Neural Network Solutions to Cylindrical Microwave Cavities

Nathan Schwartz

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Applied Physics

John Colton, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Neural Network Solutions to Cylindrical Microwave Cavities

Nathan Schwartz
Department of Physics and Astronomy, BYU
Bachelor of Science

A microwave cavity, also known as a radio frequency cavity, is a specific type of resonator. Typical cavities consist of a closed metal structure that acts as a truncated waveguide for electromagnetic fields in the microwave spectrum. Technological devices that utilize microwave cavities are widespread and include filters, amplifiers, and oscillators. However, the computational resource cost and time expense required to find solutions to these cavities are prohibitive. Thus, a set of neural networks has been developed to find the frequency and mode characteristics of any given cavity configuration in ~0.05 seconds with a mean absolute error of 0.012 for a normalized frequency range of 0.143-1.949, a cumulative mean absolute error of 0.0022 for 200 coefficients, and an accuracy of 95.46% for mode predictions. The creation, refinement, and final outputs of the neural networks are discussed.

Keywords: Machine Learning, Neural Network, Microwave Cavity

ACKNOWLEDGEMENTS

I would like to acknowledge my professors who have cultivated my curiosity and given me the tools and skills necessary to complete this project. I would like to especially thank Dr. John Colton for advising me on this project. His expertise, good-natured humor, and quick thinking made this project run smoothly from day one. I would also like to thank Charles Lewis and Jordan Bryan for their contributions towards creating the TE_{011} solver and the neural networks, respectively. Lastly, I would like to thank Kane Fanning and Jonathan Hale for their work on the neural network refinement.

Contents

Chapter 1

Introduction

1.1 Microwave Cavities

Cavity resonators are metal-bound structures which closely relate to a truncated waveguide. Waveguides transmit electromagnetic (EM) waves, whereas cavity resonators do not. Microwave cavities, also known as radio frequency cavities, are a specific type of resonator made of a closed (or mostly closed) metal structure. This structure contains dielectric material and confines EM fields within the microwave spectrum. When electromagnetic waves of resonant frequency are introduced into the cavity, they reinforce to become standing waves.[1] Multiple resonant frequencies exist for a cavity of any given dimension. Each resonant frequency will have a corresponding EM field mode, one of which is the TE_{011} mode (given the cavity is cylindrically symmetric).

1.2 TE_{011} Mode

The TE_{011} mode is relevant for its applications in resonance, specifically as it relates to the quality factor (Q factor) and the power to field conversion efficiency. The resonance associated with the TE_{011} mode is quite stable, and therefore allows for a consistent Q factor and conversion efficiency.

The natural resonance frequency of a TE_{011} mode in a cylindrical cavity of radius r and length l is given by Equation 1, [2]

$$f = \frac{c}{2}\sqrt{\left(\frac{x'_{01}}{\pi r}\right) + \frac{1}{l^2}} \tag{1}$$

where c is the speed of light in vacuum and $x'_{01}$is the first nonzero root of the Bessel function derivative, $J'_0(x)$.

The TE_{011} mode can be described as follows: The electric field is only in the phi direction and has a node at the center of the cavity. The magnetic field is only in the z direction. The first zero means there is no phi dependence, the first 1 means there is an antinode node radially in the electric field, and the second 1 means there is an antinode in the z direction in the magnetic field.

Throughout this paper, references will be made to the quasi-TE_{011} mode. This mode results from deviations from a standard cylindrical cavity either by including dielectrics or omitting the metal shell. The quasi-TE011 mode is the mode that qualitatively looks like the TE011 of a regular cylindrical cavity. We could have had the network represent the field directly, but to save computational resources we chose to represent the mode in terms of the empty cavity mode. This is why we output 200 empty cavity expansion coefficients as described in Section 2.2.1.

1.3 Prior Work at BYU

Computational work for solving resonance cavities has been done by Kyle Miller et al. [3] Miller used MATLAB to find and solve for the frequency and modes for any given cavity configuration. As explained in Miller's paper, 10 different configurations were solved with computations taking anywhere from two to six hours. All solutions were then compared to a physical model, and yielded accuracies within 1% save for one configuration. While Miller's work is a step forward towards fast and computationally inexpensive solutions to the resonance cavity problem, his work can be improved upon through machine learning.

1.4 Machine Learning

Machine learning is a specific set of algorithms contained in the field of artificial intelligence. Machine learning uses statistics to identify trends and to make predictions based on the data fed into the algorithms. In essence, machine-learning algorithms take in data, whether that be in the form of pictures, sounds, or website activity, and use that data to make an educated guess on what behavior can be expected in the future. For example, Facebook uses machine learning algorithms to take in data about a user's age, gender, and consumer preferences to show ads that other users with a similar profile have found to be useful. [4]

Within machine learning lies the class of artificial intelligence called deep learning, which relies on a neural network to make predictions. Neural networks are a concept loosely based on the workings of the human brain. The networks are composed of multiple layers of nodes, akin to neurons, which act together to detect and enlarge small patterns. [5] This pattern identification is

what enables the neural network to make accurate guesses about the implications of other data points.

Machine learning algorithms could offer a distinct advantage to Miller's aforementioned approach. Instead of solving for the TE_{011} mode in a given cavity through a long and computationally expensive process, a neural network could be trained using tens of thousands of cavity configuration data points. This neural network would then be able to make an accurate prediction of any cylindrical cavity's resonant characteristics by relying on previous training. Although the upfront time cost to create and train the neural network is high, the final computation process would take mere seconds and a much smaller amount of computer resources.

Chapter 2

Experimental Methods

2.1 The Initial Neural Network

As mentioned in Section 1.4, an artificial neural network (ANN) is a subclass of machine learning that imitates the function of a human brain. The neurons in this analogy, referred to as nodes, are contained in individual layers and interconnected in series. The first layer, known as the input layer, receives data that is then fed into the next layer of nodes. Each node receives a weighted combination of the outputs of the previous layer, applies an "activation function", and then passes its output to the next layer. After the data passes through each inner layer, or hidden layer, the final output is passed to an output layer. [6] See Figure 2.1 for reference.

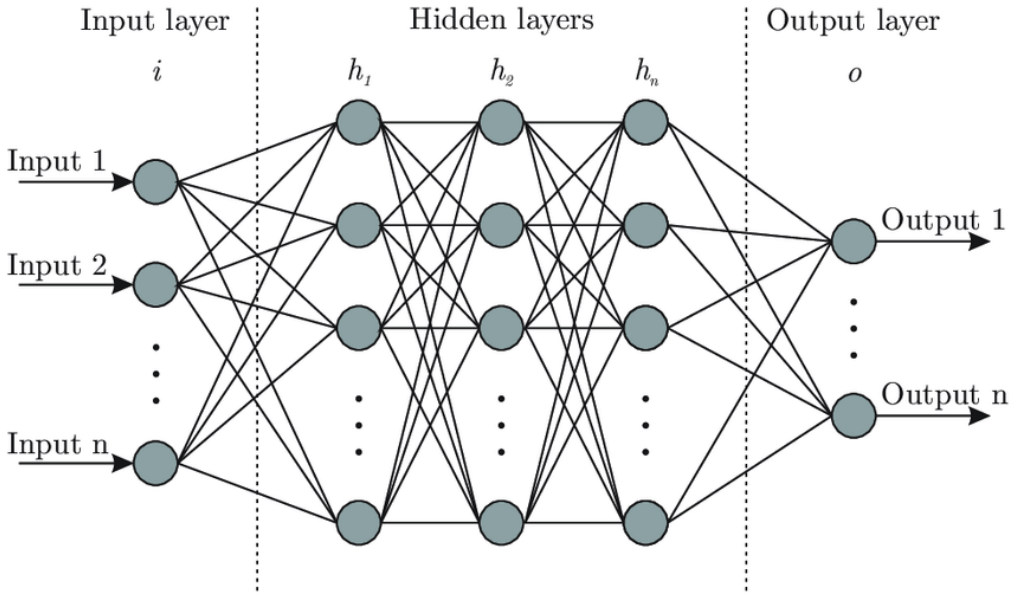


Figure 2.1 The neural network receives inputs, passes those inputs through a predetermined set of hidden layers, then outputs the results. The data passed between each layer is a weighted combination of the outputs of the previous layer after having applied a nonlinear activation function. Figure from [7].

In our case, artificial neural networks were employed to take in six cavity configuration parameters: aspect ratio (AR), dielectric inner radius (IR), dielectric outer radius (OR), dielectric height (DRH), dielectric separation distance (SEP), and the value of the dielectric constant (DR) as visualized in Figure 2.2.

