. Dielectric loss factor is the measure of dissipated electric energy.The complex permittivity is depends on frequency and it is given by the following equation.

$$\epsilon(f) = \epsilon'(f) - \epsilon''(f)$$

$$\epsilon'(f) = \epsilon_0 \epsilon_r(f)$$

where ϵ_0 is the permittivity of the free space , ϵ_r is the relative permittivity (reprents the energy stored in the medium), $\epsilon^{''}$ is the out of phase loss factor representing loss of energy within the medium. The above dielectric can be modified into the following:

$$\epsilon(f) = \epsilon_r(f) - \frac{\sigma(f)}{\omega \epsilon_0}$$

where σ is the electrical conductivity of the material and ω is the angular frequency of the field. We can parametrize loss factor in terms of the loss tangent by the following equation.

$$\tan\delta = \frac{\epsilon^{''}(f)}{\epsilon^{'}(f)}$$

It is known from the analysis that when EM waves from one biological tissue to another the impedance difference between the two tissue types results in reflection of some energy. Hence, it reduces the power of signal which travels to other side of the interface. We can find the corresponding reflection and transmission coefficient from which used to calculate dielectric properties of the tissue. From literature it is also known that biological tissues respond weakly to the magnetic field, so we can assume it's permeability to be approximately unity.

4. Parameters required for our project

4.1 Permittivity and conductivity of different tissues at various frequency

These following results [1] are already known .

Table 1: At the frequency 100 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	72.9	0.4910
Fat	12.7	0.0684
Muscle	66.0	0.7080
Cartilage	55.8	0.4750
Lung	31.6	0.3060
Brain(GM)	80.1	0.5590

Table 2: At the frequency 200 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	55.7	0.5820
Fat	12.0	0.0726
Muscle	60.2	0.7430
Cartilage	49.2	0.5180
Lung	26.6	0.3350
Brain (GM)	65.1	0.6390

Table 3: At the frequency 300 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	55.7	0.5820
Fat	12.0	0.0726
Muscle	60.2	0.7430
Cartilage	49.2	0.5180
Lung	26.6	0.3350
Brain (GM)	65.1	0.6390

Table 4: At the frequency 400 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	46.8	0.6880
Fat	11.6	0.0807
Muscle	57.1	0.7960
Cartilage	45.5	0.5860
Lung	23.8	0.3740
Brain (GM)	57.4	0.7370

Table 5: At the frequency 500 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	44.9	0.7280
Fat	11.5	0.0854
Muscle	56.4	0.8220
Cartilage	44.6	0.6210
Lung	23.2	0.3910
Brain(GM)	55.8	0.7790

Table 6: At the frequency 600 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	43.6	0.7650
Fat	11.5	0.0905
Muscle	56.0	0.8500
Cartilage	44.0	0.6580
Lung	22.8	0.4070
Brain (GM)	54.7	0.8190

Table 7: At the frequency 700 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.0010
Skin	42.7	0.8000
Fat	11.4	0.0962
Muscle	55.6	0.8790
Cartilage	43.5	0.6970
Lung	22.5	0.4230
Brain(GM)	53.9	0.8600

Table 8: At the frequency 800 MHz

Tissue	Epsilon(Permittivity)	Sigma(Conductivity)
Air	1.0	0.001
Skin	42.0	0.834
Fat	11.4	0.102
Muscle	55.3	0.910
Cartilage	43.0	0.738
Lung	22.2	0.440
Brain(GM)	53.3	0.900

5. Observations

All the calculations of Brewster's angle, Skin depth and reflection has been done by using the code given in Appendix C, D and E using basic formulae taken from standard optics text.

5.1 Brewster's angle, Reflectance and Skin Depth of Tissues at various frequencies

Table 9: At the frequency 100 Hz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	503.292121
Skin	83.319872	0.624599	22.713255
Fat	74.325521	0.315574	60.854413
Muscle	82.982676	0.609647	18.914876
Cartilage	82.375153	0.583499	23.092626
Lung	79.913057	0.487132	28.771296
Brain (GM)	83.624580	0.638388	21.286989

Table 10: At the frequency 200 MHz

Tissue	Brewster Angle (in degree)	Reflectance	Skin Depth (in metre)
Air	45.000000	0.000000	355.881272
Skin	82.368393	0.583213	14.751753
Fat	73.897886	0.304684	41.767341
Muscle	82.655930	0.595459	13.056018
Cartilage	81.886219	0.563168	15.636525
Lung	79.026979	0.455887	19.443871
Brain(GM)	82.934823	0.607551	14.078446

Table 11: At the frequency 300 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	290.575842
Skin	81.934598	0.565152	11.477056
Fat	73.703578	0.299828	33.222230
Muscle	82.532210	0.590163	10.464833
Cartilage	81.683613	0.554924	12.356542
Lung	78.645735	0.442956	15.400489
Brain(GM)	82.643834	0.594939	11.046036

Table 12: At the frequency 400 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	251.646061
Skin	81.683613	0.554924	9.593916
Fat	73.637228	0.298182	28.012597
Muscle	82.461441	0.587152	8.919358
Cartilage	81.567328	0.550239	10.395403
Lung	78.415973	0.435308	13.012308
Brain(GM)	82.480941	0.587980	9.269502

Table 13: At the frequency of 500 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	225.079079
Skin	81.511988	0.548022	8.341986
Fat	73.570060	0.296524	24.356013
Muscle	82.415345	0.585198	7.850534
Cartilage	81.483906	0.546899	9.032112
Lung	78.271234	0.430544	11.382738
Brain(GM)	82.375153	0.583499	8.064295

Table 14: At the frequency 600 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	205.468148
Skin	81.388246	0.543091	7.428717
Fat	73.570060	0.296524	21.598332
Muscle	82.388621	0.584068	7.047499
Cartilage	81.426895	0.544627	8.009982
Lung	78.171671	0.427292	10.184678
Brain(GM)	82.299779	0.580324	7.179638

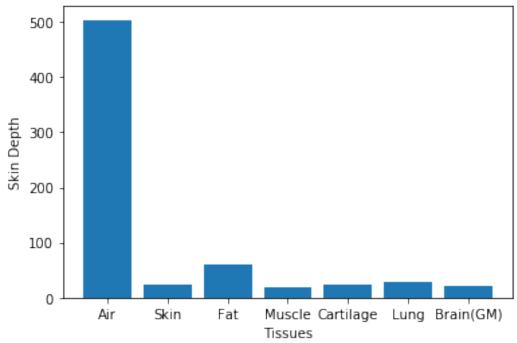
Table 15: At the frequency 700 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	190.226541
Skin	81.299337	0.539571	6.725524
Fat	73.502057	0.294851	19.394723
Muscle	82.361614	0.582927	6.416182
Cartilage	81.378501	0.542704	7.205344
Lung	78.095312	0.424811	9.249131
Brain(GM)	82.243536	0.577964	6.486671

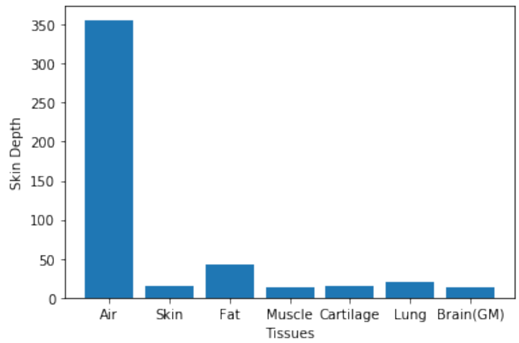
Table 16: At the frequency 800 MHz

Tissue	Brewster Angle	Reflectance	Skin Depth
Air	45.000000	0.000000	177.940636
Skin	81.228250	0.536771	6.161580
Fat	73.502057	0.294851	17.618748
Muscle	82.341168	0.582065	5.898675
Cartilage	81.329279	0.540754	6.550085
Lung	78.017455	0.422293	8.482987
Brain(GM)	82.200535	0.576166	5.931355

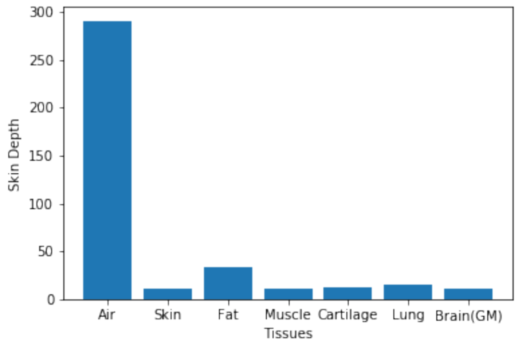
5.2 Graph of skin depth vs different tissues at different frequency



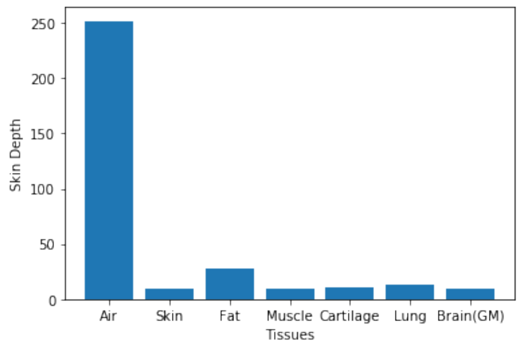
(a) At frequency 100 Hz



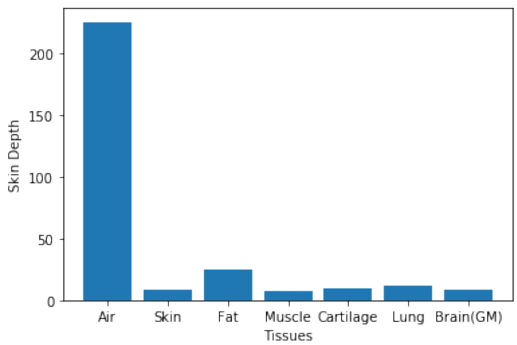
(b) At frequency 200 Hz



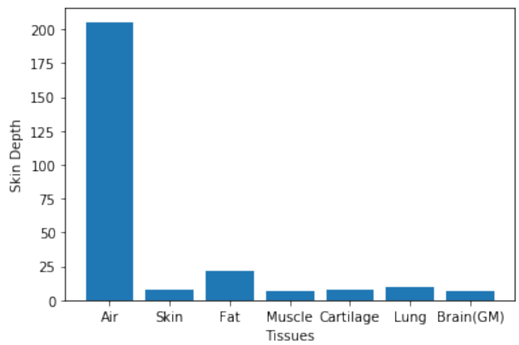
(c) At frequency 300 Hz



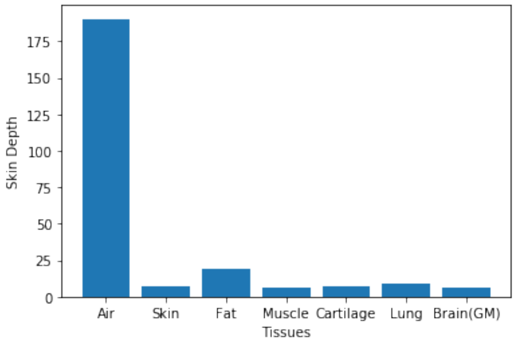
(d) At frequency 400 Hz



(e) At frequency 500 Hz

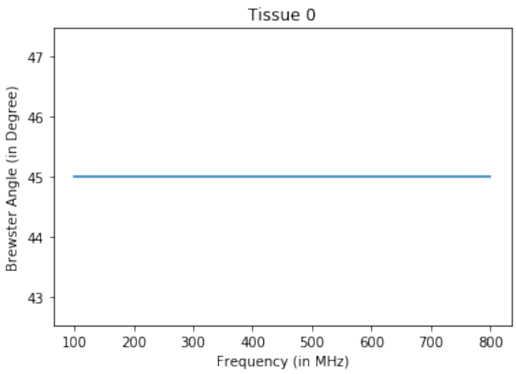


(f) At frequency 600 Hz

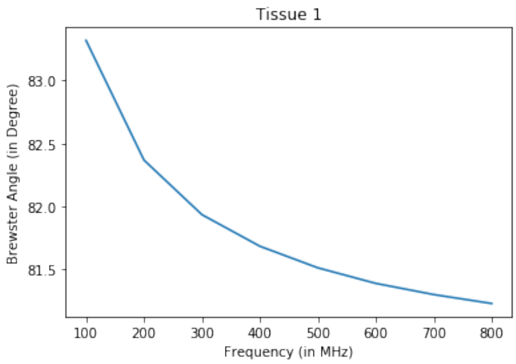


(g) At frequency 700 Hz

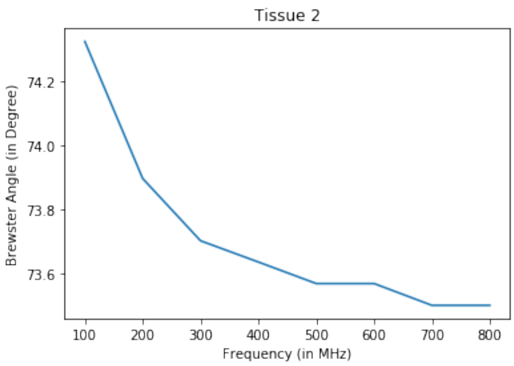
5.3 Graph of Brewster's angle vs frequency for different tissues



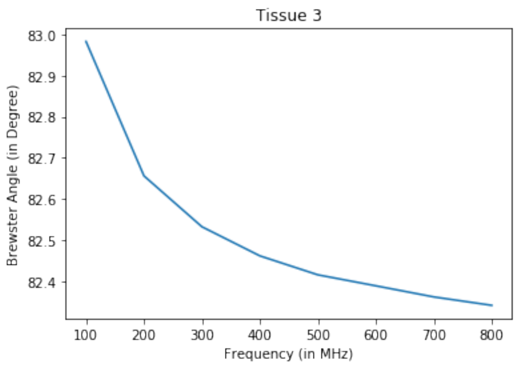
(h) Graph of Brewster's angle vs frequency for Air



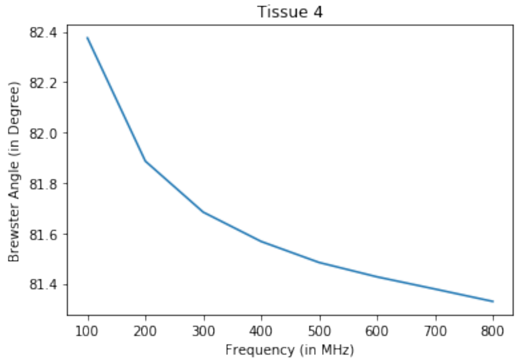
(i) Graph of Brewster's angle vs frequency for Skin



