Simulating the $TE_{01\delta}$ Mode of a Dielectric Resonator in Free Space

Daniel John King

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

Bachelor of Science

John Colton, Advisor

Department of Physics and Astronomy

Brigham Young University

ABSTRACT

Simulating the $TE_{01\delta}$ Mode of a Dielectric Resonator in Free Space

Daniel John King
Department of Physics and Astronomy, BYU
Bachelor of Science

Dielectric resonators, which are used to create standing waves in the microwave band, have been an important topic of research for decades due to their usefulness as antennae, filters, enhancing electron spin resonance, and a variety of other applications. For all these uses it is important to know the resonant frequency of a given resonator, but predicting this frequency is difficult, as only approximate formulas exist and performing precise numerical calculations to determine the frequency is computationally expensive. In previous work, we demonstrated that a neural network can be trained to accurately predict the $TE_{01\delta}$ mode frequency and field modes of a pair of dielectric resonators inside a cylindrical microwave cavity, with the training data created for a variety of different configurations using the finite element method. Here, we modify this approach to similarly determine the frequency and field modes for the $TE_{01\delta}$ mode for a single dielectric resonator in free space, by using a finite-difference method with perfectly-matched layer boundary conditions.

Keywords: Resonator, Dielectric, Simulation, Microwave, Finite-difference, PML

ACKNOWLEDGMENTS

I would like to thank my faculty advisor Dr. John Colton for his continued help and support, as well as the other members of my research group I've had the privilege working with. A special thanks as well to Emma McClure, who taught me almost everything I know about working in the lab. Finally, I would like to thank my parents, whose dedication, love, and support have made all of this possible for me.

Contents

Table of Contents **vii**

1 Introduction **1**

2 Methods **3**
2.1 Overview of Simulation Methods 3
2.2 Description of Resonant Mode and Boundary Conditions 4
2.3 Explanation of Perfectly Matched Layers 5
2.4 Introduction to Meep 7
2.5 Code Implementation 8

3 Results **13**
3.1 Single Resonator Field Values 13
3.2 Single Resonator Configuration Runtimes 15
3.3 Single Resonator Frequency Accuracy 17
3.4 Frequency Results Across Parameter Space 18
3.5 Future Work 20

Appendix **25**

Bibliography **29**

Index **31**

Chapter 1

Introduction

A dielectric resonator is a piece of polarizable material designed to create standing electric and magnetic fields, typically in the microwave band [1]. These standing waves are formed due to the boundary between the edge of the dielectric material and free space. The exact layout of the standing waves that are formed, and the resonant frequency at which this occurs, depends upon the shape and relative permittivity of the resonator, and the frequency being used to excite the resonator, as any given resonator has a number of different resonant modes that can be excited [2]. Although they can come in a variety of sizes, shapes and materials, the most common resonators are cylindrical, made of ceramic, and on the order of centimeters in size.

The idea of using a finite piece of dielectric material as an electromagnetic resonator was first proposed in 1939 by Robert Richtmyer [1]. Now, because of their small size and cheap cost to produce, dielectric resonators are used commonly in industry for applications such as band-pass filters, electronic oscillators, and microwave antennas [3]. In research settings, they are often used to enhance field strength for electron spin resonance spectroscopy experiments, typically by placing a sample of a material to be measured inside of a hollowed out resonator, and then placing that resonator inside of a conductive microwave cavity [4].

This specific configuration of a hollow resonator in a cavity was the focus of a research paper

by the Colton group, published in October 2021 in the journal *IEEE Transactions on Microwave Theory and Techniques* [4]. The object of this paper was to determine a method of more quickly calculating the resonant frequency and electric fields of a specific resonant mode, called the $TE_{01\delta}$ mode, of a resonator (or set of stacked resonators) inside a microwave cavity, all of arbitrary dimensions. Because no analytical solutions exist for determining these values, one must resort to either experimentally measuring them, running time-consuming simulations to determine them, or relying on formulas that can only approximate these values within a certain range of dimensions or dielectric values. The solution reached in this paper was to use a supercomputer to run simulations for a large number of different configurations, and use the resulting data to train a neural network to very accurately predict the fields and resonant frequencies for any set of input dimensions and dielectric values.

The work of this thesis is to show how this method can be extended to work for a dielectric resonator in free space, as even though this configuration is much simpler, there still is no analytical solution for determining its fields and resonant frequencies. Extending this computational approach to work for this configuration presents a number of challenges, chief of which is modifying the boundary conditions of a finite simulation space to approximate an infinite space. This was accomplished by implementing a type of artificial absorbing boundary called a perfectly-matched layer (PML). Implementing this necessitated other modifications, including changing the type of modeling technique used and the software library used to implement the simulation. Upon doing this, we show that a dielectric resonator in free space can indeed be accurately modeled, achieving a frequency accuracy nearly identical to previously used methods. The potential of utilizing this data in training a predictive neural network with similar levels of accuracy is then discussed.