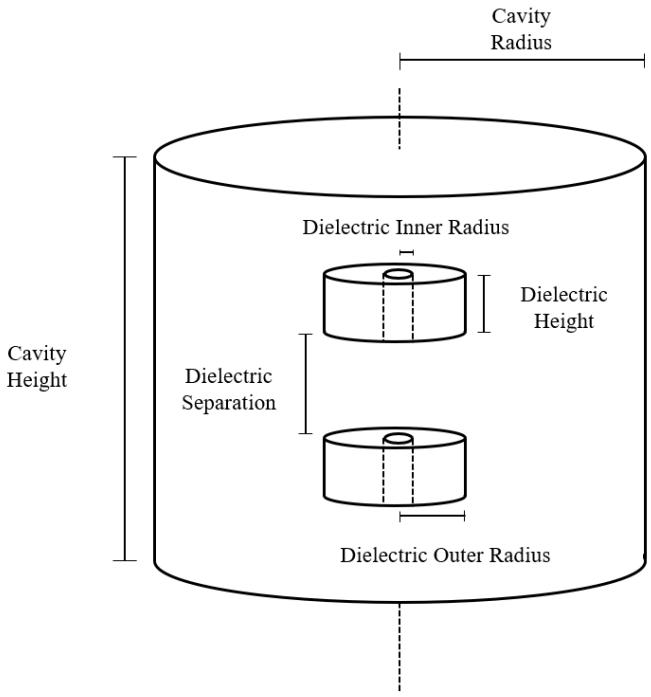


Figure 2.2 The cavity configurations can be described by the aspect ratio, dielectric inner radius, dielectric outer radius, the dielectric height, the dielectric separation, and the value of the dielectric constant. The aspect ratio is obtained using the cavity height and diameter as in Equation 2.

Cavity radius, height, and aspect ratio are related through a term known as the harmonic mean (HM). Our experiment set the harmonic mean to be one, since two cavities which differ only by a scaling factor have the same resonant modes with resonant frequencies differing only by the ratio of the two sizes. Normalizing our cavities in this way allows us to streamline our computational work in training and testing the neural networks, or alternatively to get more accuracy for the same amount of computational work.

The relationship between AR and HM and can be derived as follows:

$$Aspect\ Ratio = \frac{Height}{Diameter} \tag{2}$$

$$Harmonic\ Mean = \frac{2*Diameter*Height}{Height+Diameter} \tag{3}$$

where diameter is twice the cavity radius.

From Equations 2 and 3, we can derive the height and diameter of the cavity.

$$Height = \frac{(1+AR)*HM}{2} \tag{4}$$

$$Diameter = \frac{(1 + AR) * HM}{2 * AR} \tag{5}$$

Using approximately 100,000 parameters generated via Latin Hypercube sampling and their respective solutions, the machine was trained to solve for the TE_{011} mode. The machine's learning was then tested using approximately 10,000 points of holdout data. Latin Hypercube sampling is explained in further detail in Section 2.1.2.

2.1.1 Configurations

Our dielectric resonator consisted of a cylindrically symmetric cavity containing two dielectrics who were centered on the axis of the vertical axis of the cavity. The dielectrics were separated by a distance of SEP. These identical dielectrics are characterized by the same dielectric constant, which was used in conjunction with the 5 other parameters mentioned in the previous section to describe the geometry of our cavity. Alternatively, instead of using the aspect ratio, we could have used the cavity radius and height, but since HM was always set to one, we could convey the same information using one parameter instead of two. Since we intend for the final neural networks to provide quick and accurate results of the quasi-TE_{011} mode and its resonant frequency in real world experiments, we implemented two weighting functions similar to the remapping described in Section 2.2.4, so that the dielectric constants were more likely to take higher values, and the aspect ratio was equally distributed on both sides of 1.

Certain measures were taken to ensure all generated configurations would have valid geometries of sufficient resolution for the frequency and mode calculations. I.e., the outer radius had to be larger than the inner radius, the heights of the two dielectrics combined with their separation distance couldn't exceed the total cavity height, and the dielectric had to have sufficiently large dimensions for our code's resolution.

For each of the cavities, AR was set by random number selection between 0 and 1 as explained in Section 2.1.2, then rescaled to be between 0.25 and 4 using a logarithmic scale. For a given choice of AR, the cavity height and cavity radius were determined via Equations 4 and 5 with HM=1. IR and OR were determined by linearly rescaling random numbers between 0 and 1 to be between 0 and the cavity radius, then swapping if IR was greater than OR. The difference

between the OR and IR was divided by the max of the cavity radius and the cavity height, and the configuration was removed if the resulting ratio was less than 0.01. The heights of the dielectrics were linearly mapped between zero and the cavity height divided by two. The dielectric height value was compared to the max of the cavity radius and the cavity height, and the configuration was removed if the ratio was less than 0.01. The separation between the dielectrics was mapped from zero to the cavity height minus the sum of the dielectric heights, thus ensuring the dielectrics never extended beyond the roof and floor of the cavity. Finally, the dielectric constant was scaled from 1 to 45.

2.1.2 Latin Hypercube Sampling

Latin Hypercube sampling (LHS) is a method of generating random parameter values. LHS relies on the concept of a two-dimensional Latin Square design, where there is only one sample in each row and each column. Any square of three dimensions or higher is known as a Latin Hypercube.

LHS is a form of stratified sampling that ensures near-random sampling of a multivariable distribution. First, the distribution is divided into N equally sized intervals. Then N sample points are placed such that the requirements of the Latin square design are satisfied. This placement forms N equal divisions for each variable. This process is repeated for every variable, thus ensuring there is only one sample point for either one row or column of every variable. This interval assignment results in a Latin Hypercube of dimensions N^n, where n is the total number of parameters in the distribution and N is the total number of sample points. This method covers the entire multiparameter distribution, but still maintains the randomness required for machine learning datasets.[8]

2.1.3 TE_{011} Solutions Using the BYU Supercomputer

To properly train the neural network, we needed to generate solutions for the ~100,000 resonant cavities training data. These solutions were calculated using FEniCS, an open-source package for solving partial differential equations through finite element programming in Python. [9] We used FEniCS to define a finite element representation of the two-dimensional cross section of a generalized cylindrical cavity and then numerically approximated the resonant frequency and electric field of the TE_{011} mode.

The finite element method (FEM) numerically approximates an unknown function by expressing a partial differential equation as a variational problem. FEM creates the variational problem by multiplying the unknown function by a test function, integrating the resulting equation over the domain, and performing integration by parts of terms with second-order derivatives.[9]

The FEniCS code to solve the TE_{011} mode was written by my collaborator Charles Lewis, who describes the details as follows[1]:

> To solve for the electric field and resonant frequency of the quasi-TE_{011}, the defined dimensions of the cavity were used to create a rectangular 2D mesh of the cylindrical cavity mapping the height versus the radius with the FEniCS-related packages "mshr" and "dolfin". The dielectrics were mapped with their own 2D rectangular meshes within the mesh of the cavity. The appropriate value for the relative permittivity of each mesh domain was assigned and Dirichlet boundary conditions were applied so that the central axis and conductive boundary of the cavity would ensure a value of 0 for the electric field. This 2D problem is valid for resonant modes where m equals 0 since both the electric and magnetic fields will be uniform in the phi-direction.
>
> Once the problem has been setup, the program can solve for the electric field and the resonant frequency using the finite element method. However, because there are infinite solutions, the program requires an estimated target frequency. The resonant frequency of the DRs corresponding empty cavity is chosen as the starting target, since the quasi-TE_{011} for a cavity with dielectrics will have a frequency that is always greater than the resonant

[1] Personal communication reflects updated practice

frequency for the TE_{011} mode of the corresponding empty cavity. FEniCS will attempt to solve for a resonant mode in the vicinity of that frequency and if none are found then the target frequency will be increased iteratively at a higher and higher rate until it finds a resonant mode, or until a time limit imposed by the code is reached. If the limit is exceeded, the code outputs "TE_{011} mode not found". Once a resonant mode is found, the increment value of the frequency is reset and the program tests whether the mode is quasi-TE_{011} by first approximating the electric field by sampling a 50 by 50 grid of the field.

The mode is determined to be the quasi-TE_{011} mode by two separate checks. The first is to check if the resonant mode is a TE or TM mode by comparing the sum of the electric field at all points in the phi direction with the r and z directions. If the sum of the field at all points in the phi direction is greater than in the other directions together, then the resonant mode is a TE mode. The second check was done by taking the curl of the electric field to get the field pattern of the magnetic field. If the z-component of the magnetic field always points in the same direction and never flips over the 50 by 50 grid, then this resonant mode was successfully determined to be the quasi-TE_{011} mode.

After this code was written, it was uploaded to the BYU Supercomputer (Appendix A.4) to solve the ~100,000 training configurations and the ~10,000 holdout configurations generated from the Latin Hypercube sampling. It should be noted that the output frequency is dimensionless (i.e. not in Hertz). To achieve the more common definition of frequency in Hertz, we multiplied our output frequencies by the speed of light and divided by the Harmonic Mean.

2.2 The Revised Neural Networks

To improve the accuracy of our network, we split the initial ANN into three networks focused on the cavity frequency, mode, and empty cavity expansion coefficients, identified the worst parameter value ranges (i.e., the configurations that our initial network had trouble generating accurate solutions for), and supplied each neural network with carefully selected datapoint distributions within the aforementioned ranges as described in Sections 2.2.1-2.2.4 below.