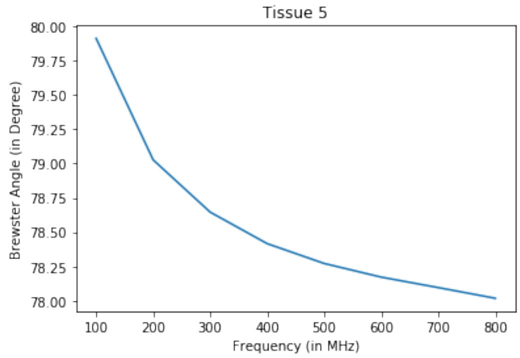
(j) Graph of Brewster's angle vs frequency for Fat



(k) Graph of Brewster's angle vs frequency for Muscle



(l) Graph of Brewster's angle vs frequency for Cartilage



(m) Graph of Brewster's angle vs frequency for Lung

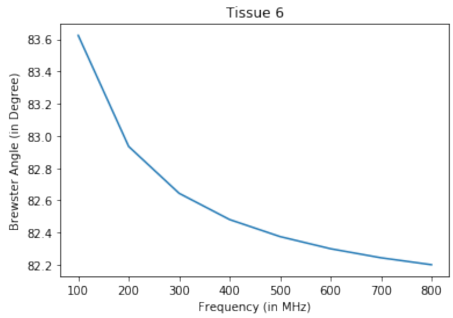
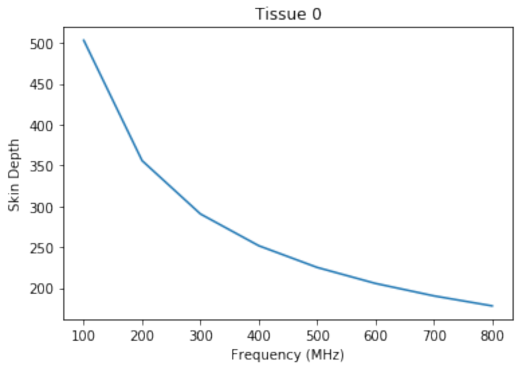
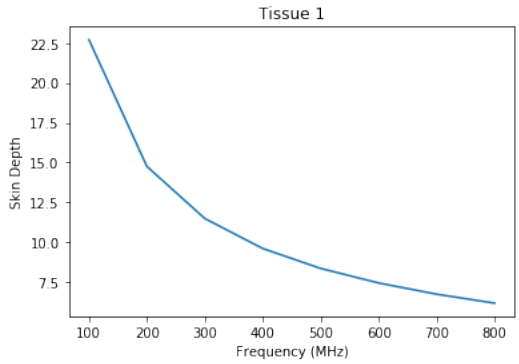


Figure 3: Graph of Brewster's angle vs frequency for Brain (GM)

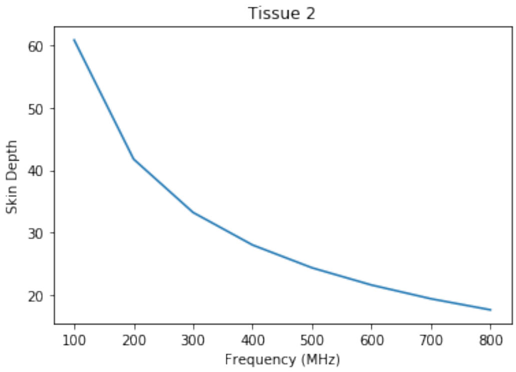
5.4 Skin Depth vs. Frequency for different tissues



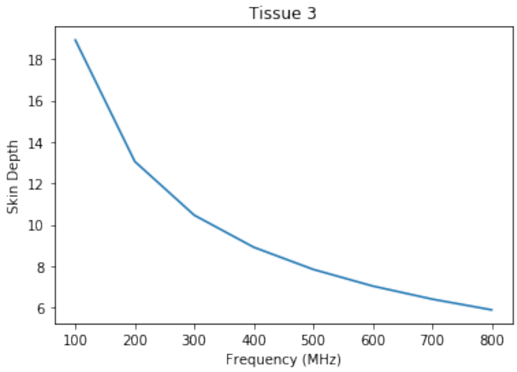
(a) Graph of Skin Depth vs. Frequency for Air



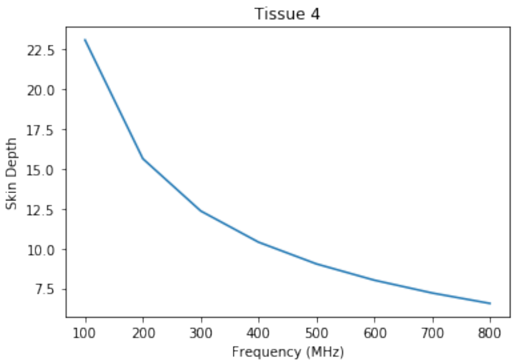
(b) Graph of Skin Depth vs. Frequency for Skin



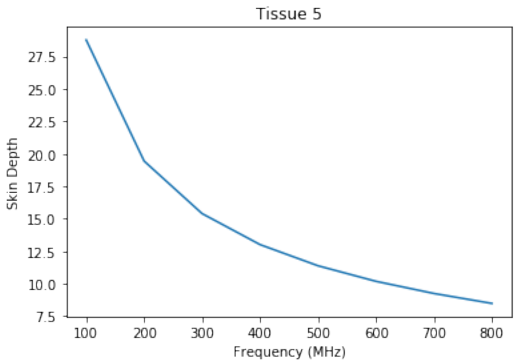
(c) Graph of Skin Depth vs. Frequency for Fat



(d) Graph of Skin Depth vs. Frequency for Muscle



(e) Graph of Skin Depth vs. Frequency for Cartilage



(f) Graph of Skin Depth vs. Frequency for Lung

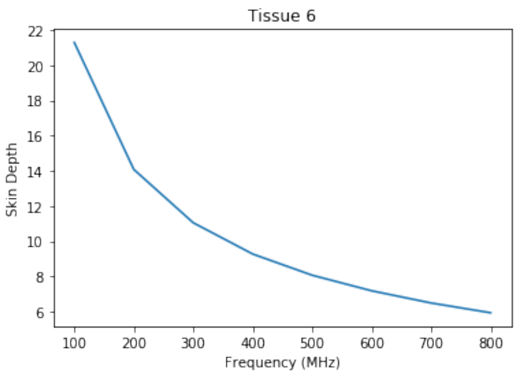
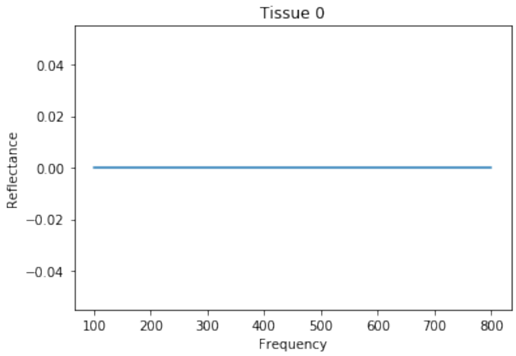
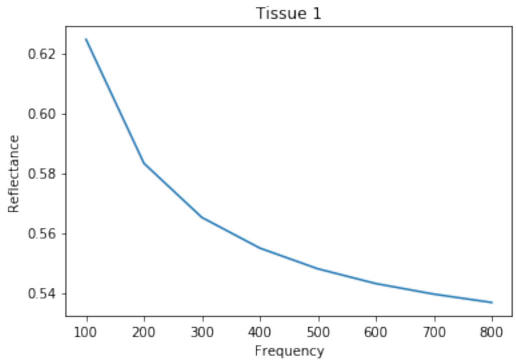


Figure 4: Graph of Skin Depth vs. Frequency for Brain

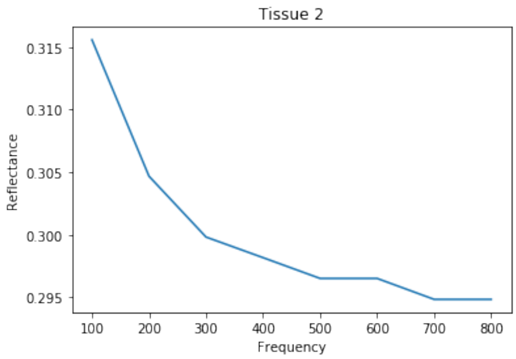
5.5 Reflectance vs. Frequency for different tissues



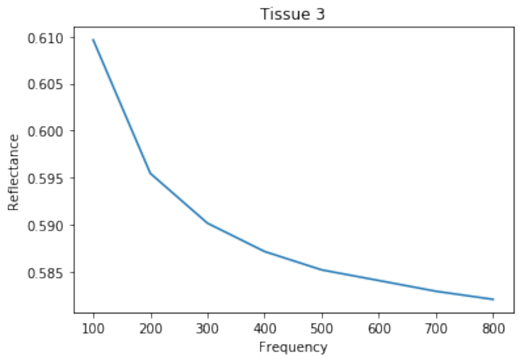
(a) Graph of Reflectance vs. Frequency for Air



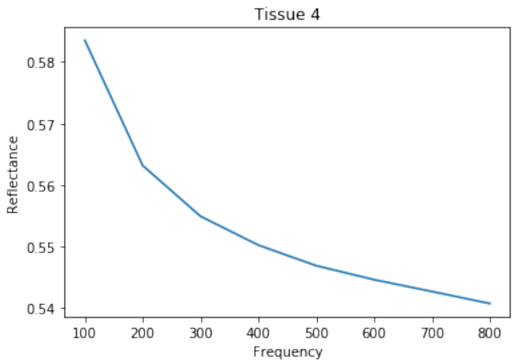
(b) Graph of Reflectance vs. Frequency for Skin



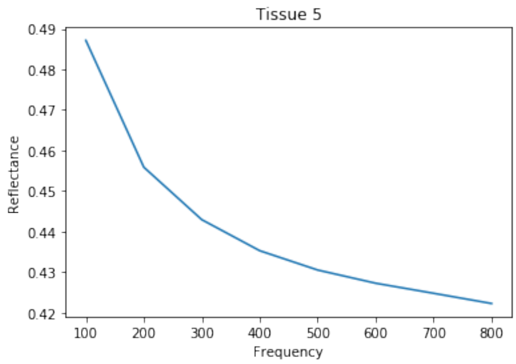
(c) Graph of Reflectance vs. Frequency for Fat



(d) Graph of Reflectance vs. Frequency for Muscle



(e) Graph of Reflectance vs. Frequency for Cartilage



(f) Graph of Reflectance vs. Frequency for Lung

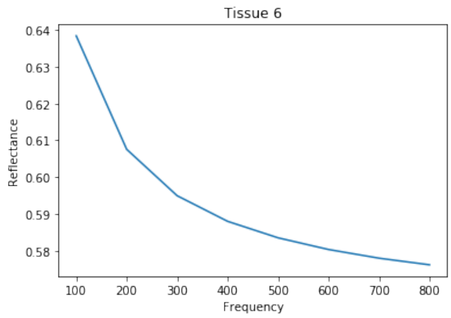


Figure 5: Graph of Reflectance vs. Frequency for Brain (Grey Matter)

5.6 Observation from the plots

5.6.1 Frequency vs skin depth

When a radiation is penetrated a depth of $\frac{1}{K_I}$, the amplitude is decreased by the factor $\frac{1}{e}$. This factor $\delta = \frac{1}{K_I}$, , is called the skin depth.

$$\delta = \sqrt{\frac{2}{\sigma \mu_0 \omega}}$$

Hence δ is inversely proportional to the square-root of the frequency f and also for a constant frequency the skin depth is inversely proportional to the square-root of the conductivity. We observed that our plots satisfies the above equation and it confirms this relationship and also from the plots we infer the order of the conductivity of the different tissues. We found the following order of conductivity Muscle $>$ Brain $>$ Skin $>$ Cartilage $>$ Lung $>$ Fat $>$ Air. This perfectly agrees with the conductivity of these tissues which we found in the literature.

5.6.2 Frequency vs Brewster's angle

The angle at which , there is no reflection at all is known as Brewster's angle. It depends on the media in which the EM wave is travelling. We studied frequency vs Brewster's angle for different tissues and try to analyse those. We have assume the first medium to be air. The Brewster's angle is given by the following formula

$$\tan \theta_B = \frac{n_2}{n_1}$$

where $\tan \theta_B$ is the Brewster's angle and n_1 is the refractive index of first medium and n_2 is the refractive index of the second medium From the plotting we get exactly what we expect. If both the media are air than the Brewster's angle is exactly 45^0 and in other tissues like muscle, heart etc. we found that the Brewster's angle is inversely varying with frequency.

5.6.3 Frequency vs Reflectance

The reflectance or the reflection coefficient is given by the ratio of the intensity of the reflected wave and intensity of the incident wave. Which can be expressed as follows

$$R = \frac{I_R}{I_I}$$

As reflectance depends on refractive index and refractive index is also depended on e frequency it turns out that we should expect some relationship between frequency and the reflectance. We try to plot it for different type of tissues. We also have plotted it for air. As expected we didn't see any dependence of reflectance with frequency in this case whereas for tissues we observed as inverse relationship of reflectance with frequency which is well known in literature. Hence we confirm this dependence as well.

6. Observation

For observation see the section Figures as Output 8.