Chapter 2

Methods

In this chapter, we present an overview of the methods used in implementing the dielectric resonator simulation. We start in Section 2.1 with a brief overview of the general simulation methods used both in our past work and here. In Section 2.2 we explain in greater depth the nature of the problem we are trying to simulate, which is elaborated upon in Section 2.3 where we present the solution to the problem of simulating an infinite space. Section 2.4 introduces the code library used to implement the simulation, and in Section 2.5 the code itself for the simulation is outlined.

2.1 Overview of Simulation Methods

One popular method of simulating electrodynamic systems is the finite element method. This method works by dividing the simulation space into a series of cells (called elements), and finding an approximate solution to Maxwell's equations for each element, in such a manner that the function is piecewise-continuous with the functions of the surrounding elements [5]. These elements can take on a number of different geometries, and can vary from one element to another within the same simulation. This flexibility allows the finite element method to accurately model systems with complex geometries [6]. Implementing this method for vector valued functions such as electric

and magnetic fields is often difficult, but can be simplified in some cases by exploiting symmetries in the system. This method was what was used in our previous work, to model a set of stacked resonators inside of a conducting microwave cavity.

Another popular class of methods are finite-difference methods. Unlike the finite element method, this method discretizes the simulation space using a rectangular grid, with constant spacing in each dimension [7]. After specifying some type of electromagnetic source, the simulation then uses Maxwell's equations to estimate how the fields in the system will evolve, either in a time or frequency domain. These methods are often easier to implement, but generally can't handle complex geometries as well as the finite element method. For example, if material boundaries in the simulation don't exactly line up with the spacing of the computational grid, they will typically be rounded to match, which will introduce error into the resulting fields or frequencies being calculated. Though this does present some limitations to the number of configurations that can be tested, a frequency domain variant of the finite difference method is what we have elected to use here, due its ease of use.

2.2 Description of Resonant Mode and Boundary Conditions

Our goal is to determine both the electric and magnetic fields, as well as the resonant frequency, of the $TE_{01\delta}$ mode of a cylindrical dielectric resonator. TE stands for transverse electric, meaning that electric field E has no z component, and for this specific mode it also has no r component as well. This means it points only in the ϕ direction, relative to the resonator (using the typical cylindrical coordinates of r, ϕ, and z). The 0 means that the E field has no ϕ dependence, i.e. the field is rotationally symmetric. The 1 denotes a single anti-node (maximum field value) in the r direction, and the delta has a similar meaning for the z direction, though we use delta instead of 1 because the field in the z direction only goes to zero in the limit of infinity in both directions,

so the field extends beyond half a wavelength in this dimension. As is consistent with Maxwell's equations, the magnetic field B for this mode is zero in the ϕ direction, but nonzero in both the r and z directions. From a distance the B field is similar to that of a magnetic dipole, meaning this field mode is sometimes called the "magnetic dipole mode" [2].

In our previous work, these field patterns were altered by the presence of conducting walls surrounding the resonator. These conducting walls act as boundaries, forcing both the E and B fields to go to zero at these walls. This is the easiest case to simulate, as any electromagnetic simulation requires boundary conditions, and so the values at these boundaries are simply kept at zero. This of course alters the fields and resulting resonant frequency of the resonator itself, because of how the fields reflect off of these boundaries. However, what we are interested in now is simulating a resonator in free space, meaning that we can't use this boundary condition, or any similar boundary condition that results in reflections that alter the resonant frequency and fields. Of course, we still need some form of boundary, because we can't use an infinite simulation space with our finite computing capabilities. What we need then is some kind of boundary that is able to absorb electromagnetic fields of any frequency without producing reflections, which turns out to be a difficult mathematical problem. Luckily, a solution to this conundrum was presented in 1994 by Berenger [8], in the form of something called a perfectly matched layer.

2.3 Explanation of Perfectly Matched Layers

A perfectly matched layer (PML, for short) is not a boundary; rather, it is a layer of finite thickness that pads the boundary of the simulation. It works by implementing a mathematical transformation that causes any electromagnetic fields that enter this layer to quickly decay, so that by the time the field reflects off of the boundary and exits the layer back into the simulation space, it is nearly zero and therefore has no effect on the simulation. This mathematical transformation does not correspond

to any real phenomenon or material; it is simply a tool that lets us approximate an infinite space with our finite simulation.