2.2.1 Three Neural Networks

The creation of the three neural networks is described in a soon to be published paper as follows:

[10]

> For all network training, the Keras API with Tensorflow backend was used. Optimal hyperparameters, including activation functions, number and type of layers, optimizer, learning rate and number of training epochs, were found using standard procedures. The frequency neural network consisted of 6 fully connected layers comprised of 256, 128, 64, 32, 16 and 1 nodes. The sigmoid activation function, $f(x) = x\ sigmoid(x)$, was used by the first five layers, while the output layer utilized a linear activation function. Other hyperparameters include adam optimizer, initial learning rate of 0.005, batch size of 50, and 1000 training epochs.
>
> The categorical mode network converted the integer mode number values ranging from 1 to 13 to a categorical array of length 13. The optimal neural network for the mode predictions contained 8 fully connected layers with 1024, 512, 256, 128, 64, 32, 16, 13 nodes. To combat overfitting, we added dropout layers after the third and fifth main layers. The first seven layers used the swish activation function, and the final layer used the softmax function as is common in categorical problems. The loss function was categorical cross entropy. Other parameters chosen for this network include adam optimizer, an initial learning rate of 0.0005, batch size of 50, and 500 epochs.
>
> The expansion coefficent neural network contained seven fully connected layers with 2048, 1024, 512, 256, 256, 256, 200 nodes. After the main layers, a lambda layer was added to ensure all predictions were L2 normalized. Swish activation functions were used for all layers. The loss function was mean squared error. Other parameters chosen for this network include adam optimizer, learning rate of 0.0005, batch size of 25, and 500 epochs.
>
> Each of the three networks also used three callbacks: EarlyStopping, ModelCheckpoint, and LearningRateScheduler. EarlyStopping, implemented with patience of 75 for the mode and coefficients network, patience of 150 for the frequency network and minimum delta of 10^{-5}, ensured that no time was wasted on unproductive training. If the error metric failed to improve by at least 10^{-5} in the given number of consecutive epochs, training was stopped early. ModelCheckpoint made sure that a new version of the network was saved only when the error metric improved. After a specific number of epochs, LearningRateScheduler began decreasing the learning rate according to the following formula: $lr_{new} = lr_{old}\mathrm{e}^{-0.1}$. For the frequency network, this process started after 100 epochs, 25 epochs for the coefficients network and 40 epochs for the mode network.

After creation and initial validation, each neural network could then be retrained using the worst parameters from all three neural networks. After each neural network achieved a satisfactory accuracy, their results could be used in tandem to yield accurate predictions for any given cavity configuration.

2.2.2 Highest Errors Lists

The next step after dividing the neural networks was to identify the configuration parameters that yielded the highest errors, henceforth referred to as the "worst parameters". To identify these parameters, we compared the neural network predictions to the solutions generated through the FEniCS code. By taking the difference between these values, a measure of difference in prediction was determined. The worst 1% of these configurations for each of the three neural networks were collected as comma separated values into a spreadsheet and visualized with histograms using *Wolfram Mathematica* (Appendix A.1).

2.2.3 Mathematica Inverse Functions

Upon isolating the worst parameters and visualizing them through histograms, we needed a way to alter our Latin Hypercube sampling method to focus on these problematic areas. In order to address this problem, we created the inverse function remapping method. The method is implemented by taking a function, deriving its inverse, and then combining the two to remap a given distribution. It should be noted that not every distribution identified in the worst parameter datasets could be described by a single function with a definable inverse. For example, some distributions were best characterized by a Cauchy or Gaussian distribution. To address this,

certain parameters were remapped using Python's built-in remapping capabilities in the scipy.stats.distribution library. The syntax for enacting this remap is located in Appendix A.1.

The inverse function method works by identifying a function, such as an exponential decay, and deriving its associated inverse. For example, let the function $f(x)$ be equal to the exponential of x. The inverse $f^{-1}(x)$ will then be the natural log of x. The data in our specific case ranges from 0 to 1 as per Latin Hypercube sampling. By inspection of equations 6 and 7 below, one can see that for a data value of 0, the remap will yield the value of a, and for a data value of 1, the remap will yield the value of b. Thus, the inverse function remaps the Latin Hypercube dataset [0,1] to the range $[a, b]$ while simultaneously spacing the datapoints according to the shape of an exponential function, as visualized in Figure 2.3.

Any data set originally along the interval $[c, d]$ can then be remapped along a new interval $[a, b]$ using these two functions as follows:

$$Remap = f^{-1}\big(f(a) + \big(f(b) - f(a)\big) * data\big) \tag{6}$$

$$Remap = Log_e\big(e^{(a)} + \big(e^{(b)} - e^{(a)}\big) * data\big) \tag{7}$$

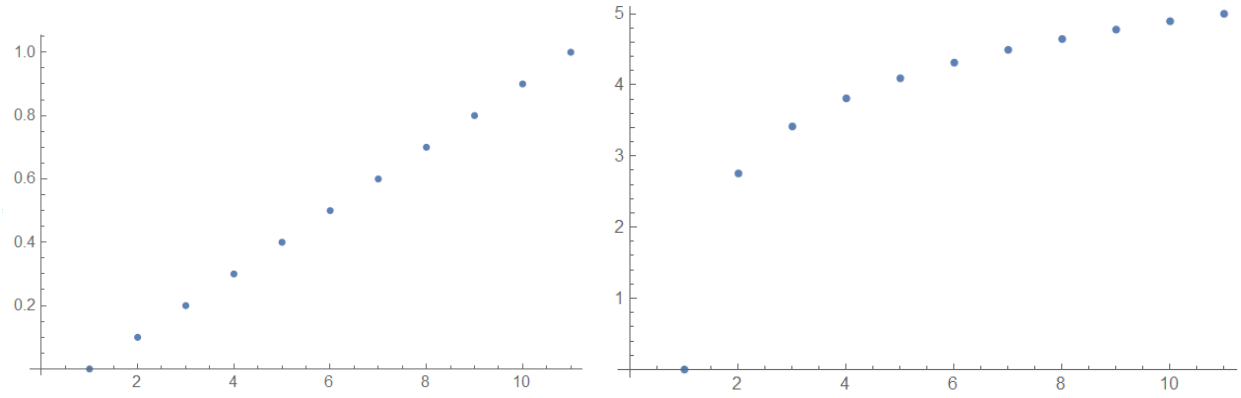


Figure 2.3 The graph on the left shows a plot of 11 datapoints ranging from 0 to 1. The x-axis refers to the point number, while the y-axis corresponds to the value before remapping of each point. The spacing between remapped points is seen in the vertical distances between points. The graph on the right shows these same 11 points after being remapped from 0 to 5 according to the formula above. The vertical spacing is much greater for lower y values but decreases exponentially for datapoints close to 5. In terms of LHC sampling, this would create a distribution that follows an exponential curve from 0 to 5.

2.2.4 New Configurations

After determining the remapping functions in both Mathematica and Python (Appendices A.1 & A.2),we wrote a program in Python for each of the neural networks: frequency, coefficient, and mode. These three programs remapped our Latin Hypercube parameters according to the functions determined in Section 2.2.3. After running the remap, we applied the same reality checks that were utilized in section 2.1.1. Once the three Python files were configured for these reality checks, the frequency, mode, and coefficient files retained 13.27%, 20.09%, and 33.12% of their original data points, respectively. After adjusting for these cut factors, roughly 20,000 configurations were generated for training solutions for each neural network, and 10,000 configurations were generated as holdout data.

Chapter 3

Results and Conclusions

3.1 Predictions from the Initial Neural Network

Our initial neural network was evaluated by calculating the mean absolute error (MAE) for two of the three outputs: normalized frequency and coefficient values. The coefficient neural network error was evaluated for accuracy by calculating the sum of the MAEs for the 200 empty cavity expansion coefficients. The mode was trained using accuracy as the metric. Frequency and coefficient values are derived from numerical networks, and mode values correspond to a categorical network (which can be evaluated based on accuracy). For context, error measurements can be compared to the original values to determine the accuracy of the prediction, with dimensionless normalized frequency ranging from 0.143 to 1.949. The sum of squares for the coefficients adds up to one. MAE is expressed as follows:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - x_i|}{n}$$

$$y_i = Predicted\ Value$$

$$x_i = True\ Value$$

$$n = Number\ of\ data\ points$$

The MAEs and square root sum of squares for our three outputs were calculated accordingly:

Normalized Frequency: 0.011

Coefficients*: 0.0024

Mode: 95.14% Accuracy

*Coefficients are calculated in sets of 200, not just one. The value here is actually a sum of the MAE for each of the 200 coefficients

3.2 Predictions from the Revised Neural Networks

The MAEs and square root sum of squares for our three outputs were calculated as follows:

Normalized Frequency: .0009

Coefficients*: 0.0013

Mode: 95.6% Accuracy

*Coefficients are calculated in sets of 200, not just one. The value here is actually a sum of the MAE for each of the 200 coefficients

3.3 Summary

A microwave cavity, also known as a radio frequency cavity, is a specific type of resonator. We seek to approximate the resonant frequency and field patterns of the quasi-TE_{011} mode for a given cylindrical resonant cavity with both speed and accuracy. Previous attempts at finding solutions to these cavities involved significant computational resource cost and time expense. Thus, we developed three neural networks and trained them with an initial set of 100,000 configurations and validated the networks' predictions with 10,000 more holdout configurations. This initial set of networks was able to predict solutions to a given cylindrical cavity with MAE of 0.011, 0.0024 and an average accuracy of 95.14% for the frequency, coefficients, and mode respectively.

We then refined the networks by selecting the configurations that represented the greatest points of inaccuracy in the network. The networks can now solve any given cavity configuration in ~0.05 seconds with MAE of 0.012, 0.0022 and an average accuracy of 95.46% for the frequency, coefficients, and mode respectively.

Bibliography

[1] PCB Cadence, "What Is A Cavity Resonator And How Is One Used In PCB Design", 2021, https://resources.pcb.cadence.com/blog/2019-what-is-a-cavity-resonator-and-how-is-one-used-in-pcb-design.

[2] C. G. Montgomery, Technique of Microwave Measurements (McGraw-Hill, New York, 1947), Sec. 5.5.