7. Assumptions

- We are using the Debye model
- We are ignoring the thickness of the different tissues as in the human body
- We take $\chi = 0$

8. Conclusion

Using the Debye Model, we have simulated the mechanism of EM Waves striking different human body tissues (namely, fat, muscle, lung, brain, cartilage and skin). We confirmed the results for how skin depth changes with frequency using simulation and how different tissues react to different frequencies of incident EM Waves. We were successful in simulating striking of pulse to different human as shown below for muscles and in the Appendix K for rest of the tissues. We also find the relations for transmittance and frequency for different tissues

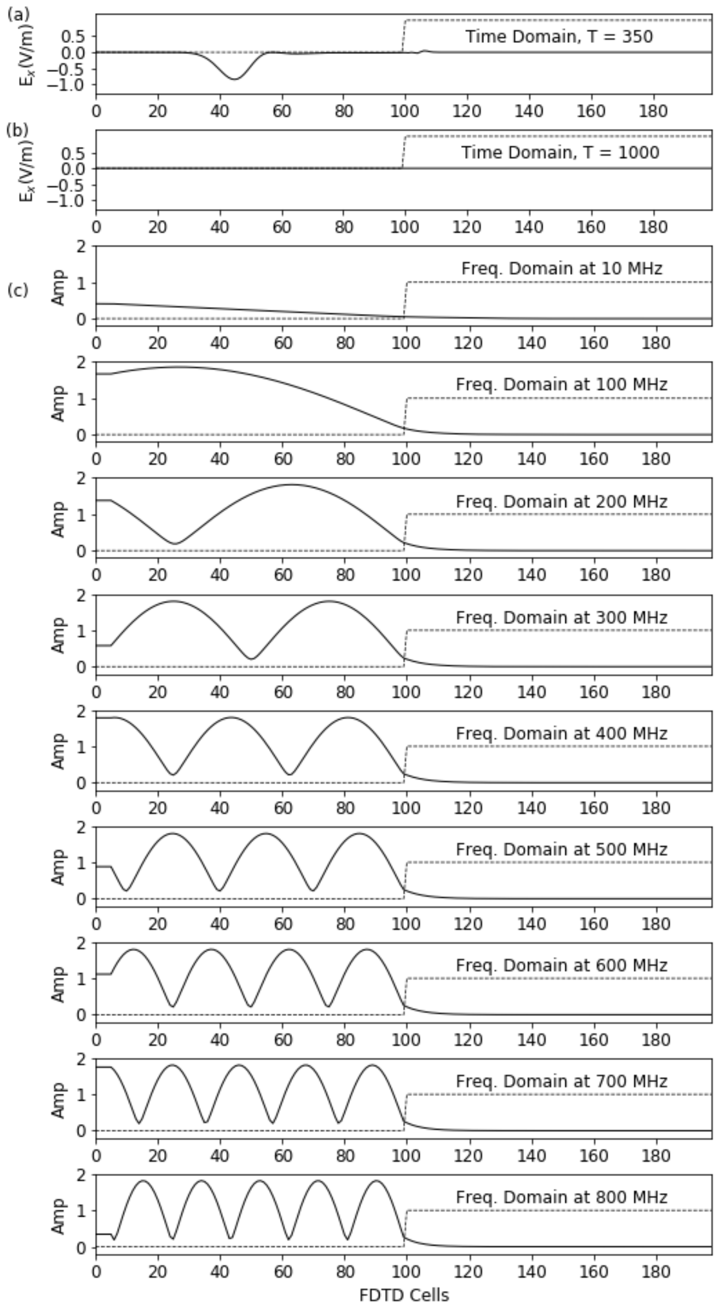


Figure 6: Muscle-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

Acknowledgement

We all are thankful to Prof. Venugopal Achanta for giving us a platform for presenting our views in this report.

References

Online Resources

[1] https://itis.swiss/virtual-population/tissue-properties/database/dielectric-properties/ (cited on page 8).

[2] https://school.eecs.wsu.edu/.

[3] https://www.lumerical.com/.

[4] https://edaboard.com.

[5] http://emlab.utep.edu/.

[6] https://www.sciencedirect.com/.

[7] https://onlinelibrary.wiley.com/.

[8] https://www.geeksforgeeks.org/.

[9] https://empossible.net/academics/emp5304/.

Article

[10] A. D'Orazio et al. "FINITE DIFFERENCE TIME DOMAIN MODELING OF LIGHT AMPLIFICATION IN ACTIVE PHOTONIC BAND GAP STRUCTURES - Abstract". In: *Journal of Electromagnetic Waves and Applications* 17.6 (2003), pages 855–857. DOI: 10.1163/156939303322503420. eprint: https://doi.org/10.1163/156939303322503420. URL: https://doi.org/10.1163/156939303322503420.

[11] Ronald Pethig. "Dielectric properties of body tissues". In: *Clinical physics and physiological measurement : an official journal of the Hospital Physicists' Association, Deutsche Gesellschaft für Medizinische Physik and the European Federation of Organisations for Medical Physics* 8 Suppl A (Feb. 1987), pages 5–12. DOI: 10.1088/0143-0815/8/4A/002.

Appendix A Disclaimer

If you are using Linux, please use Jupyter lab or Jupyter notebook to run the simulations otherwise the output maynot appeal you because the .png file is so long that it its output may look congested, otherwise please add " fig.savefig('xxxname.png') " to line number 154 or line number 218 to save the file(We have shared in the classroom), then you can see and analyse the outputs.

Appendix B Values and Graphs

```python
#!/usr/bin/env python
# coding: utf-8

# In[105]:


from math import pi, sqrt, atan
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

number = 7
number_of_frequencies = 8
permeability = 4*pi*10e-7

epsr = np.zeros((number_of_frequencies,number))
sigma = np.ones((number_of_frequencies,number))

angle = np.zeros(number)
reflectance = np.zeros(number)
skin_depth = np.zeros(number)
n = np.zeros(number)

epsr = [[1,72.9,12.7,66.0,55.8,31.6,80.1],    # 100 MHz
        [1,55.7,12.0,60.2,49.2,26.6,65.1],    # 200 MHz
        [1,49.8,11.7,58.2,46.8,24.8,60.0],    # 300 MHz
        [1,46.8,11.6,57.1,45.5,23.8,57.4],    # 400 MHz
        [1,44.9,11.5,56.4,44.6,23.2,55.8],    # 500 MHz
        [1,43.6,11.5,56.0,44.0,22.8,54.7],    # 600 MHz
        [1,42.7,11.4,55.6,43.5,22.5,53.9],    # 700 MHz
        [1,42.0,11.4,55.3,43.0,22.2,53.3]]    # 800 MHz

sigma = [[0.001,0.491,0.0684,0.708,0.475,0.306,0.559],
         [0.001,0.582,0.0726,0.743,0.518,0.335,0.639],
         [0.001,0.641,0.0765,0.771,0.553,0.356,0.692],
         [0.001,0.688,0.0807,0.796,0.586,0.374,0.737],
         [0.001,0.728,0.0854,0.822,0.621,0.391,0.779],
         [0.001,0.765,0.0905,0.850,0.658,0.407,0.819],
         [0.001,0.800,0.0962,0.879,0.697,0.423,0.860],
         [0.001,0.834,0.102,0.910,0.738,0.440,0.900]]
```

```
freq = [100,200,300,400,500,600,700,800]

for j in range(number_of_frequencies):
    x=[]
    y=[]
    z=[]
    omega = 2*pi*freq[j]
    print("Frequency = ",freq[j],"MHz")
    for k in range(number):
        n[k] = sqrt(epsr[j][k])

        angle[k] = atan(n[k])*(180/pi)
        reflectance[k] = ((1-n[k])/(1+n[k]))**2
        skin_depth[k] = sqrt(2/(sigma[j][k]*permeability*omega))

        x.append(angle[k])
        y.append(reflectance[k])
        z.append(skin_depth[k])

    for k in range(number):
        df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                           "Cartilage","Lung","Brain(GM)"],
                           "Brewster Angle":x,
                           "Reflectance":y,
                           "Skin Depth":z})


    display(df)
    print("    ")

for j in range(number):
    print("Frequency = ",freq[j],"MHz")
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.set(xlabel="Tissues",ylabel="Skin Depth")
    skd = []
    omega = 2*pi*freq[j]
    for k in range(number):
        skd.append(sqrt(2/(sigma[j][k]*permeability*omega)))
        langs = ["Air","Skin","Fat","Muscle","Cartilage","Lung","Brain(GM)"]
    ax.bar(langs,skd)
    plt.show()



# In[90]:
```

```
print ("Tabulated Values ::")
print (" ")
print ("Frequency = 100 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat",
                             "Muscle","Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)"
                             : [1,72.9,12.7,66.0,55.8,31.6,80.1],
                             "Sigma(Conductivity)"
                             : [0.001,0.491,0.0684,0.708,0.475,0.306,0.559]})
display(df)
print("  ")


# In[91]:



print (" ")
print ("Frequency = 200 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,55.7,12.0,60.2,49.2,26.6,65.1],
                             "Sigma(Conductivity)":
                             [0.001,0.582,0.0726,0.743,0.518,0.335,0.639]})
display(df)
print("  ")


# In[92]:



print (" ")
print ("Frequency = 300 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                           "Epsilon(Permittivity)":
                           [1,49.8,11.7,58.2,46.8,24.8,60.0],
                           "Sigma(Conductivity)":
                           [0.001,0.641,0.0765,0.771,0.553,0.356,0.692]})
display(df)
print("  ")


# In[93]:
```

```
print (" ")
print ("Frequency = 400 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,46.8,11.6,57.1,45.5,23.8,57.4],
                             "Sigma(Conductivity)":
                             [0.001,0.688,0.0807,0.796,0.586,0.374,0.737]})
display(df)
print("  ")


# In[94]:


print (" ")
print ("Frequency = 500 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,44.9,11.5,56.4,44.6,23.2,55.8],
                             "Sigma(Conductivity)":
                             [0.001,0.728,0.0854,0.822,0.621,0.391,0.779]})
display(df)
print("  ")


# In[95]:



print (" ")
print ("Frequency = 600 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,43.6,11.5,56.0,44.0,22.8,54.7],
                             "Sigma(Conductivity)":
                             [0.001,0.765,0.0905,0.850,0.658,0.407,0.819]})
display(df)
print("  ")
```

```
# In[96]:


print (" ")
print ("Frequency = 700 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,42.7,11.4,55.6,43.5,22.5,53.9],
                             "Sigma(Conductivity)":
                             [0.001,0.800,0.0962,0.879,0.697,0.423,0.860]})
display(df)
print("  ")


# In[97]:


print (" ")
print ("Frequency = 800 MHz")

df = pd.DataFrame({"Tissue":["Air","Skin","Fat","Muscle",
                             "Cartilage","Lung","Brain(GM)"],
                             "Epsilon(Permittivity)":
                             [1,42.0,11.4,55.3,43.0,22.2,53.3],
                             "Sigma(Conductivity)":
                             [0.001,0.834,0.102,0.910,0.738,0.440,0.900]})

display(df)
print("  ")


# In[ ]:
```

Appendix C Program For Brewster Angle vs Frequency

```
#!/usr/bin/env python
# coding: utf-8

# In[26]:


from math import pi, sqrt, atan
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
number = 7
number_of_frequencies = 8
permeability = 4*pi*10e-7

epsr = np.zeros((number_of_frequencies,number))
sigma = np.ones((number_of_frequencies,number))

angle = np.zeros(number_of_frequencies)
reflectance = np.zeros(number_of_frequencies)
skin_depth = np.zeros(number_of_frequencies)
n = np.zeros(number_of_frequencies)

epsr = [[1,72.9,12.7,66.0,55.8,31.6,80.1],    # 100 MHz
        [1,55.7,12.0,60.2,49.2,26.6,65.1],    # 200 MHz
        [1,49.8,11.7,58.2,46.8,24.8,60.0],    # 300 MHz
        [1,46.8,11.6,57.1,45.5,23.8,57.4],    # 400 MHz
        [1,44.9,11.5,56.4,44.6,23.2,55.8],    # 500 MHz
        [1,43.6,11.5,56.0,44.0,22.8,54.7],    # 600 MHz
        [1,42.7,11.4,55.6,43.5,22.5,53.9],    # 700 MHz
        [1,42.0,11.4,55.3,43.0,22.2,53.3]]    # 800 MHz

sigma = [[0.001,0.491,0.0684,0.708,0.475,0.306,0.559],
         [0.001,0.582,0.0726,0.743,0.518,0.335,0.639],
         [0.001,0.641,0.0765,0.771,0.553,0.356,0.692],
         [0.001,0.688,0.0807,0.796,0.586,0.374,0.737],
         [0.001,0.728,0.0854,0.822,0.621,0.391,0.779],
         [0.001,0.765,0.0905,0.850,0.658,0.407,0.819],
         [0.001,0.800,0.0962,0.879,0.697,0.423,0.860],
         [0.001,0.834,0.102,0.910,0.738,0.440,0.900]]

freq = [100,200,300,400,500,600,700,800]

print("Brewster Angle vs. Frequency for different tissues")
print(" ")
print("Tissue 0 : Air,Tissue 1 : Skin,Tissue 2 : Fat,Tissue 3 :\
       Muscle,Tissue 4 : Cartilage,Tissue 5 : Lung,Tissue 0 : Brain ")
print(" ")

for j in range(0,7):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.set_title('Tissue %d' %j )
    ax.set(xlabel="Frequency",ylabel="Brewster Angle")
    for k in range(0,8):
        n[k] = sqrt(epsr[k][j])
        angle[k] = atan(n[k])*(180/pi)
    ax.plot(freq,angle)
    plt.show()
```

```
# In[ ]:




# In[ ]:
```

Appendix D Program For Reflectance vs. Frequency

```
#!/usr/bin/env python
# coding: utf-8

# In[3]:


from math import pi, sqrt, atan
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

number = 7
number_of_frequencies = 8
permeability = 4*pi*10e-7

epsr = np.zeros((number_of_frequencies,number))
sigma = np.ones((number_of_frequencies,number))

angle = np.zeros(number_of_frequencies)
reflectance = np.zeros(number_of_frequencies)
skin_depth = np.zeros(number_of_frequencies)
n = np.zeros(number_of_frequencies)

epsr = [[1,72.9,12.7,66.0,55.8,31.6,80.1],    # 100 MHz
        [1,55.7,12.0,60.2,49.2,26.6,65.1],    # 200 MHz
        [1,49.8,11.7,58.2,46.8,24.8,60.0],    # 300 MHz
        [1,46.8,11.6,57.1,45.5,23.8,57.4],    # 400 MHz
        [1,44.9,11.5,56.4,44.6,23.2,55.8],    # 500 MHz
        [1,43.6,11.5,56.0,44.0,22.8,54.7],    # 600 MHz
        [1,42.7,11.4,55.6,43.5,22.5,53.9],    # 700 MHz
        [1,42.0,11.4,55.3,43.0,22.2,53.3]]    # 800 MHz

sigma = [[0.001,0.491,0.0684,0.708,0.475,0.306,0.559],
         [0.001,0.582,0.0726,0.743,0.518,0.335,0.639],
         [0.001,0.641,0.0765,0.771,0.553,0.356,0.692],
         [0.001,0.688,0.0807,0.796,0.586,0.374,0.737],
         [0.001,0.728,0.0854,0.822,0.621,0.391,0.779],
```

```
          [0.001,0.765,0.0905,0.850,0.658,0.407,0.819],
          [0.001,0.800,0.0962,0.879,0.697,0.423,0.860],
          [0.001,0.834,0.102,0.910,0.738,0.440,0.900]]

freq = [100,200,300,400,500,600,700,800]

print("Reflectance vs. Frequency for different tissues")
print(" ")
print("Tissue 0 : Air,Tissue 1 : Skin,Tissue 2 : Fat,Tissue 3 :\
       Muscle,Tissue 4 : Cartilage,Tissue 5 : Lung,Tissue 0 : Brain ")
print(" ")

for j in range(0,7):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.set_title('Tissue %d' %j )
    ax.set(xlabel="Frequency",ylabel="Reflectance")
    for k in range(0,8):
        n[k] = sqrt(epsr[k][j])
        reflectance[k] = ((1-n[k])/(1+n[k]))**2
    ax.plot(freq,reflectance)
    plt.show()


# In[ ]:
```