PMLs work by using what is called an "analytic continuation" into a complex space. What this means is that the real electric and magnetic field values gain an imaginary component via a mathematical operation inside the PML that causes them to decay exponentially. For a scalar wave equation in one dimension, the transformation takes this form [9]:

$$\frac{\partial}{\partial x} \rightarrow \frac{1}{1 + i\frac{\sigma_x(x)}{\omega}} \frac{\partial}{\partial x} \tag{2.1}$$

where ω is the frequency of the incident wave, and σ_x is a function that is nonzero within the PML, and zero without. The function within the PML can vary, but typically is chosen to be something that starts at zero, turns on gradually, and increases throughout the layer, like x^2, for example. This formula can be extended to work for higher dimensions, and can be further modified to work for alternate coordinate systems such as cylindrical or spherical. For the corresponding transformations for the three-dimensional, cylindrical coordinate system being used in our simulations, see Teixeira et al [10].

Unfortunately, the code library (called FEniCS) used to implement the finite element method approach used previously does not include native support for complex numbers, which makes a direct implementation of PMLs impossible. There is an updated version of the FEniCS library, called FEniCSx, which does include support for complex numbers; however, it differs enough from FEniCS to make a port of the old code to FEniCSx very difficult. This is compounded by the fact that the documentation for FEniCSx is rather sparse and poorly organized. Because of this, we decided to turn to a different code library that could much more easily facilitate this task. This library is called Meep.

2.4 Introduction to Meep

Meep is, according to the documentation web page, a "free and open-source software package for electromagnetics simulation via the finite-difference time-domain method spanning a broad range of applications" [11, 12]. It was originally developed by a research group out of MIT, and the current version uses a Python interface. Its chief advantages over the previously used FEniCS library are that the user doesn't have to implement the electromagnetic equations to be solved, and it has PML functionality and support for cylindrical coordinates built in. All the user needs to do is specify the configuration they wish to model, including where they want PMLs, and the Meep code backend does the rest. Of course, since Meep uses a finite difference method instead of the finite-element method like FEniCS, it cannot support as complex of geometries, and finding resonant modes poses some challenges that can increase simulation runtime. However, the benefits of the ease of implementation far outweigh these drawbacks.

One quirk of Meep is that it does not use standard SI units such as meters. Instead, Meep uses a type of dimensionless unit system that the developers refer to as "Meep units" [13]. In Meep units, the speed of light c, the permittivity of free space ε_0, and the permeability of free space μ_0 are all set to 1. This makes all length values simply expressed as ratios of each other, and the frequency of a given electromagnetic wave just becomes the inverse of its wavelength (using the standard formula $c = \lambda f$). Then, when you calculate a frequency in Meep units, if you want to convert that to SI units, you start by picking a real length value in SI units to correspond to the unit length of your simulation. Then, you divide your frequency by that length value, and multiply by the SI value of c. This means that the actual size of a resonator being simulated is irrelevant. Therefore, the only relevant quantities for determining the resonant frequency in Meep units of a simple, non-hollow, cylindrical resonator are its relative permittivity and its aspect ratio, which is the ratio of its height to its diameter.

Now because the resonator size is irrelevant, it is helpful to keep the size normalized between

resonators using some metric, so that resonators of different aspect ratios occupy a similar amount of area within their respective simulations. In keeping consistent with our previous work using FEniCS, we determined that the best metric to use would be the harmonic mean of the diameter and height of the resonator. The harmonic mean is a type of average given by the formula:

$$HM = \frac{4hr}{h+2r} \tag{2.2}$$

where r is the radius and h is the height of the resonator. For our simulations we chose to keep the harmonic mean of each resonator at or as close to 1 as possible.

2.5 Code Implementation

What follows here is a brief list of the general steps required to implement the simulation in Meep, along with a basic explanation of each step. For the actual code itself, refer to the Appendix. The steps are as follows:

1. Define the simulation and resonator geometry
2. Implement the PML boundary layers
3. Create a source function
4. Run the simulation and check the output mode

To define the simulation geometry, a coordinate system must first be specified, which in our case is cylindrical. Doing this simply requires setting a dimension parameter equal to an enumerated constant that corresponds to cylindrical coordinates. Once this is done, the geometry of the simulation space is defined through specifying the radius and height of the space. One would normally also specify a value for the angular coordinate ϕ, but in this case we can actually set this to zero, which tells the Meep backend that our simulation will be rotationally symmetric. This reduces

our simulation space from a 3-dimensional space down to a 2-dimensional space, greatly speeding up runtime. We then specify the radius, height, and relative permittivity of our dielectric resonator, which will be centered in our simulation space.