[3] Kyle Miller, “Resonance of Complex Cylindrically Symmetric Cavities Using an Eigenfunction Expansion in Empty Cavity Modes”, Oct 2016

[4] Facebook Research. “Machine Learning.” Facebook Research, 20 May 2020, research.fb.com/category/machine-learning.

[5] Hao, Karen. “What Is Machine Learning?” *MIT Technology Review*, 2 Apr. 2020 www.technologyreview.com/2018/11/17/103781/what-is-machine-learning-we-drew-you-another-flowchart

[6] F. Bre, J. M. Gimenez, and V. D. Fachinotti, “Prediction of wind pressure coefficients on building surfaces using artificial neural networks,” Energy and Buildings 158, 1429–1441 (2018).

[7] Charles Lewis, “Neural Network Approximations of the Temperature of CdTe Quantum Dots”, Brigham Young University Senior Thesis, 2020

[8] McKay, M., Beckman, R., & Conover, W. (1979). A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics, 21*(2), 239-245. doi:10.2307/1268522

[9] Solving PDEs in Python, The FEniCS Tutorial Volume 1, Hans Langtangen, Anders Logg

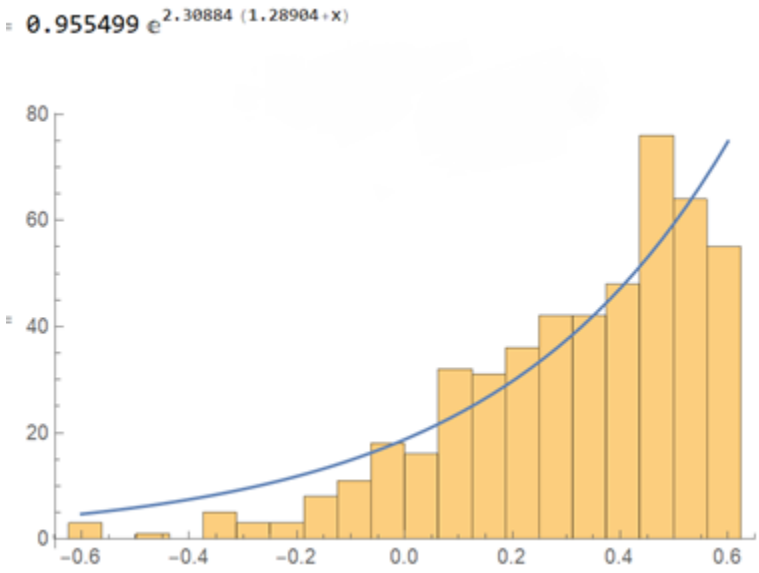
[10] Charles Lewis, Nathan Schwartz, Jordan Bryan, Jonathan Hale, Kane Fanning, John Colton, “Machine learning to predict quasi-TE 011 mode resonances in stacked dielectric cavities” *To Be Submitted*

Appendix

A.1 Remapping using Mathematica

The inverse of a simple function that describes a distribution can be found using Wolfram Mathematica or a similar tool:

1. Identify the function in question



2. Switch out the x and y values and solve

```
(*Find the Inverse of the Functions*)
ifAR = x /. Solve[fARfunction == y, x] /. y → x ;(*Single exponential*)
```

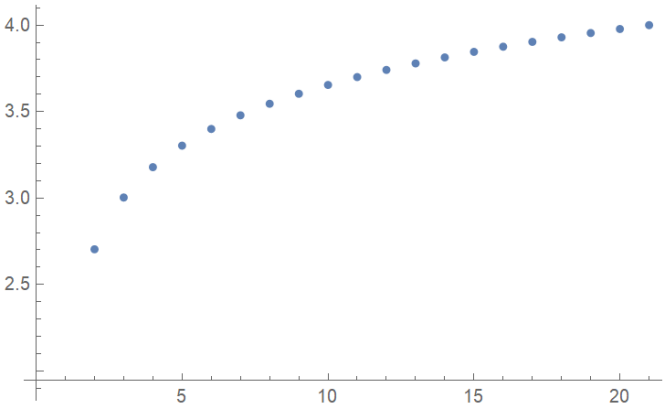
3. The functions may then be checked for desired remapping

```
fARfun[x_] = fARfunction
fARfunp[x_] = ifAR
bc = {.25, 4}; (*.25 to 4 on a log scale*)
ifARlist = Table[fARfunp[ fARfun[bc[[1]]] + (fARfun[bc[[2]]] - fARfun[bc[[1]]]) x], {x, 0, 1, .05}];
ListPlot[fix[ifARlist]]
```

Out[•]= $0.955499\, e^{2.30884\,(1.28904+x)}$

Out[•]= {0.433119 Log[0.05336173372475804 x]}

Out[•]=



As explained in Fig. 2 in the text, the vertical spacing guarantees that most of the distribution will be remapped to the right boundary condition according to an exponential growth curve

The functions may then be implemented in Python:

```
def fAR(value,a,b): # value is the value to remap, a and b are bounds
    return .433119*np.log(      0.05336173372475804 *(0.955499*np.exp(2.30884*(1.28904+a))) +
                               (0.05336173372475804 *(0.955499*np.exp(2.30884*(1.28904+b))) -
                                0.05336173372475804 *(0.955499*np.exp(2.30884*(1.28904+a))) ) * value)
```

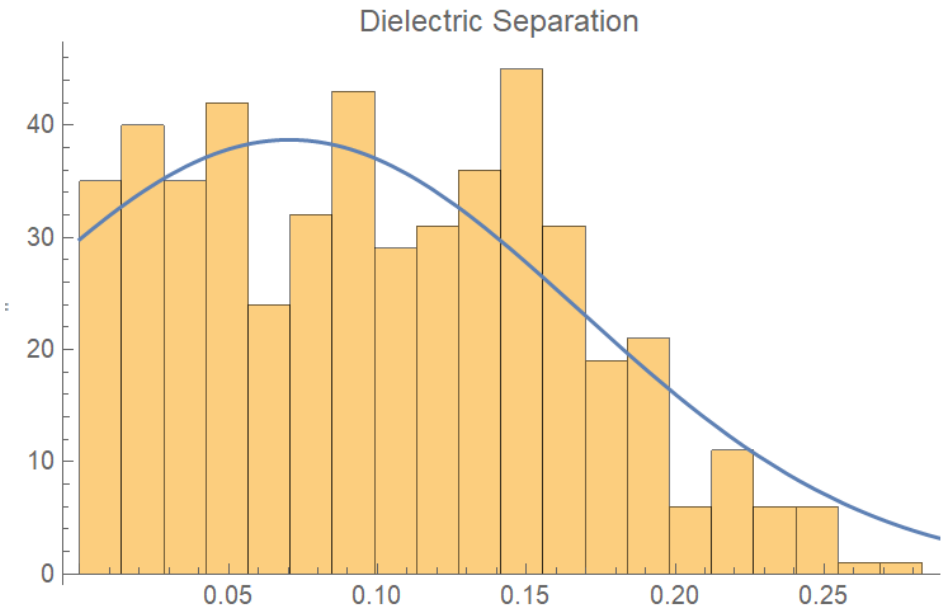
A.2 Remapping using Python

A distribution may be remapped in Python according to the following code:

```
# Separation of Dielectric Constants - PPD
SEPmean = [0.0704356]
SEPstdv = [0.0975359]
y[:, 3] = norm(loc=SEPmean, scale=SEPstdv).ppf(y[:, 3])
```

This remaps the third column in our Latin Hypercube according to a normal distribution with a mean of SEPmean and a standard deviation of SEPstdev.

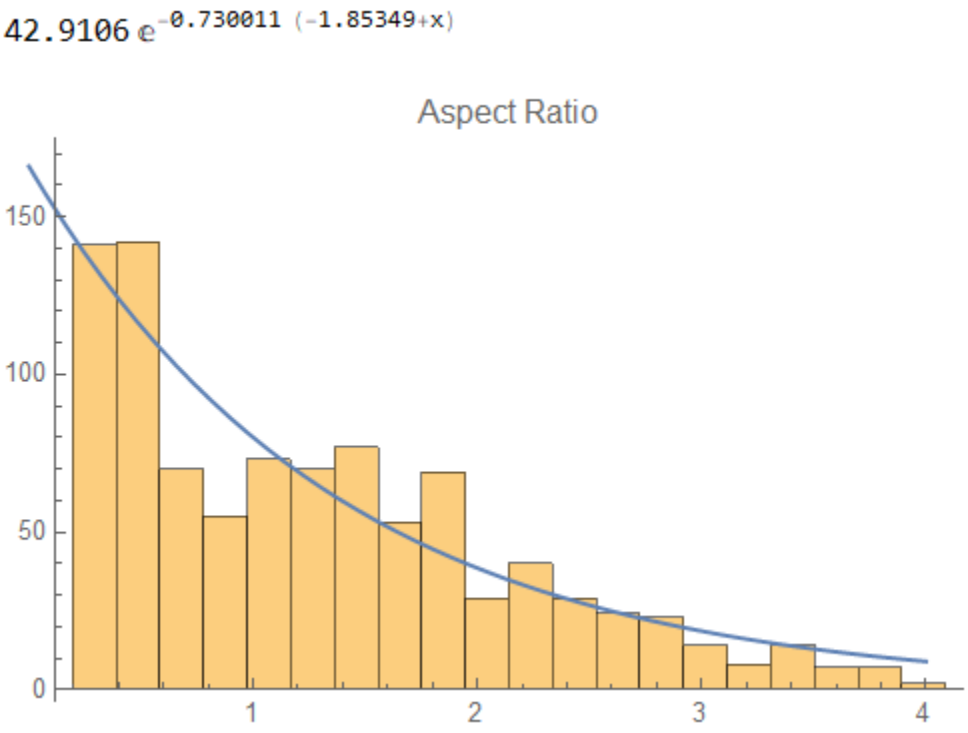
After removing values according to prespecified boundary conditions, the final distribution is visualized:



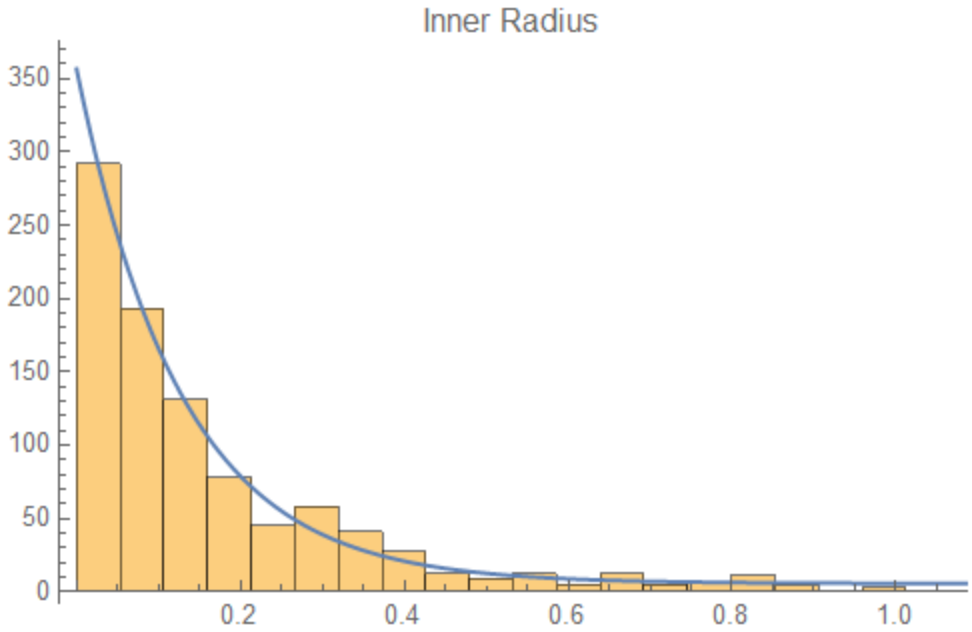
A.3 Mathematica Remap Histograms

Here are the remapping histograms for the three neural networks: Frequency, Coefficient, Mode

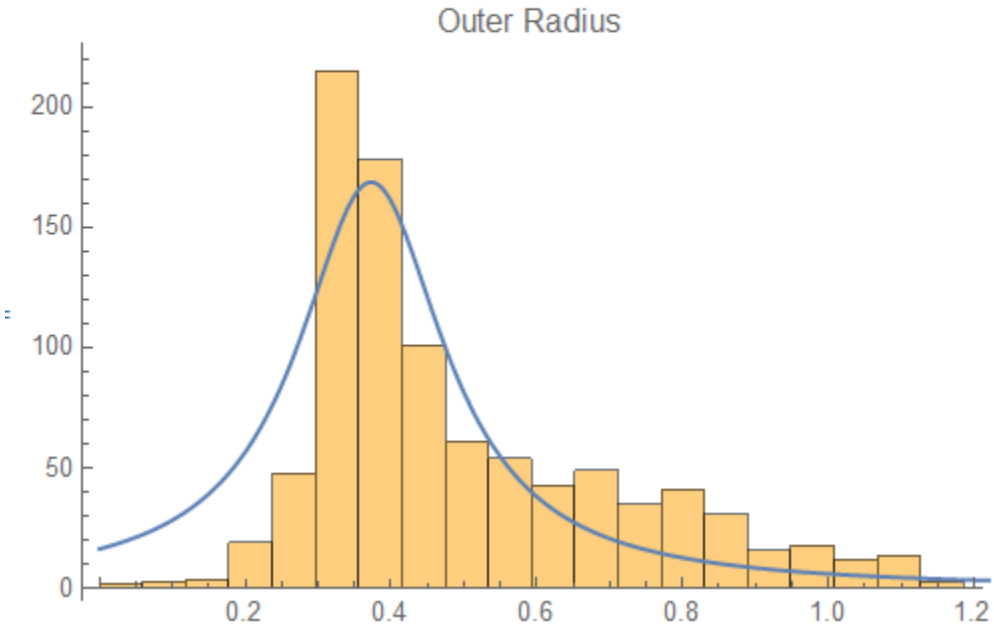
Frequency Neural Network



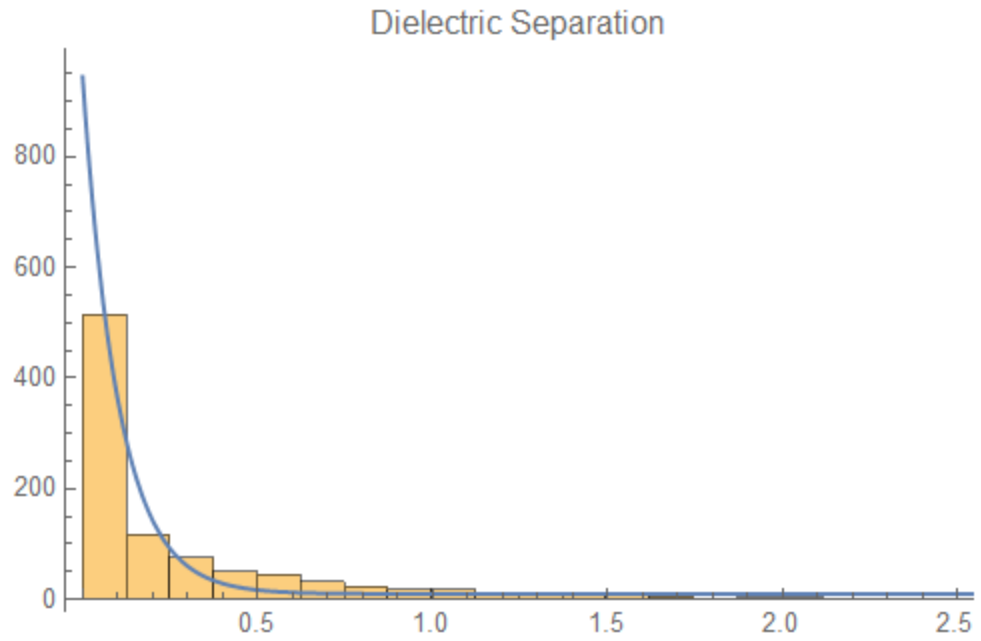
$$5.61702 + 0.000024243\, e^{-7.83629\,(-2.10409+x)}$$



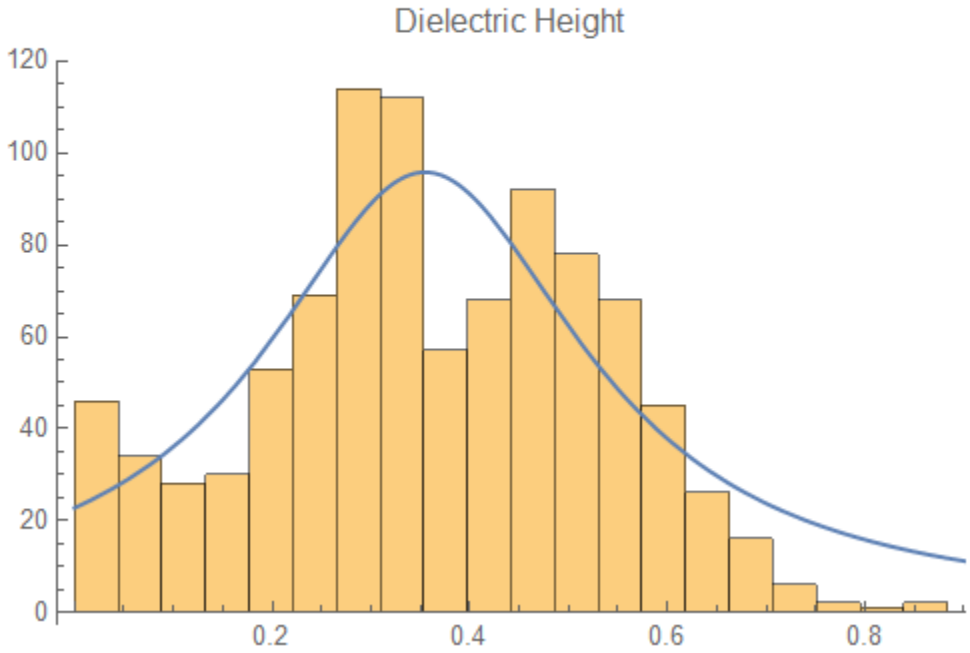
$$\frac{168.718}{1 + 66.0982\,(-0.372746 + x)^2}$$