Appendix E Program for Frequency vs. Skin Depth

```
#!/usr/bin/env python
# coding: utf-8

# In[5]:


from math import pi, sqrt, atan
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

number = 7
number_of_frequencies = 8
permeability = 4*pi*10e-7

epsr = np.zeros((number_of_frequencies,number))
sigma = np.ones((number_of_frequencies,number))

angle = np.zeros(number_of_frequencies)
reflectance = np.zeros(number_of_frequencies)
skin_depth = np.zeros(number_of_frequencies)
```

```
n = np.zeros(number_of_frequencies)

epsr = [[1,72.9,12.7,66.0,55.8,31.6,80.1],    # 100 MHz
        [1,55.7,12.0,60.2,49.2,26.6,65.1],    # 200 MHz
        [1,49.8,11.7,58.2,46.8,24.8,60.0],    # 300 MHz
        [1,46.8,11.6,57.1,45.5,23.8,57.4],    # 400 MHz
        [1,44.9,11.5,56.4,44.6,23.2,55.8],    # 500 MHz
        [1,43.6,11.5,56.0,44.0,22.8,54.7],    # 600 MHz
        [1,42.7,11.4,55.6,43.5,22.5,53.9],    # 700 MHz
        [1,42.0,11.4,55.3,43.0,22.2,53.3]]    # 800 MHz

sigma = [[0.001,0.491,0.0684,0.708,0.475,0.306,0.559],
         [0.001,0.582,0.0726,0.743,0.518,0.335,0.639],
         [0.001,0.641,0.0765,0.771,0.553,0.356,0.692],
         [0.001,0.688,0.0807,0.796,0.586,0.374,0.737],
         [0.001,0.728,0.0854,0.822,0.621,0.391,0.779],
         [0.001,0.765,0.0905,0.850,0.658,0.407,0.819],
         [0.001,0.800,0.0962,0.879,0.697,0.423,0.860],
         [0.001,0.834,0.102,0.910,0.738,0.440,0.900]]

freq = [100,200,300,400,500,600,700,800]

print("Skin Depth vs. Frequency for different tissues")
print(" ")
print("Tissue 0 : Air,Tissue 1 : Skin,Tissue 2 : Fat,Tissue 3 :\
       Muscle,Tissue 4 : Cartilage,Tissue 5 : Lung,Tissue 0 : Brain ")
print(" ")

for j in range(0,7):
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.set_title('Tissue %d' %j )
    ax.set(xlabel="Frequency",ylabel="Skin Depth")
    for k in range(0,8):
        omega = 2*pi*freq[k]
        skin_depth[k] = sqrt(2/(sigma[k][j]*permeability*omega))
    ax.plot(freq,skin_depth)
    plt.show()


# In[ ]:




# In[ ]:
```

Appendix F Simulation Program for Air

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2
# initializing the variables
ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)
# creating the required geometry
ddx = 0.01                                  # Size of the cell
dt = ddx / 6e8                              # Time step size
number_of_frequencies = 9
# required frequency
freq_in = np.array((10e6, 100e6, 200e6, 300e6,
                    400e6, 500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# values of dielectric mediums
epsz = 8.854e-12
epsr = 1              # at 800MHz
sigma = 0           # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100
# boundary values
boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]
# iterative variables
gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
```

```
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.ones((number_of_frequencies, ke))

nsteps = 1000

# Dictionary to of desired timestep
plotting_points = [{'num_steps': 350, 'ex': None,
                    'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000,
                    'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]


# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx in timedomain
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

# Put a pulse in timedomain
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]

    # Calculate the Ex field from Dx in time domain
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Calculate the Fourier transform of Ex in frequency domain
    for k in range(ke):
        for m in range(number_of_frequencies):
            real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
```

```
            imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

# Fourier Transform of the input pulse before 200 timesteps
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Absorbing Boundary Conditions for ex
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

# Calculate the Hy field from ex
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

# Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                   (imag_pt[m, k]) ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = (amp[m, k])**2


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
```

```
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    plt.ylim(-0.2, 2.0)
    plt.text(170, 0.5, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
# saving the figure
fig.savefig('airtrns.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor,
                 gb_scaling_factor, y_ticks, y_min,
                 y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
```

```
            horizontalalignment='center')
    # plt.plot(gb * gb_scaling_factor / gb[120], 'k--',linewidth=0.75)
    # gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx,
                 plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'],
                 plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to for amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[5], 'amp': amp[5],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[6], 'amp': amp[6],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[7], 'amp': amp[7],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[8], 'amp': amp[8],
                   'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
```

```
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # gb for sacling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
# saving the graph
fig.savefig('airsimu.png')
```

Appendix G Simulation Program for Skin

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9

freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# Create Dielectric Profile
epsz = 8.854e-12
epsr = 42           # at 800MHz
sigma = 0.197         # at 10MHz
```

```
tau = 0.001 * 1e-6
chi = 0
k_start = 100

boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary to keep track of desired points for plotting
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]
```

```
# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]

    # Calculate the Ex field from Dx
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Calculate the Fourier transform of Ex
    for k in range(ke):
        for m in range(number_of_frequencies):
            real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
            imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

    # Fourier Transform of the input pulse
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

    # Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
```

```
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                 + imag_pt[m, k] ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.015, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
```

```
              plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
fig.savefig('skintran.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to keep track of plotting for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
```

```
                        {'freq': freq_in[5], 'amp': amp[5],
                         'label': '', 'x_label': ''},
                        {'freq': freq_in[6], 'amp': amp[6],
                         'label': '', 'x_label': ''},
                        {'freq': freq_in[7], 'amp': amp[7],
                         'label': '', 'x_label': ''},
                        {'freq': freq_in[8], 'amp': amp[8],
                         'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('skinsimu.png')
```

Appendix H   Simulation Program for Fat

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
```

```
sx = np.zeros(ke)
hy = np.zeros(ke)

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9

freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# Create Dielectric Profile
epsz = 8.854e-12
epsr = 53.3                   # at 800MHz
sigma = 2.92E-1         # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100

boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))
```

```
nsteps = 1000

# Dictionary to keep track of desired points for plotting
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]



# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]

    # Calculate the Ex field from Dx
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Calculate the Fourier transform of Ex
    for k in range(ke):
        for m in range(number_of_frequencies):
            real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
            imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

    # Fourier Transform of the input pulse
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
```

```
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

    # Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                  + imag_pt[m, k] ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
```

```
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.015, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
fig.savefig('fattran.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
```

```
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to keep track of plotting for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[5], 'amp': amp[5],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[6], 'amp': amp[6],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[7], 'amp': amp[7],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[8], 'amp': amp[8],
                   'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
```

```
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('fatsimu.png')
```

Appendix I Simulation Program For Muscle

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9

freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# Create Dielectric Profile
epsz = 8.854e-12
epsr = 55.3            # at 800MHz
sigma = 0.617         # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100

boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)
```

```
gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary to keep track of desired points for plotting
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]



# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]
```

```
    # Calculate the Ex field from Dx
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Calculate the Fourier transform of Ex
    for k in range(ke):
        for m in range(number_of_frequencies):
            real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
            imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

    # Fourier Transform of the input pulse
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

    # Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                  + imag_pt[m, k] ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
```

```
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[1], 'TRN': TRN[1, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[2], 'TRN': TRN[2, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[3], 'TRN': TRN[3, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[4], 'TRN': TRN[4, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[5], 'TRN': TRN[5, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[6], 'TRN': TRN[6, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[7], 'TRN':TRN[7, ],
                        'label': '', 'x_label': ''},
                       {'freq': freq_in[8], 'TRN': TRN[8, ],
                        'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.012, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
fig.savefig('muscletran.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
```

```
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to keep track of plotting for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[5], 'amp': amp[5],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[6], 'amp': amp[6],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[7], 'amp': amp[7],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[8], 'amp': amp[8],
                   'label': '', 'x_label': 'FDTD Cells'}]
```

```
def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('musclesimu.png')
sx = np.zeros(ke)
hy = np.zeros(ke)

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9

freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# Create Dielectric Profile
epsz = 8.854e-12
epsr = 55.3            # at 800MHz
sigma = 0.617          # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100
```

```
boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary to keep track of desired points for plotting
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]



# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx
```

```
for k in range(1, ke):
    dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

# Put a sinusoidal at the low end
pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
dx[5] = pulse + dx[5]

# Calculate the Ex field from Dx
for k in range(1, ke):
    ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
    ix[k] = ix[k] + gbx[k] * ex[k]
    sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

# Calculate the Fourier transform of Ex
for k in range(ke):
    for m in range(number_of_frequencies):
        real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
        imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

# Fourier Transform of the input pulse
if time_step < (190):
    for m in range(number_of_frequencies):
        real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
        imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

# Absorbing Boundary Conditions
ex[0] = boundary_low.pop(0)
boundary_low.append(ex[1])
ex[ke - 1] = boundary_high.pop(0)
boundary_high.append(ex[ke - 2])

# Calculate the Hy field
for k in range(ke - 1):
    hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Save data at certain points for later plotting
for plotting_point in plotting_points:
    if time_step == plotting_point['num_steps']:
        plotting_point['ex'] = np.copy(ex)

# Calculate the amplitude and phase at each frequency
        for m in range(number_of_frequencies):
            amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
            phase_in[m] = atan2(imag_in[m], real_in[m])
            for k in range(ke):
                amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                    + imag_pt[m, k] ** 2)
                phase[m, k] = atan2(imag_pt[m, k],
                                    real_pt[m, k]) - phase_in[m]
```

```
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.002, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
fig.savefig('muscletran.png')
```

```python
# Plot the outputs in Fig. 2.3
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to keep track of plotting for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[5], 'amp': amp[5],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[6], 'amp': amp[6],
                   'label': '', 'x_label': ''},
```

```
                  {'freq': freq_in[7], 'amp': amp[7],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[8], 'amp': amp[8],
                   'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('musclesimu.png')
```

Appendix J Simulation Program for Cartilage

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)
# creating the geometry
ddx = 0.01  # Cell size
```

```
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9
# required input frequency
freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# information about the dielectric mediums
epsz = 8.854e-12
epsr = 43            # at 800MHz
sigma = 3.69E-1         # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100
# initializing boundary Conditions
boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

# iterative variables
gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# variables for Fourier transform:
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary for the timestep
```

```
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]


# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx in time domain
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a pulse wave in time domain
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]

    # Calculate the Ex field from Dx in time domain
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Fourier Transform of the input pulse
    for k in range(ke):
        for m in range(number_of_frequencies):
            real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
            imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

    # Fourier Transform of the input pulse
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    #  Hy field calculations
```

```
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Saving the data plotting_points
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

# Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                    + imag_pt[m, k] ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)

# Dictionary for transmission
fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]

# plotting transmission
def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
```

```
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.015, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
# saving the figure
fig.savefig('cartilagetran.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plotting e field
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
```

```
                  plotting_point['label'],
                  plotting_point['label_loc'])

# Dictionary  for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                    'label': '(c)', 'x_label': ''},
                   {'freq': freq_in[1], 'amp': amp[1],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[2], 'amp': amp[2],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[3], 'amp': amp[3],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[4], 'amp': amp[4],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[5], 'amp': amp[5],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[6], 'amp': amp[6],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[7], 'amp': amp[7],
                    'label': '', 'x_label': ''},
                   {'freq': freq_in[8], 'amp': amp[8],
                    'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
```

```
fig.savefig('cartilagesimu.png')
```

Appendix K   Simulation Program for Lungs

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9

freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# Create Dielectric Profile
epsz = 8.854e-12
epsr = 22.2             # at 800MHz
sigma = 2.25E-1          # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100

boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]

gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# To be used in the Fourier transform
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
```

```
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary to keep track of desired points for plotting
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]



# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx
    for k in range(1, ke):
        dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    dx[5] = pulse + dx[5]

    # Calculate the Ex field from Dx
    for k in range(1, ke):
        ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
        ix[k] = ix[k] + gbx[k] * ex[k]
        sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

    # Calculate the Fourier transform of Ex
    for k in range(ke):
        for m in range(number_of_frequencies):
```

```
                real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
                imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

    # Fourier Transform of the input pulse
    if time_step < (190):
        for m in range(number_of_frequencies):
            real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
            imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])
    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
            plotting_point['ex'] = np.copy(ex)

    # Calculate the amplitude and phase at each frequency
            for m in range(number_of_frequencies):
                amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
                phase_in[m] = atan2(imag_in[m], real_in[m])
                for k in range(ke):
                    amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                + imag_pt[m, k] ** 2)
                    phase[m, k] = atan2(imag_pt[m, k],
                                        real_pt[m, k]) - phase_in[m]
                for k in range(100, ke):
                    TRN[m, k] = ((amp[m, k])**2)*sqrt(epsr)


fig = plt.figure(figsize=(8, 16))
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
```

```
                     'label': '', 'x_label': ''},
                    {'freq': freq_in[6], 'TRN': TRN[6, ],
                     'label': '', 'x_label': ''},
                    {'freq': freq_in[7], 'TRN':TRN[7, ],
                     'label': '', 'x_label': ''},
                    {'freq': freq_in[8], 'TRN': TRN[8, ],
                     'label': '', 'x_label': 'FDTD Cells'}]


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.02, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
fig.savefig('lungtran.png')

plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
```

```
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary to keep track of plotting for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[5], 'amp': amp[5],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[6], 'amp': amp[6],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[7], 'amp': amp[7],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[8], 'amp': amp[8],
                   'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
```

```
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return

# Plot the amplitude at each of the frequencies of interest


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('lungsimu.png')
```

Appendix L Simulation Program for Brain

```
import numpy as np
from matplotlib import pyplot as plt
from math import pi, exp, cos, sin, sqrt, atan2

ke = 200
ex = np.zeros(ke)
dx = np.zeros(ke)
ix = np.zeros(ke)
sx = np.zeros(ke)
hy = np.zeros(ke)
# creating the geometry
ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
number_of_frequencies = 9
# required input frequency
freq_in = np.array((10e6, 100e6, 200e6, 300e6, 400e6,
                    500e6, 600e6, 700e6, 800e6))
t0 = 50
spread = 10


# information about the dielectric mediums
epsz = 8.854e-12
epsr = 53.3                   # at 800MHz
sigma = 2.92E-1         # at 10MHz
tau = 0.001 * 1e-6
chi = 0
k_start = 100
```

```
# initializing boundary Conditions
boundary_low = [0, 0]
boundary_high = [0*i for i in range(2*int(sqrt(epsr)))]
# iterative variables
gax = np.ones(ke)
gbx = np.zeros(ke)
gcx = np.zeros(ke)

gax[k_start:] = 1 / (epsr + (sigma * dt / epsz) + chi * dt / tau)
gbx[k_start:] = sigma * dt / epsz
gcx[k_start:] = chi * dt / tau

del_exp = exp(-dt / tau)

# variables for Fourier transform:
arg = 2 * np.pi * freq_in * dt
real_pt = np.zeros((number_of_frequencies, ke))
imag_pt = np.zeros((number_of_frequencies, ke))
real_in = np.zeros(number_of_frequencies)
imag_in = np.zeros(number_of_frequencies)
amp_in = np.zeros(number_of_frequencies)
phase_in = np.zeros(number_of_frequencies)
amp = np.zeros((number_of_frequencies, ke))
phase = np.zeros((number_of_frequencies, ke))

# REF = np.zeros((number_of_frequencies,ke))

TRN = np.zeros((number_of_frequencies, ke))

nsteps = 1000

# Dictionary for the timestep
plotting_points = [{'num_steps': 350, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2, 'y_text_loc': 0.3,
                    'label': '(a)',
                    'label_loc': 1},
                   {'num_steps': 1000, 'ex': None, 'scaling_factor': 1,
                    'gb_scaling_factor': 1,
                    'y_ticks': (np.arange(-1, 1, step=0.5)),
                    'y_min': -1.3, 'y_max': 1.2,
                    'y_text_loc': 0.3, 'label': '(b)', 'label_loc': 1}]


# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate Dx in time domain
```

```
for k in range(1, ke):
    dx[k] = dx[k] + 0.5 * (hy[k - 1] - hy[k])

# Put a pulse wave in time domain
pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
dx[5] = pulse + dx[5]

# Calculate the Ex field from Dx in time domain
for k in range(1, ke):
    ex[k] = gax[k] * (dx[k] - ix[k] - del_exp * sx[k])
    ix[k] = ix[k] + gbx[k] * ex[k]
    sx[k] = del_exp * sx[k] + gcx[k] * ex[k]

# fourier transform of Ex
for k in range(ke):
    for m in range(number_of_frequencies):
        real_pt[m, k] = real_pt[m, k] + cos(arg[m] * time_step) * ex[k]
        imag_pt[m, k] = imag_pt[m, k] - sin(arg[m] * time_step) * ex[k]

# Fourier Transform of the input pulse
if time_step < (190):
    for m in range(number_of_frequencies):
        real_in[m] = real_in[m] + cos(arg[m] * time_step) * ex[10]
        imag_in[m] = imag_in[m] - sin(arg[m] * time_step) * ex[10]

# Boundary Conditions
ex[0] = boundary_low.pop(0)
boundary_low.append(ex[1])
ex[ke - 1] = boundary_high.pop(0)
boundary_high.append(ex[ke - 2])

#  Hy field calculations
for k in range(ke - 1):
    hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Saving the data plotting_points
for plotting_point in plotting_points:
    if time_step == plotting_point['num_steps']:
        plotting_point['ex'] = np.copy(ex)

# Calculate the amplitude and phase at each frequency
        for m in range(number_of_frequencies):
            amp_in[m] = sqrt(imag_in[m] ** 2 + real_in[m] ** 2)
            phase_in[m] = atan2(imag_in[m], real_in[m])
            for k in range(ke):
                amp[m, k] = (1 / amp_in[m]) * sqrt((real_pt[m, k]) ** 2
                                                   + imag_pt[m, k] ** 2)
                phase[m, k] = atan2(imag_pt[m, k],
                                    real_pt[m, k]) - phase_in[m]
```

```
                for k in range(100, ke):
                    TRN[m, k] = (amp[m, k])**2


fig = plt.figure(figsize=(8, 16))
# Dictionary for transmission
plotting_trns = [{'freq': freq_in[0], 'TRN': TRN[0, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[1], 'TRN': TRN[1, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[2], 'TRN': TRN[2, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[3], 'TRN': TRN[3, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[4], 'TRN': TRN[4, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[5], 'TRN': TRN[5, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[6], 'TRN': TRN[6, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[7], 'TRN':TRN[7, ],
                  'label': '', 'x_label': ''},
                 {'freq': freq_in[8], 'TRN': TRN[8, ],
                  'label': '', 'x_label': 'FDTD Cells'}]

# plotting transmission


def plot_trn(data1, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data1, color='k', linewidth=1)
    plt.ylabel('TRNS')
    plt.xticks(np.arange(100, 199, step=20))
    plt.xlim(100, 198)
    # plt.yticks(np.arange(0, 2.1))
    # plt.ylim(-0.2, 2.0)
    plt.text(170, 0.002, 'Freq. at {} MHz'.format(int(round(freq / 1e6))),
             horizontalalignment='center')
    # plt.plot(gb * 1 / gb[120], 'k--',linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num1, plotting_trn in enumerate(plotting_trns):
    ax = fig.add_subplot(9, 1, subplot_num1+1)
    plot_trn(plotting_trn['TRN'], plotting_trn['freq'],
             plotting_trn['label'], plotting_trn['x_label'])
```

```
plt.subplots_adjust(bottom=0.1, hspace=0.45)
# saving the figure
fig.savefig('braintrns.png')


plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 16))


def plot_e_field(data, gb, timestep, scaling_factor, gb_scaling_factor,
                 y_ticks, y_min, y_max, y_text_loc, label, label_loc):
    """Plot of E field at a single time step"""
    plt.plot(data * scaling_factor, color='k', linewidth=1)
    plt.ylabel('E$_x$(V/m)', fontsize='12')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(y_ticks)
    plt.ylim(y_min, y_max)
    plt.text(150, y_text_loc, 'Time Domain, T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot(gb * gb_scaling_factor / gb[120], 'k--', linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, label_loc, label, horizontalalignment='center')
    return


# Plotting e field
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(11, 1, subplot_num + 1)
    plot_e_field(plotting_point['ex'], gbx, plotting_point['num_steps'],
                 plotting_point['scaling_factor'],
                 plotting_point['gb_scaling_factor'],
                 plotting_point['y_ticks'],
                 plotting_point['y_min'],
                 plotting_point['y_max'], plotting_point['y_text_loc'],
                 plotting_point['label'],
                 plotting_point['label_loc'])

# Dictionary for the amplitudes
plotting_freqs = [{'freq': freq_in[0], 'amp': amp[0],
                   'label': '(c)', 'x_label': ''},
                  {'freq': freq_in[1], 'amp': amp[1],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[2], 'amp': amp[2],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[3], 'amp': amp[3],
                   'label': '', 'x_label': ''},
                  {'freq': freq_in[4], 'amp': amp[4],
```

```
                                 'label': '', 'x_label': ''},
                                {'freq': freq_in[5], 'amp': amp[5],
                                 'label': '', 'x_label': ''},
                                {'freq': freq_in[6], 'amp': amp[6],
                                 'label': '', 'x_label': ''},
                                {'freq': freq_in[7], 'amp': amp[7],
                                 'label': '', 'x_label': ''},
                                {'freq': freq_in[8], 'amp': amp[8],
                                 'label': '', 'x_label': 'FDTD Cells'}]


def plot_amp(data, gb, freq, label, x_label):
    """Plot of amplitude at one frequency"""
    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('Amp')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 198)
    plt.yticks(np.arange(0, 2.1, step=1))
    plt.ylim(-0.2, 2.0)
    plt.text(150, 1.2, 'Freq. Domain at {} MHz'.format(int(round(freq / 1e6)
             horizontalalignment='center')
    plt.plot(gb * 1 / gb[120], 'k--',
             linewidth=0.75)
    # The math on gb is just for scaling
    plt.text(-25, 0.6, label, horizontalalignment='center')
    plt.xlabel(x_label)
    return


for subplot_num, plotting_freq in enumerate(plotting_freqs):
    ax = fig.add_subplot(11, 1, subplot_num+3)
    plot_amp(plotting_freq['amp'], gbx, plotting_freq['freq'],
             plotting_freq['label'], plotting_freq['x_label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()
fig.savefig('brainsimu.png')
```

Appendix M Disclaimer

Don’t use jupyter notebook for animation it may not show any output. please uncomment the last line to save the file and analyse it

Appendix N Animation Using Soft Source

```
import numpy as Np
import matplotlib
import matplotlib.pyplot as plt
```

```
from matplotlib import animation
# from math import Exp
matplotlib.use('TkAgg')
# please see the first program to understand it better

cells = 200
Ex = Np.zeros(cells)
Hy = Np.zeros(cells)
x = Np.arange(200)
boundary_low1 = [0, 0]
boundary_high1 = [0, 0]
boundary_low2 = [0, 0]
boundary_high2 = [0, 0]
# pulse parameters
kc = 5
t0 = 40.0
width = 12
nsteps = 300


def init():
    line.set_data([], [])
    line1.set_data([], [])
    return line,


# main FDTD Loop
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 3.5))
axis = fig.add_subplot(211, xlim=(0, 200), xticks=(Np.arange(0, 201, step=20
                       ylim=(-1.2, 1.2), yticks=(Np.arange(-1, 1.2, step=1))
                       xlabel='cells', ylabel='$E_{x}$')
axis1 = fig.add_subplot(212, xlim=(0, 200), xticks=(Np.arange(0, 201, step=2
                        ylim=(-1.2, 1.2), yticks=(Np.arange(-1, 1.2, step=1)
                        xlabel='cells', ylabel='$H_{y}$')
line, = axis.plot([], [], color='b', lw=1)
line1, = axis1.plot([], [], color='c', lw=1)


def animation_frame(i):
    global cells, Ex, Hy, kc, t0, nsteps, width, x
    boundary_low1 = [0, 0]
    boundary_high1 = [0, 0]
    boundary_low2 = [0, 0]
    boundary_high2 = [0, 0]
    Ex = Np.zeros(cells)
    Hy = Np.zeros(cells)
    for time_step in range(1, i+1):
        # x = Np.zeros(cells)
```

```
        # calculate the Hy field
        for k in range(1, cells):
            Hy[k] = Hy[k] + 0.5*(Ex[k-1] - Ex[k])

        pulse = Np.Exp(-0.5 * ((t0 - time_step) / width) ** 2)
        Hy[kc] = pulse + Hy[kc]

        Hy[0] = boundary_low2.pop(0)
        boundary_low2.append(Hy[1])

        Hy[cells-1] = boundary_high2.pop(0)
        boundary_high2.append(Hy[cells - 2])

        for k in range(cells-1):
            Ex[k] = Ex[k] + 0.5*(Hy[k] - Hy[k+1])

        Ex[0] = boundary_low1.pop(0)
        boundary_low1.append(Ex[1])

        Ex[cells-1] = boundary_high1.pop(0)
        boundary_high1.append(Ex[cells - 2])

    for i in range(0, kc):
        Ex[i] = 0
        Hy[i] = 0
    # y = Ex
    line.set_data(x, Ex)
    line1.set_data(x, Hy)
    return line, line1


# plt.tExt(100, 0.5, 'T = {}'.format(time_step), horizontalalignment='center
anim = animation.FuncAnimation(fig, func=animation_frame,
                               init_func=init,
                               frames=800,
                               interval=100,
                               blit=True)

# plt.show()
anim.save('softsource.mp4', writer='ffmpeg', fps=30)
```

Appendix O Animation using soft source from air to dielectric medium of refractive index 2

```
import numpy as Np
import matplotlib
import matplotlib.pyplot as plt
from matplotlib import animation
```

```
from math import sin, pi, Exp
matplotlib.use('TkAgg')


cells = 200
Ex = Np.zeros(cells)
Hy = Np.zeros(cells)
x = Np.arange(200)

boundary_low1 = [0, 0]
boundary_high1 = [0, 0, 0, 0]
boundary_low2 = [0, 0]
boundary_high2 = [0, 0, 0, 0]

# pulse parameters
kc = 80
t0 = 40.0
width = 12

cb = Np.ones(cells)
cb = 0.5 * cb
cell_start = 100
epsilon = 4
cb[cell_start:] = 0.5 / epsilon

ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
freq_in = 700e6


def init():
    line.set_data([], [])
    time_tExt.set_tExt('')
    return line, time_tExt


# main FDTD Loop
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 4.5))
# fig = plt.figure((0.5 / cb - 1) / 3, 'k--', linewidth=0.75)
axis = fig.add_subplot(xlim=(0, 199),
                       xticks=(Np.arange(0, 200, step=20)),
                       ylim=(-1.5, 1.5),
                       yticks=(Np.arange(-1.5, 1.5, step=0.5)),
                       xlabel='cells', ylabel='$E_{x}$')
time_tExt = axis.tExt(0.55, 0.95, '', horizontalalignment='center',
                      verticalalignment='top', transform=axis.transAxes)

line, = axis.plot([], [], color='b', lw=1)
```

```
line1, = axis.plot((0.5 / cb - 1) / 3, 'k--', linewidth=0.75)


def animation_frame(i):
    global cells, Ex, Hy, kc, t0, width, x, cb, cell_start, epsilon, co

    boundary_low1 = [0, 0]
    boundary_high1 = [0, 0, 0, 0]
    boundary_low2 = [0, 0]
    boundary_high2 = [0, 0, 0, 0]
    # pulse parameters
    kc = 5
    t0 = 40.0
    width = 12

    cb = Np.ones(cells)
    cb = 0.5 * cb
    cell_start = 100
    epsilon = 4
    cb[cell_start:] = 0.5 / epsilon

    Ex = Np.zeros(cells)
    Hy = Np.zeros(cells)

    for time_step in range(1, i+1):
        # x = Np.zeros(cells)
        # calculate the Hy field
        for k in range(1, cells):
            Ex[k] = Ex[k] + cb[k]*(Hy[k-1] - Hy[k])

        Ex[0] = boundary_low1.pop(0)
        boundary_low1.append(Ex[1])

        Ex[cells-1] = boundary_high1.pop(0)
        boundary_high1.append(Ex[cells - 2])

        for k in range(cells-1):
            Hy[k] = Hy[k] + 0.5*(Ex[k] - Ex[k+1])

        pulse = Exp(-0.5 * ((t0 - time_step) / width) ** 2)
        Hy[kc] = pulse + Hy[kc]

        Hy[0] = boundary_low2.pop(0)
        boundary_low2.append(Hy[1])

        Hy[cells-1] = boundary_high2.pop(0)
        boundary_high2.append(Hy[cells - 2])

    if(i < cell_start):
```

```
        for i in range(0, kc):
            Ex[i] = 0
            Hy[i] = 0
    time_tExt.set_tExt('Time_steps = %.1d' % i)
    # y = Ex
    line.set_data(x, Ex)
    return line,


# plt.tExt(100, 0.5, 'T = {}'.format(time_step), horizontalalignment='center
anim = animation.FuncAnimation(fig, func=animation_frame,
                               init_func=init,
                               frames=800,
                               interval=10,
                               blit=True)

plt.show()
# anim.save('anim0.gif', writer='PillowWriter', fps=30)
```