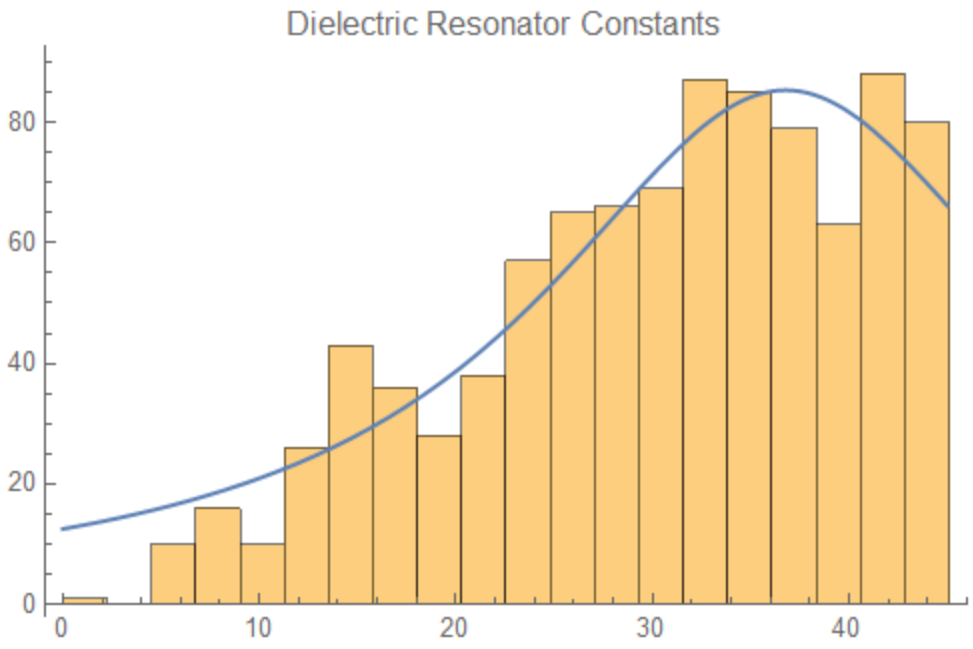
$$3.36485 \times 10^{-8}\, e^{-9.73693\,(-2.46966+x)}$$



$$\frac{95.8641}{1 + 25.54\,(-0.354696 + x)^2}$$

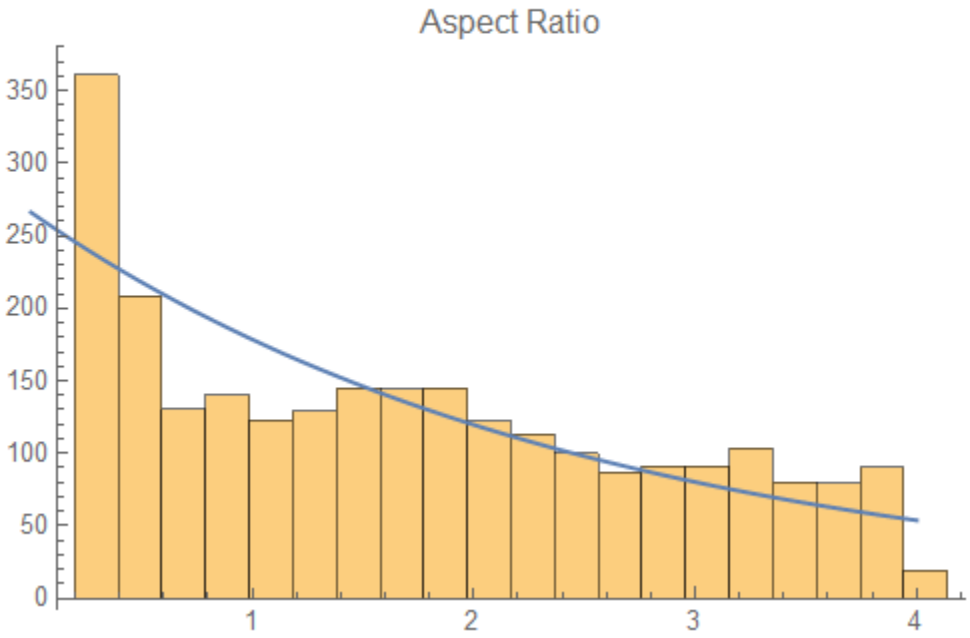


$$\frac{85.3174}{1 + 0.00427718\ (-36.779 + x)^2}$$

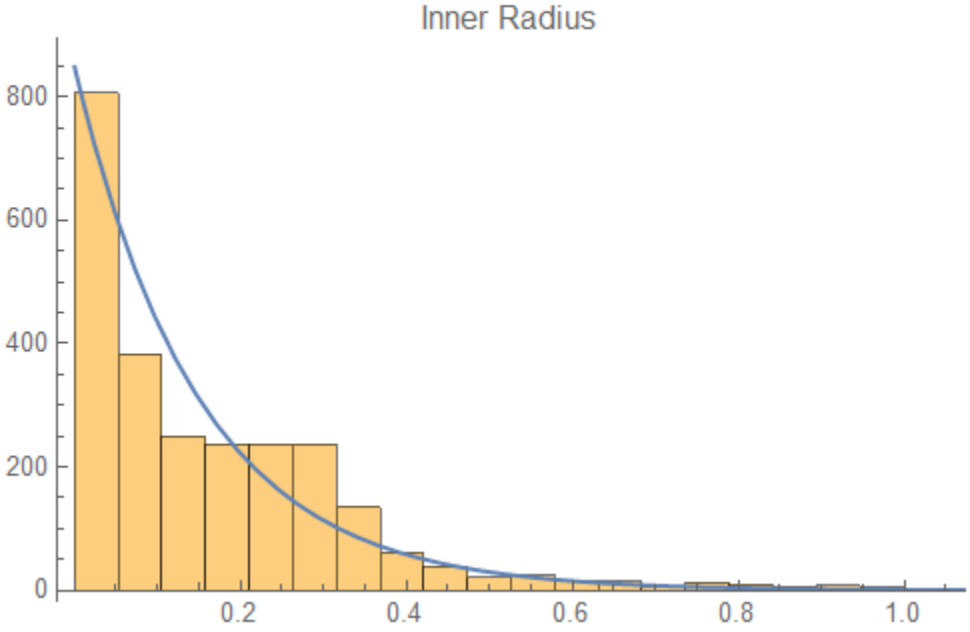


Coefficients Neural Network

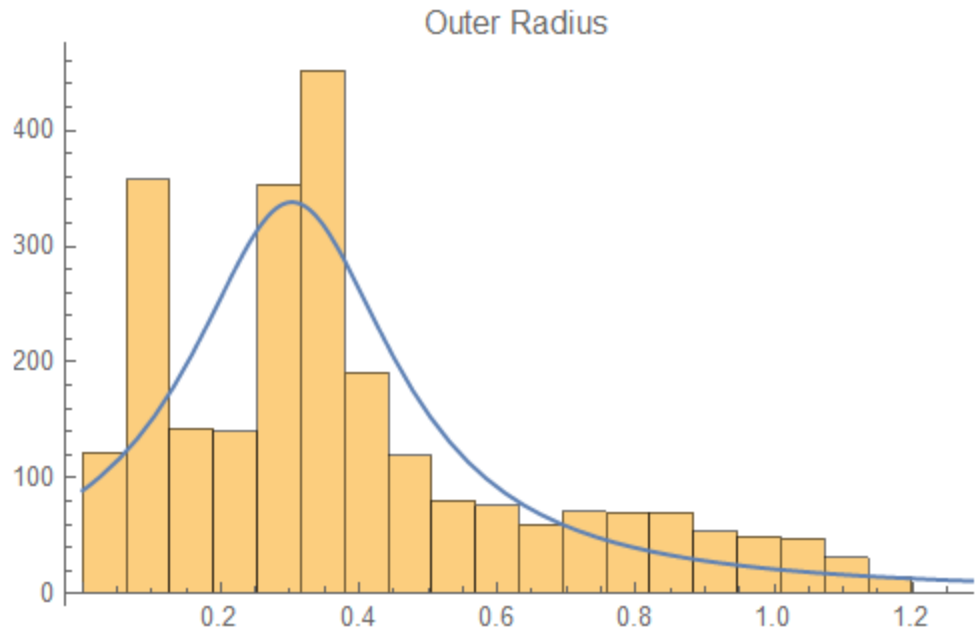
$$4.36402\, e^{-0.399729\,(-10.2847+x)}$$



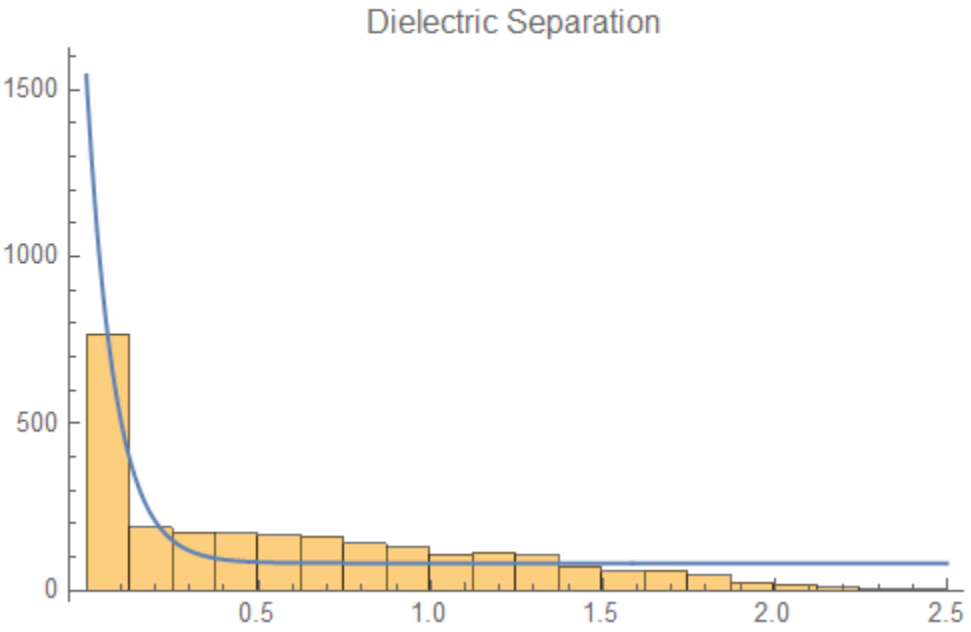
$$4.92145 \times 10^{-6}\, e^{-6.70879\,(-2.82697+x)}$$