Output as Figures

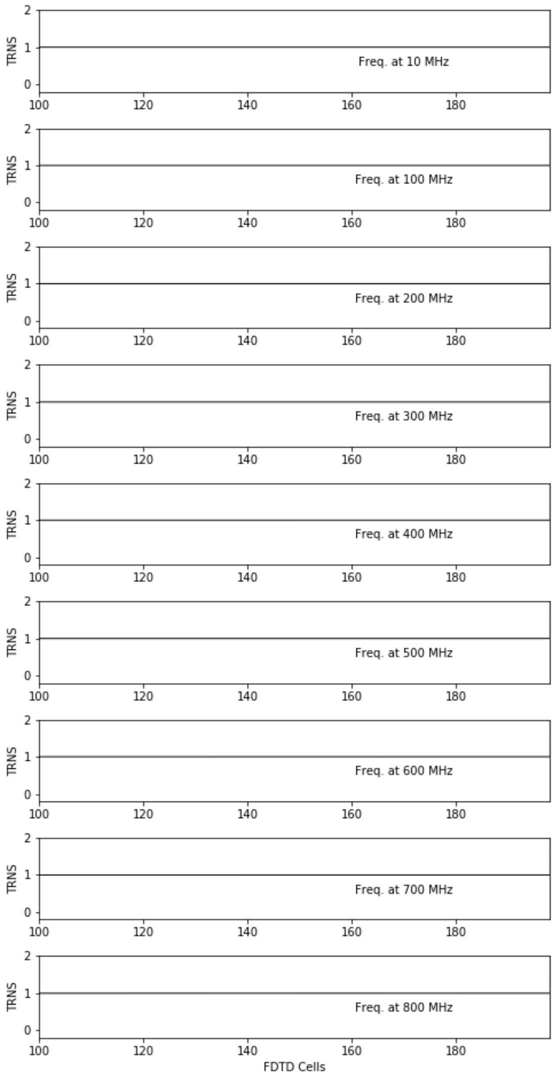


Figure 7: Transmission in Air at different frequencies

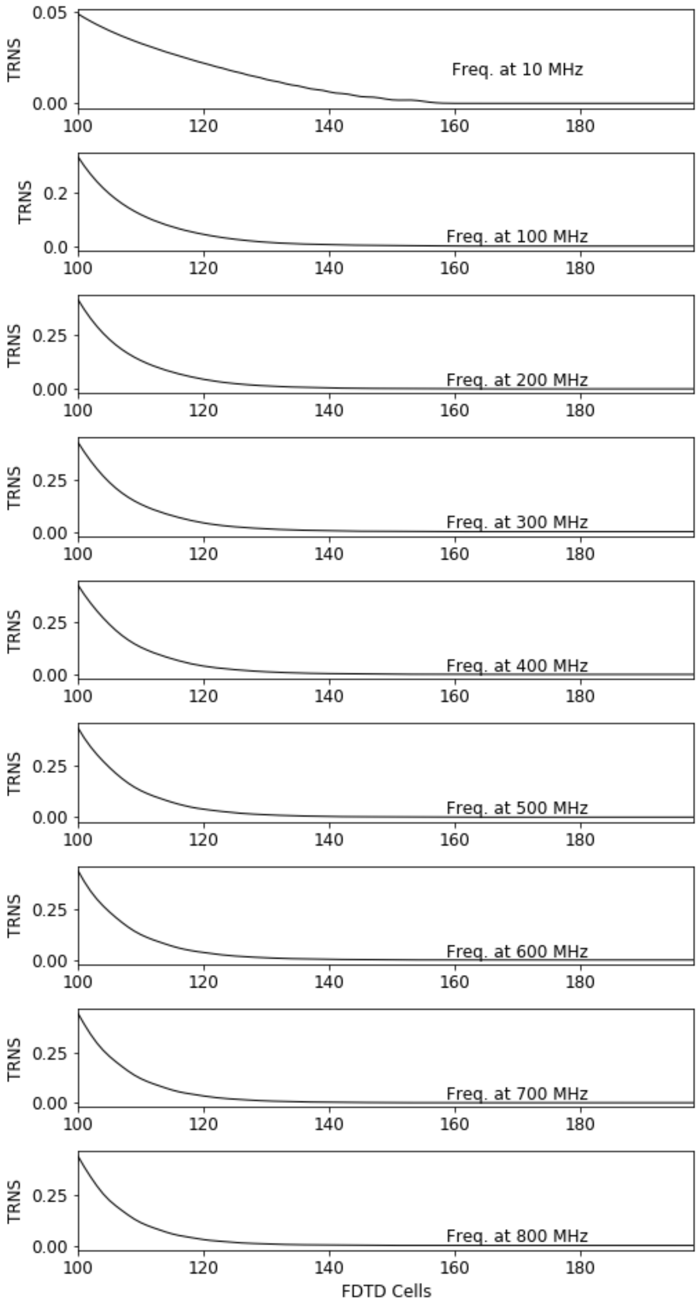


Figure 8: Transmission in Skin at different frequencies

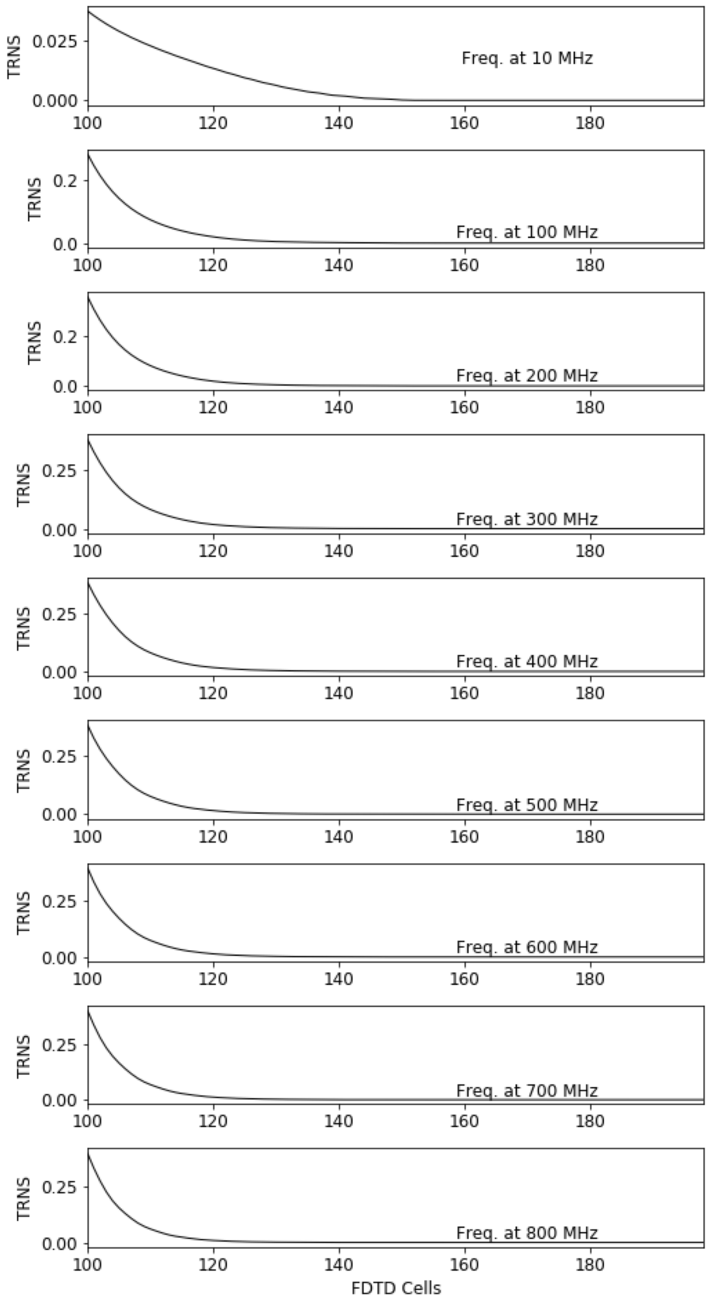


Figure 9: Transmission in Fat at different frequencies

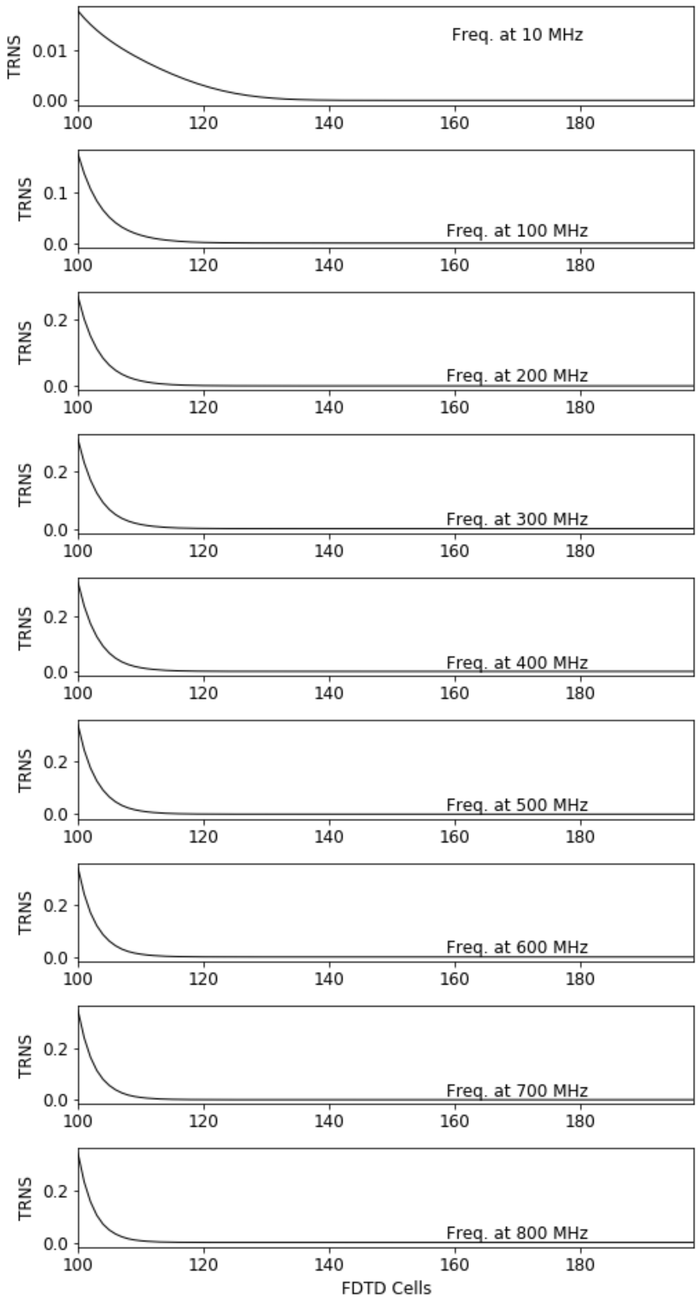


Figure 10: Transmission in Muscle at different frequencies

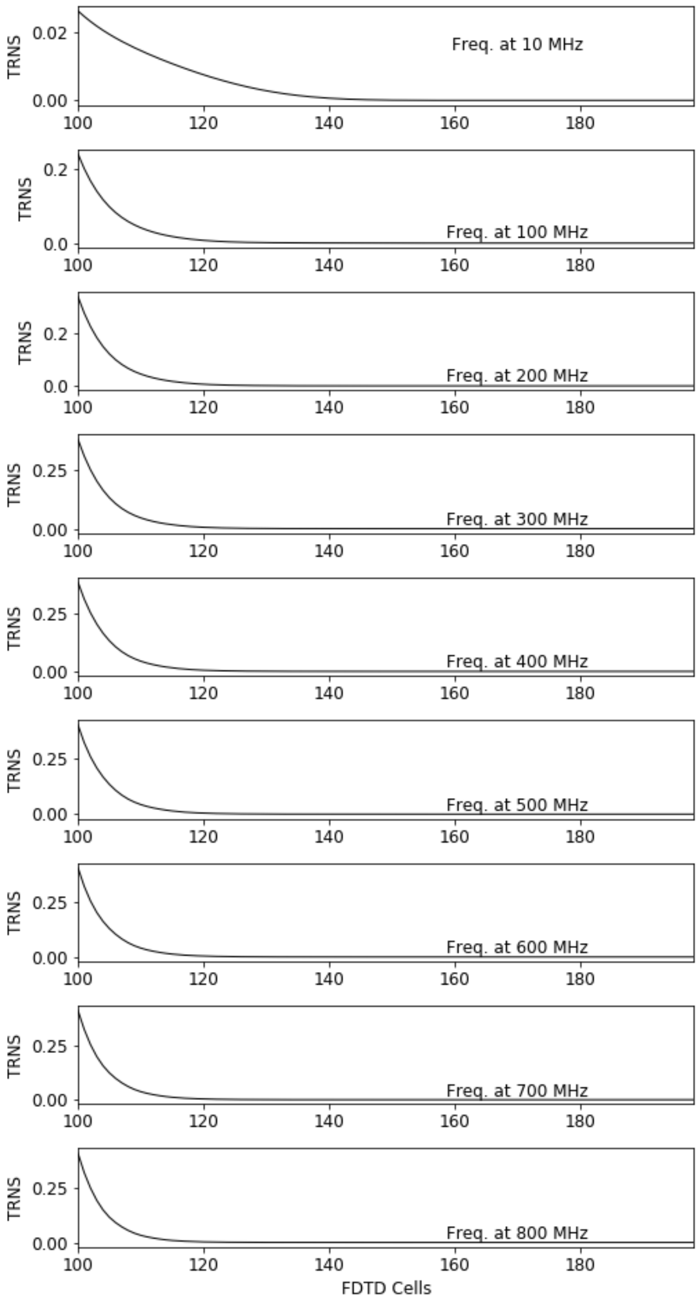


Figure 11: Transmission in Cartilage at different frequencies

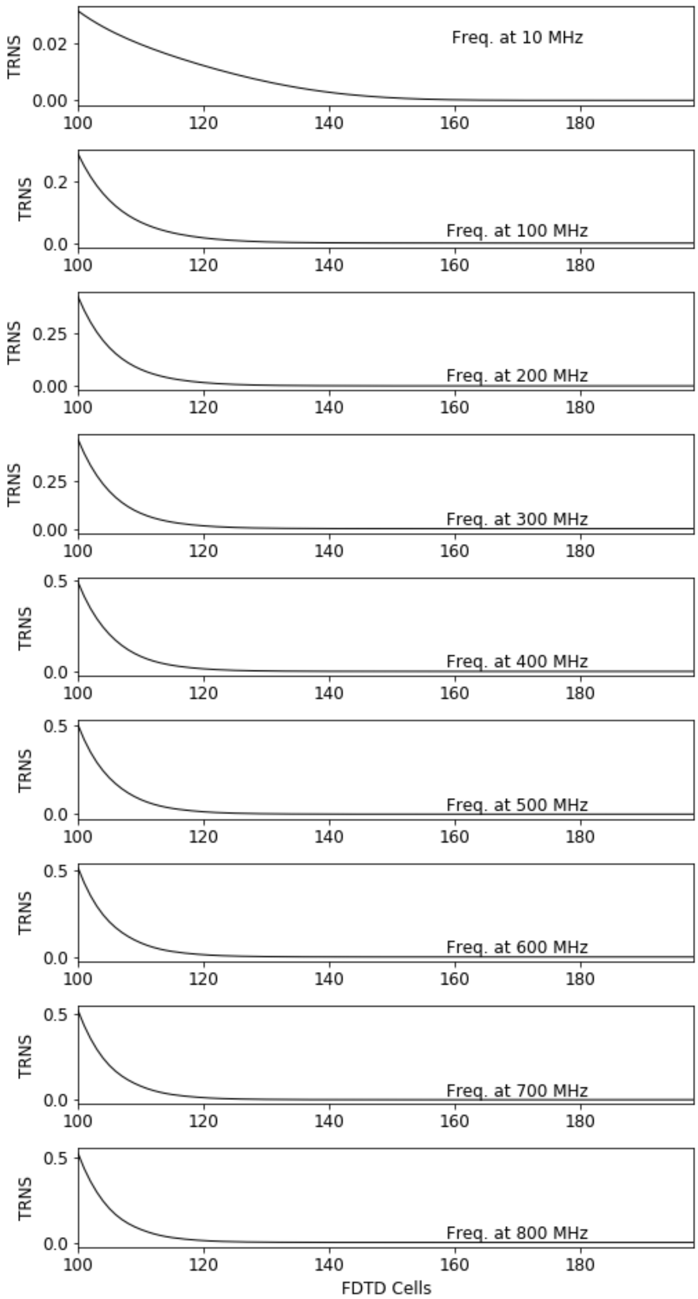


Figure 12: Transmission in Lung at different frequencies

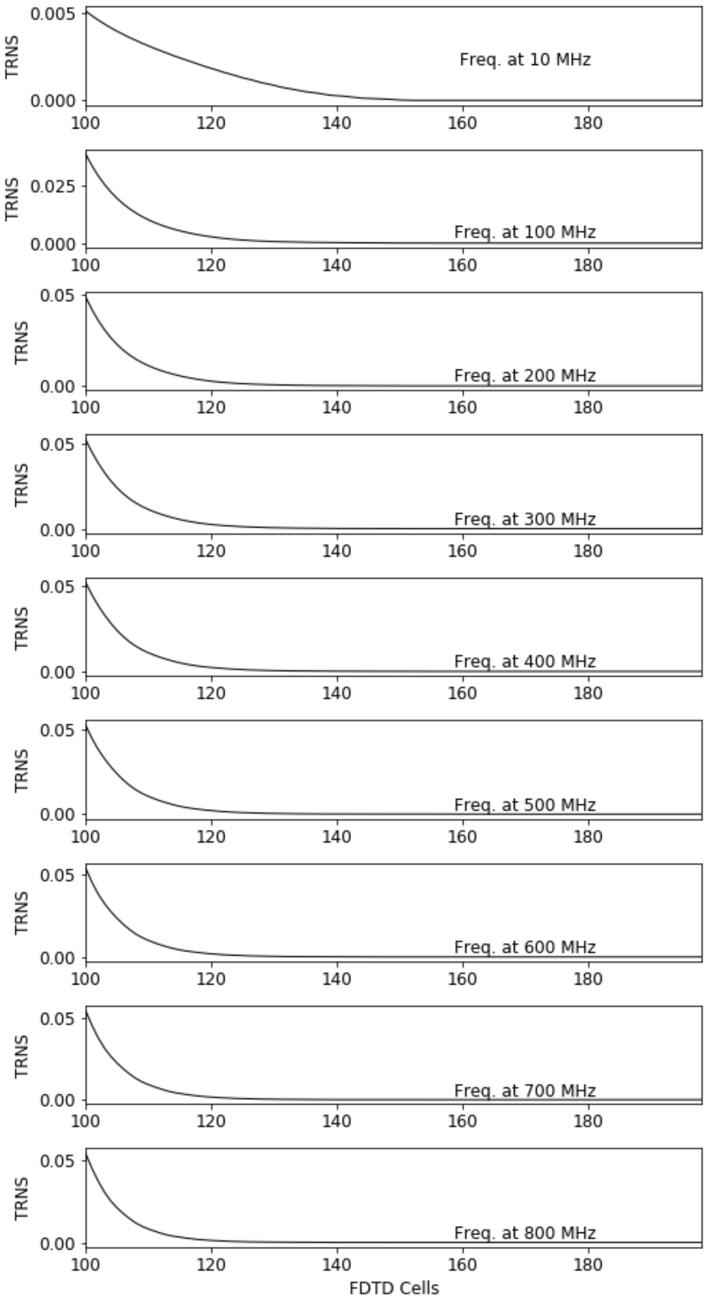


Figure 13: Transmission in Brain (GM) at different frequencies

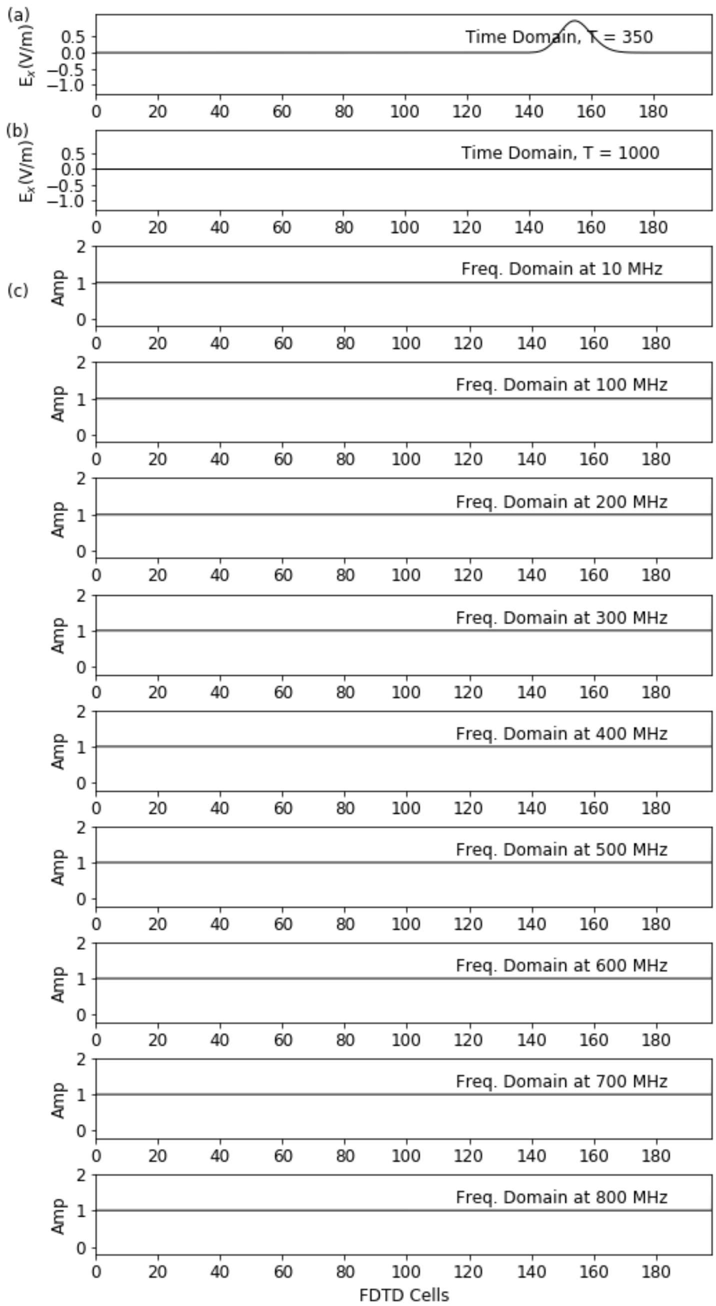


Figure 14: Air-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

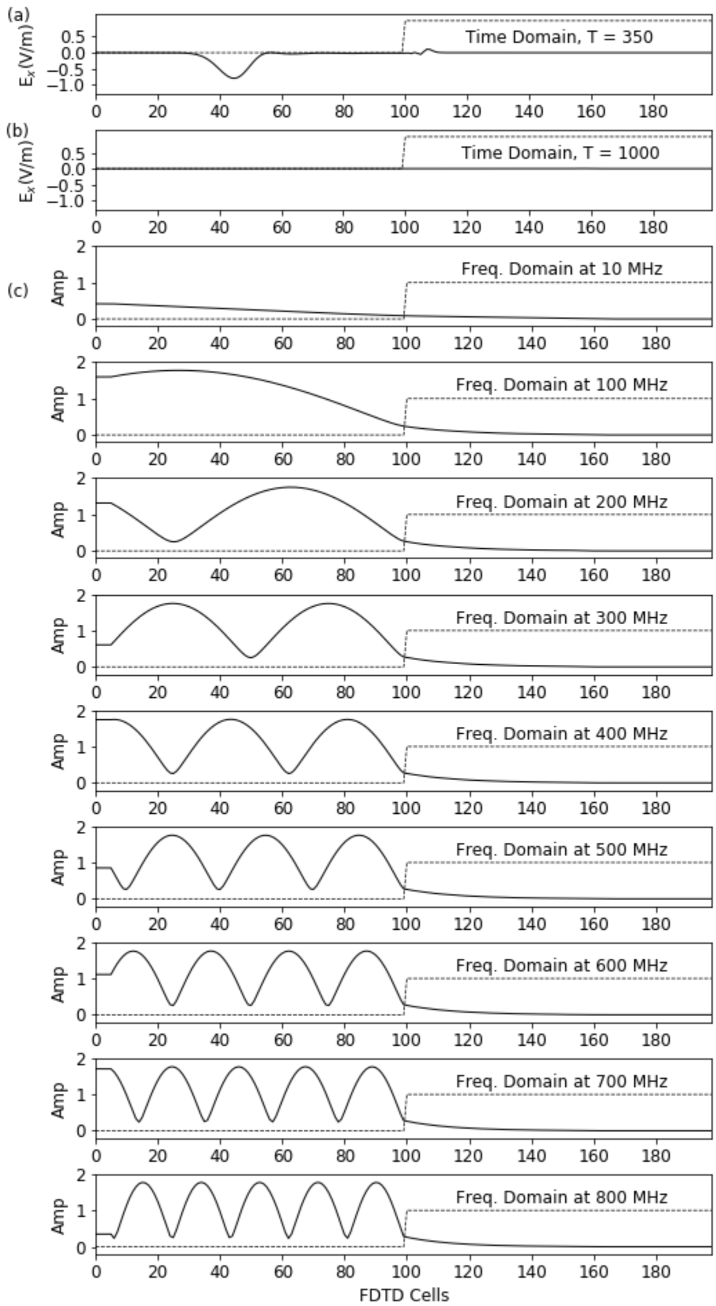


Figure 15: Skin-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

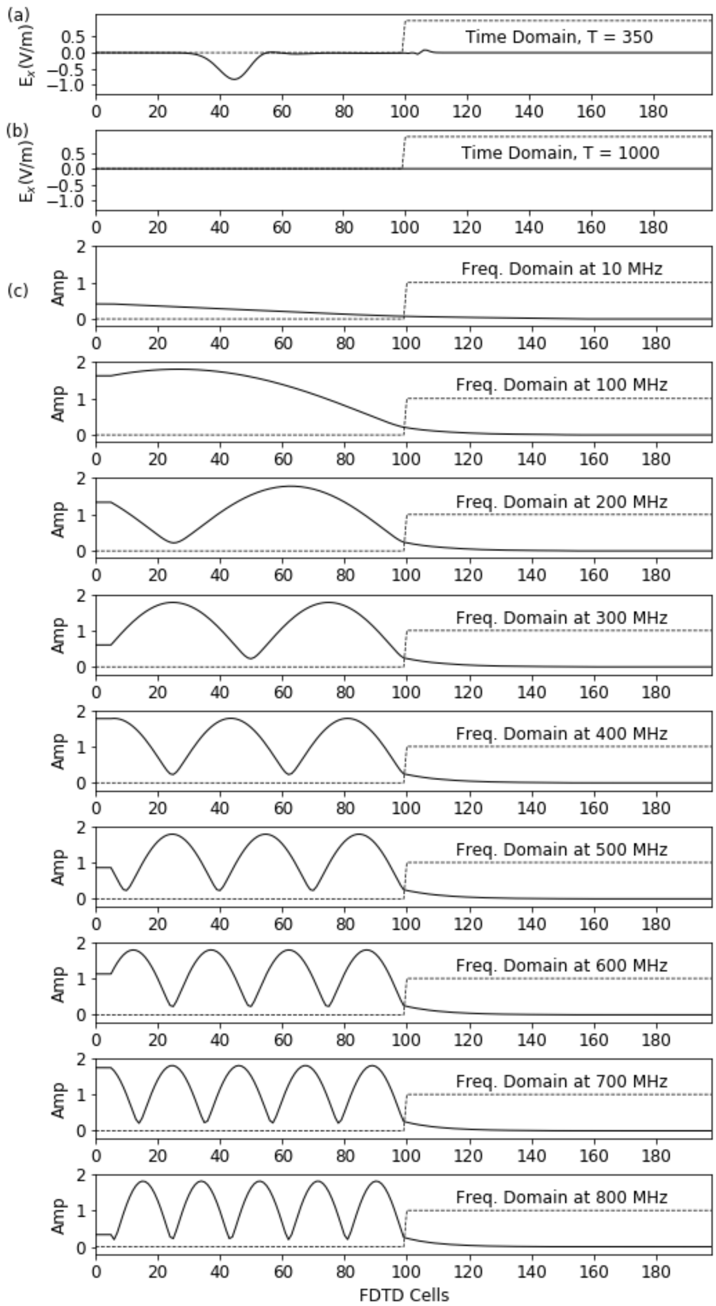


Figure 16: Fat-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

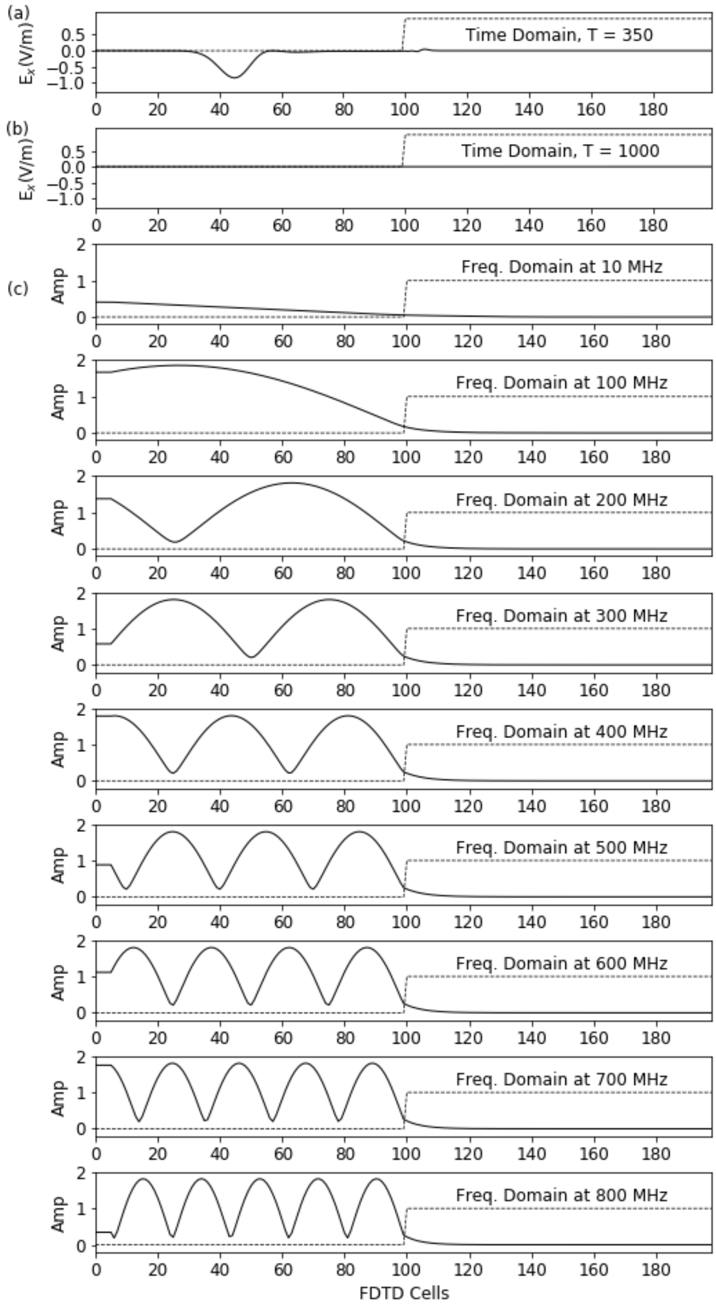