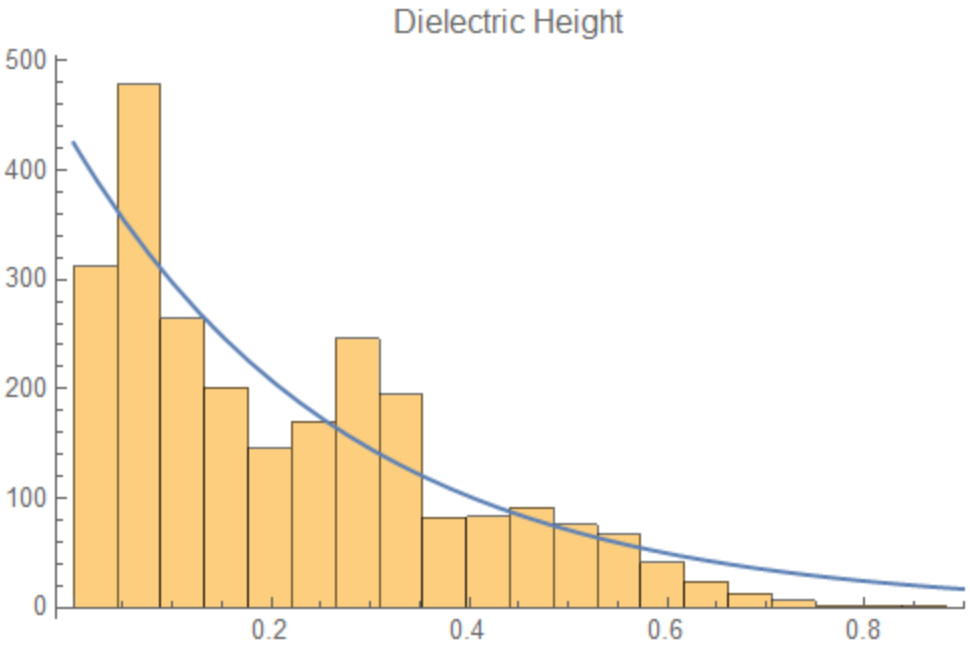
$$\frac{338.047}{1 + 30.3867\ (-0.302901 + x)^2}$$