Figure 17: Muscle-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

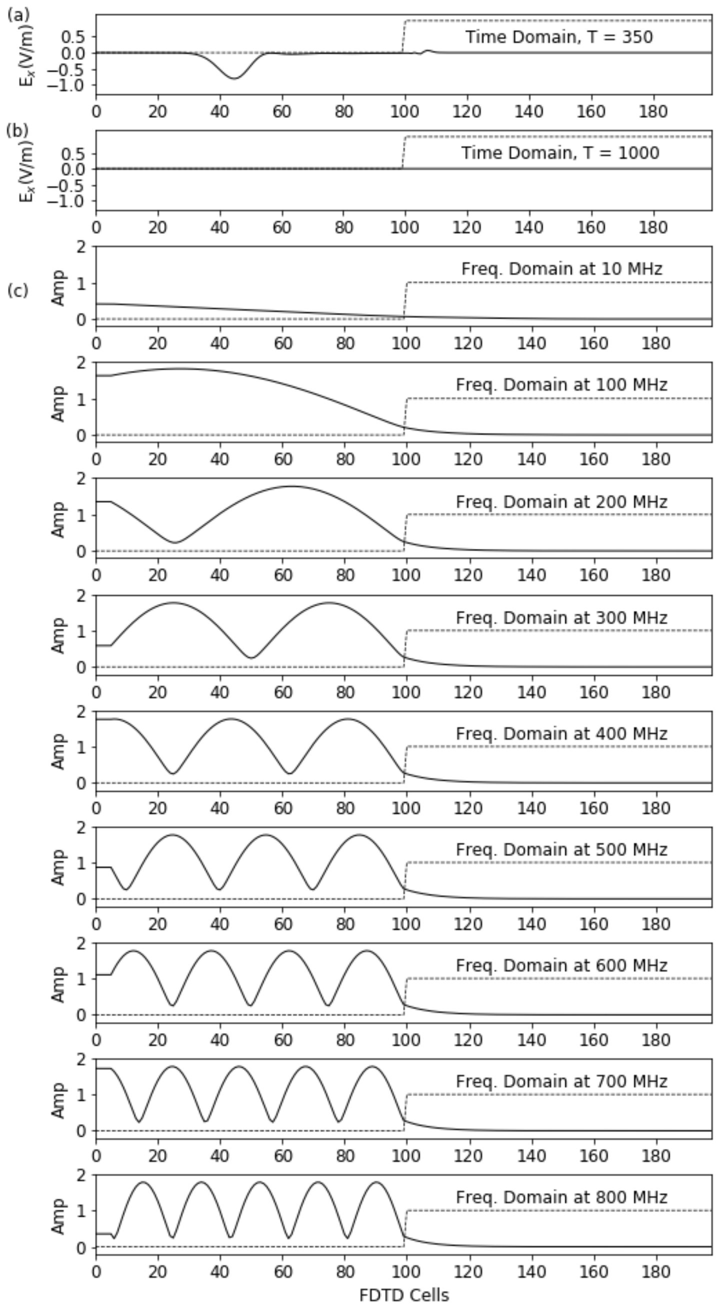


Figure 18: Cartilage-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

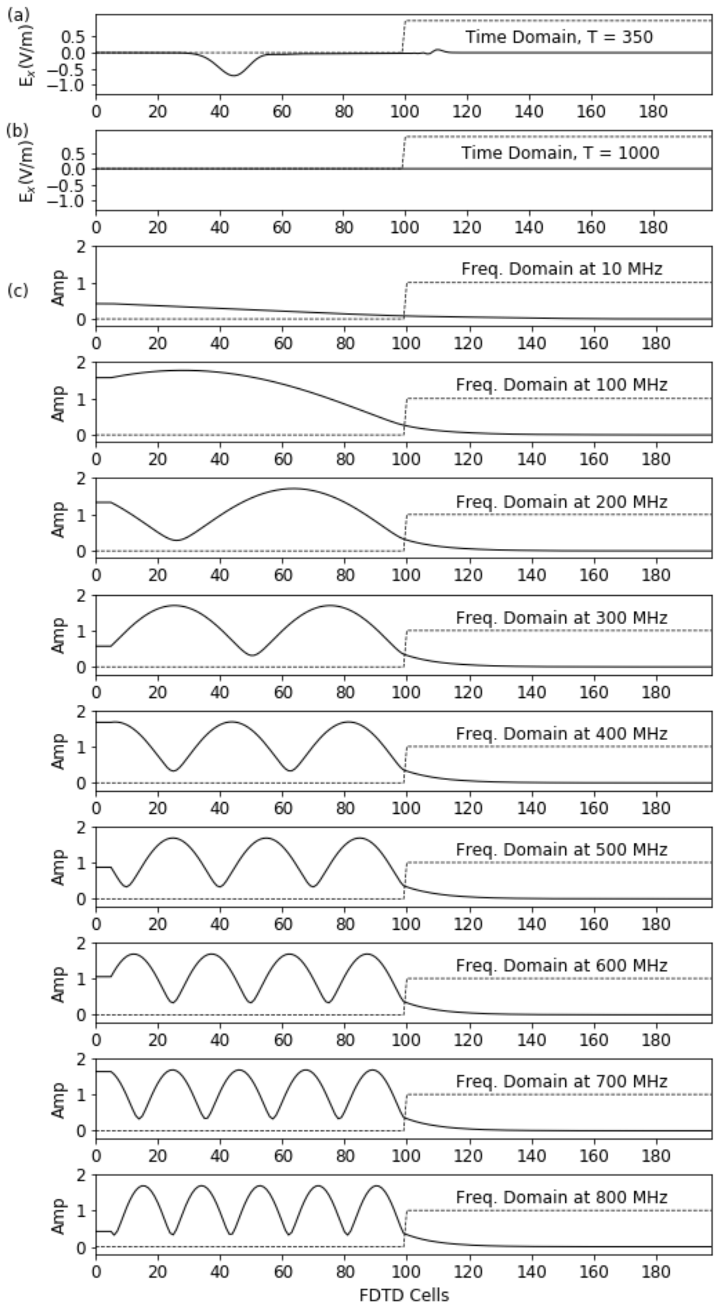


Figure 19: Lung-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies

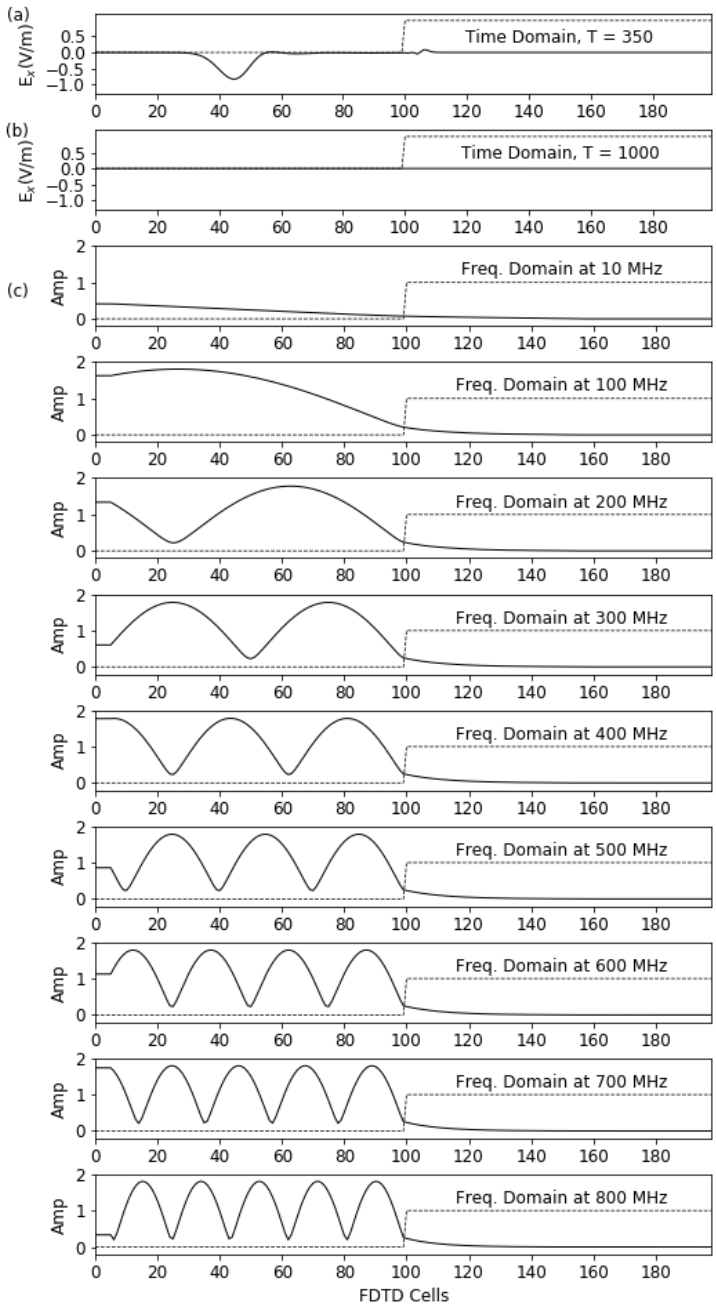


Figure 20: Brain (GM)-In the graph fig(a) and fig(b) represents pulse wave of E-field in time step 350 and 1000 respectively and fig(c) represents simulation of E-field in the frequency domain and given in the different frequencies