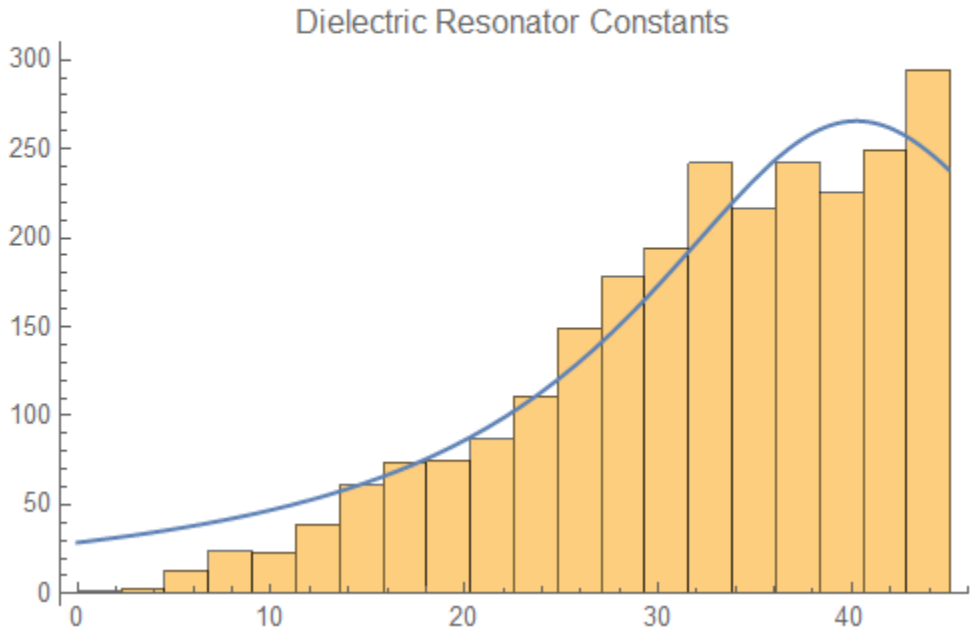
$$81.478 + 1461.77\ e^{-12.2414\ x}$$



$$424.854\, e^{-3.57698\, x}$$

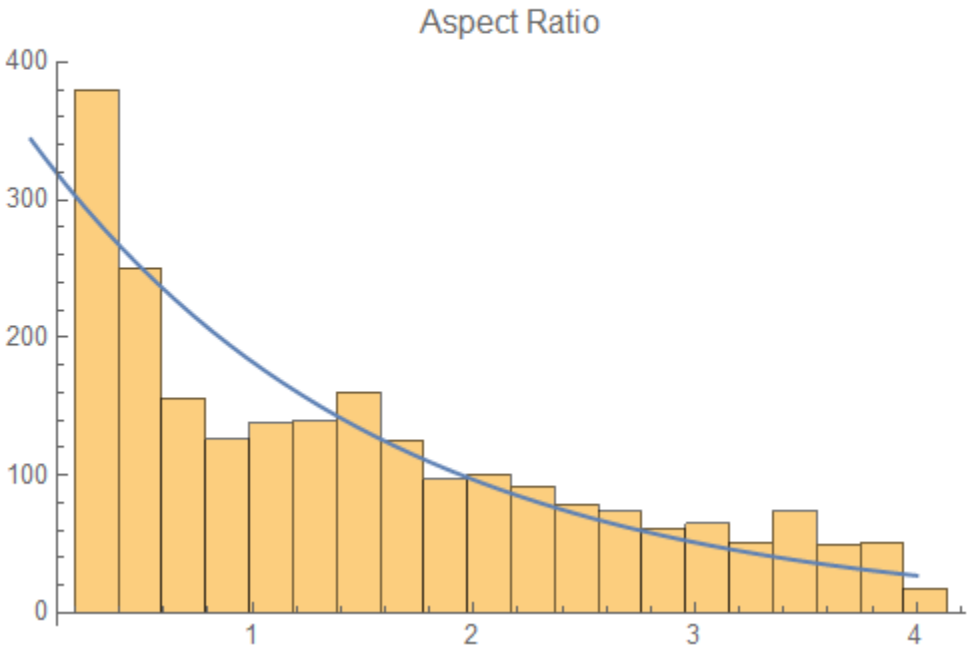


$$\frac{265.643}{1 + 0.00506268\, (-40.2523 + x)^2}$$

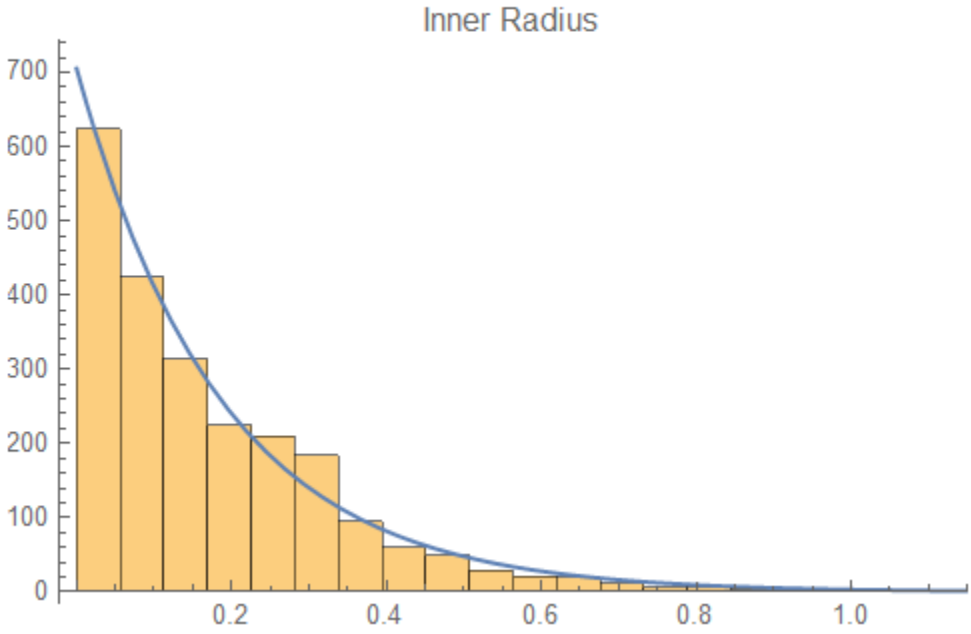


Mode Number Neural Network

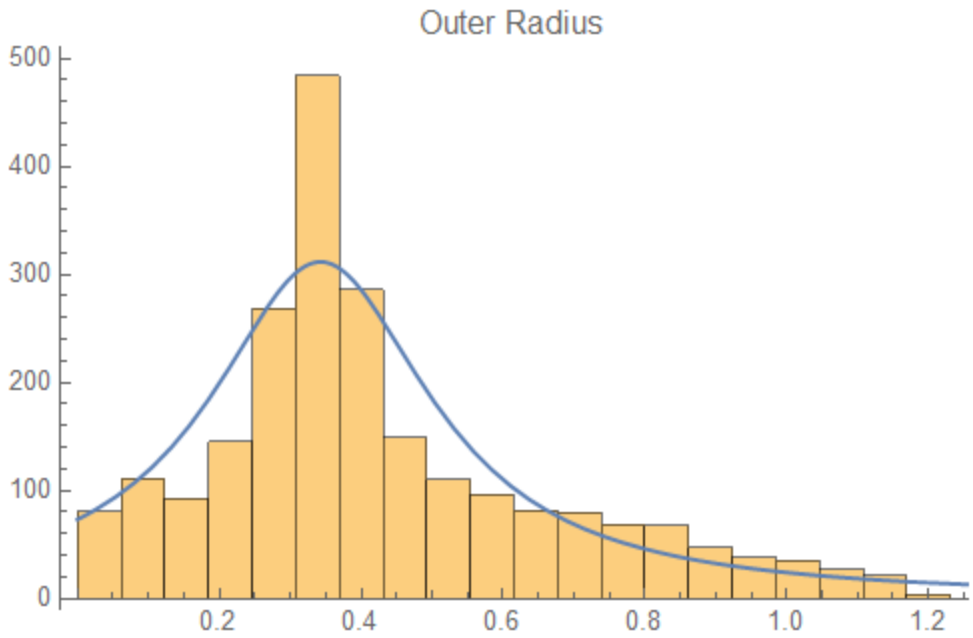
$$57.2608\, e^{-0.63431\,(-2.82631+x)}$$



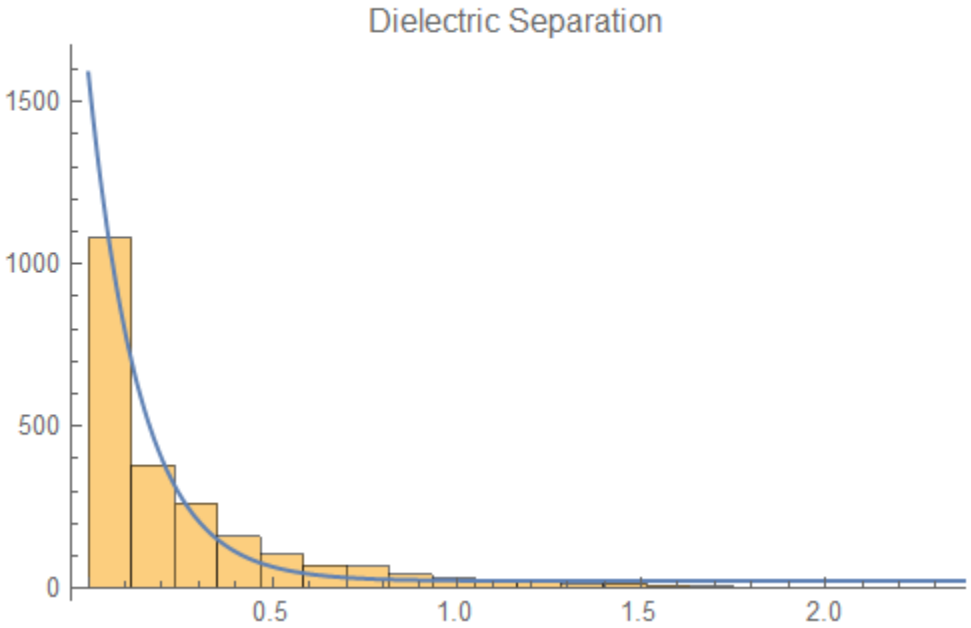
$$705.245\, e^{-5.38387\,x}$$



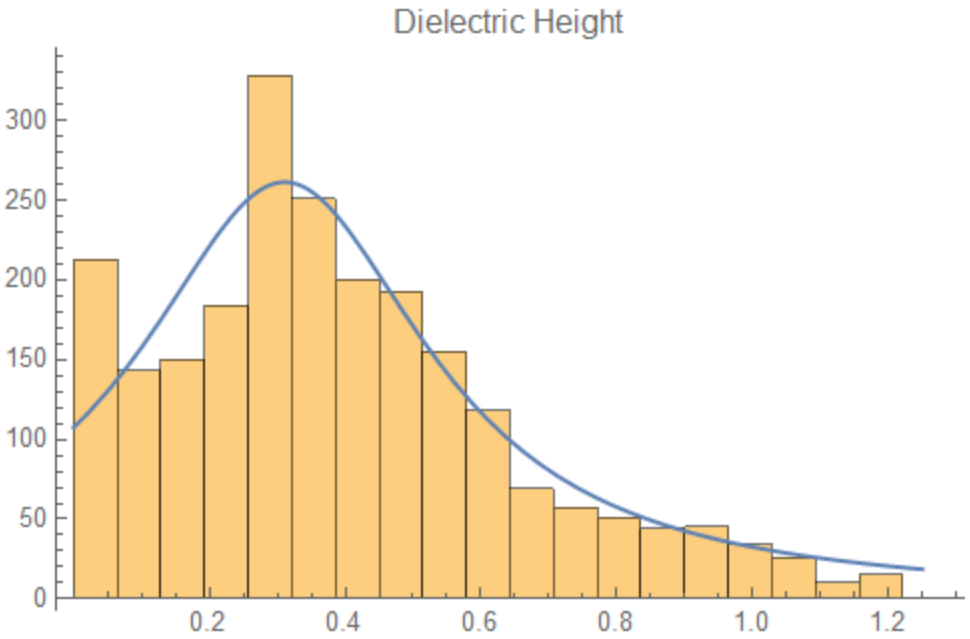
$$\frac{311.656}{1 + 27.7008\ (-0.342683 + x)^2}$$



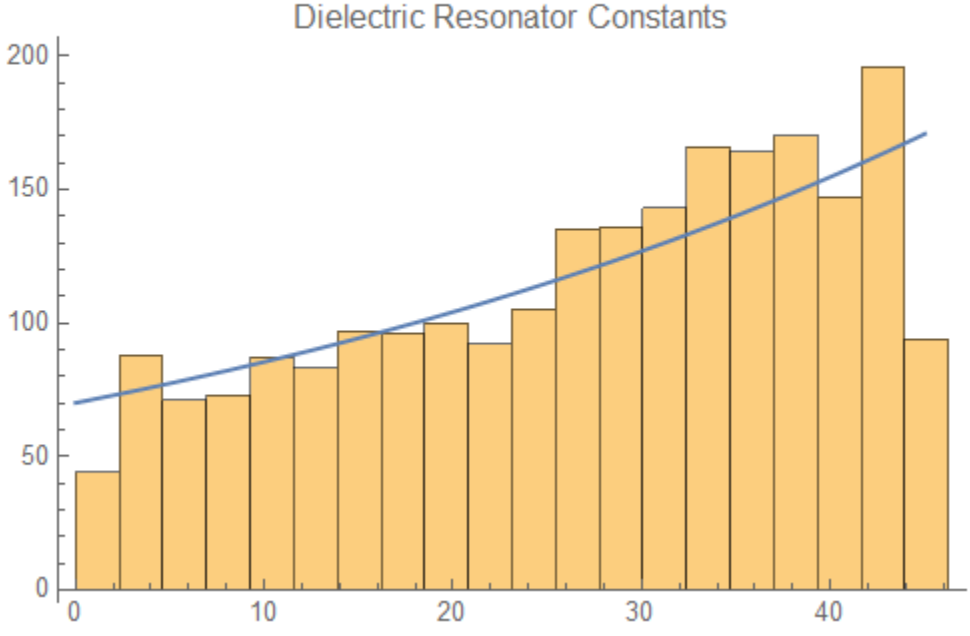
$$22.9969 + 1566.25\ e^{-7.14164\ x}$$



$$\frac{261.52}{1 + 14.8087\ (-0.310809 + x)^2}$$



$$e^{0.0198049\ (214.557+x)}$$



A.4 Supercomputer Use

There are 6 files required to use the BYU supercomputer: a .yml environment file, submissions_bash.sh, changejobscript.py, the csv containing the configurations, ultimateTElist.csv, and solveTE011v3.py. After these files have been uploaded, navigate to the directory where the files are stored and complete the following steps:

1. First, the environment file needs to be run for either linux or mac. After the environment has been created, run *conda activate fenicsproject* in the terminal to activate the environment. After the environment has been created, you will need to activate each time you log on to the supercomputer, but you won't have to create it each time.

2. Edit the submissions_bash.sh file by changing maxnum to be the maximum number of lines in your csv configurations file. Set *a* to be 1. Set *b* to be the desired submission interval. For instance, if you want to submit 1000 lines in each job, then set *b=1000*.

3. Edit changejobscript.py by changing infile to be your configurations file. Change the outfile to be the file where the solutions will be written to. Change the email to be your own.

4. Run *bash submissions_bash.sh* in the terminal to begin the submissions.