In doing all this, we make sure that we define the simulation space to be significantly larger than the resonator itself, so we can see how the fields extend into the free space surrounding the resonator. This also reduces the small amount of error introduced by the PML boundary layers. However, adding space also increases runtime, so a balance must be found between space and runtime. For our tests, we found it sufficient to set both the simulation radius and height to a length of 4. We felt that this was sufficient, given that across all the configurations we tested, no resonator ever had a radius greater than 2.4 or a height greater than 1.2.

Next we implement the PML boundary layers. In Meep, this is as simple as specifying the desired thickness of our PML layers with the variable *dpml*, and then calling the function mp.PML(*dpml*). Generally, a good rule of thumb for PML thickness is to make the layer at least half as thick as the expected wavelength of our fields, and since we know that our wavelengths are going to be on the same order of magnitude as the radius of our cylinder, we found setting *dpml* equal to 1 to be more than sufficient.

We then must create a source function, which is some type of electric or magnetic oscillator within the system. Otherwise, all E and B field values would simply return as zero. For this we chose to use a magnetic source oriented vertically in the center of the resonator, as we know from the field topology of the $TE_{01\delta}$ mode that the B field exists along the z-axis only in the z direction, and is at a maximum in the center of the resonator. There is a problem however, because this source needs a specified frequency, one that is close enough to the resonant frequency that we are trying to find that it excites the correct mode in the resonator. Luckily for us, there is an approximate formula we can use:

$$f_{GHz} = \frac{34}{r\sqrt{\varepsilon_r}}\left(\frac{r}{h} + 3.45\right) \tag{2.3}$$

Here, r is the radius of the cylinder and h is the height (both in millimeters), with the resulting frequency in GHz. Converting this into Meep units gives us:

$$f = \frac{0.113413}{r\sqrt{\varepsilon_r}}\left(\frac{r}{h} + 3.45\right) \tag{2.4}$$

This can be further rewritten in terms of the aspect ratio and harmonic mean as:

$$f = \frac{0.226826}{HM\,(1+AR)\,\sqrt{\varepsilon_r}}\,(1 + 6.9AR) \tag{2.5}$$

Eq. 2.3 is well established in literature and can be found in *Dielectric Resonators* by Kajfez and Guillon [2]. This formula is said to be accurate to within about 2% within the following ranges:

$$0.25 < AR < 1 \tag{2.6}$$

$$30 < \varepsilon_r < 50 \tag{2.7}$$

We have found through testing that outside these ranges, this formula tends to predict a value above the actual resonant frequency.

Once all of these parameters have been specified, the simulation can be initiated, and the resonant frequency and fields can be calculated using the Meep sim.solve_eiqfreq() method. This method uses an iterative frequency-domain solver to compute the resonant frequency to within a specified margin of error, which is 10^{-7} in our case. Once a solution has been converged upon, the simulation outputs the resulting resonant frequency, as well as the E and B fields as a series of 2D arrays, one array for each field component. To then ensure that the mode found is the $TE_{01\delta}$ mode, we check the E field to make sure that at no point does it switch directions; that is, if the E field points generally in the $+\phi$ direction, there aren't any points in the resonator or surrounding

space where it points in the -ϕ direction. If it does, or if the simulation fails to converge, this is generally because we're using an aspect ratio or relative permittivity value outside of the range that our approximate formula is good for, and as a result our source frequency is usually too high and doesn't excite the proper mode. To fix this, we then iteratively call the sim.solve_eigfreq() method, lowering the source frequency by 10% each time, until the proper mode is found.

Chapter 3

Results

In this chapter, we present the results and our discussion of our simulations. We first start in Section 3.1 by plotting the electric and magnetic fields, as well as the resonant frequency, for a single, typical configuration. This serves as an example from which we can learn about the simulation accuracy. Comparison with approximate formula values are also discussed. In Section 3.2, we examine the runtime behavior of our simulations, and how it scales with resolution. Similarly, in Section 3.3, we discuss how the accuracy of the determined resonant frequency for an example configuration varies with resolution. This all leads us into Section 3.4, where we apply what we have learned from these examples across a wide variety of different configurations using the BYU supercomputer. We finish in Section 3.5 with a discussion of potential future work.

3.1 Single Resonator Field Values

A plot of the electric and magnetic fields for a typical solid single resonator is shown in Fig. 3.1. As is expected for the $TE_{01\delta}$ resonant mode, we see that the E field only propagates in the ϕ direction, while the magnetic field propagates only in the r and z directions. There are a few things worth mentioning here, however, particularly the E field values along the z-axis and the total B field flux.

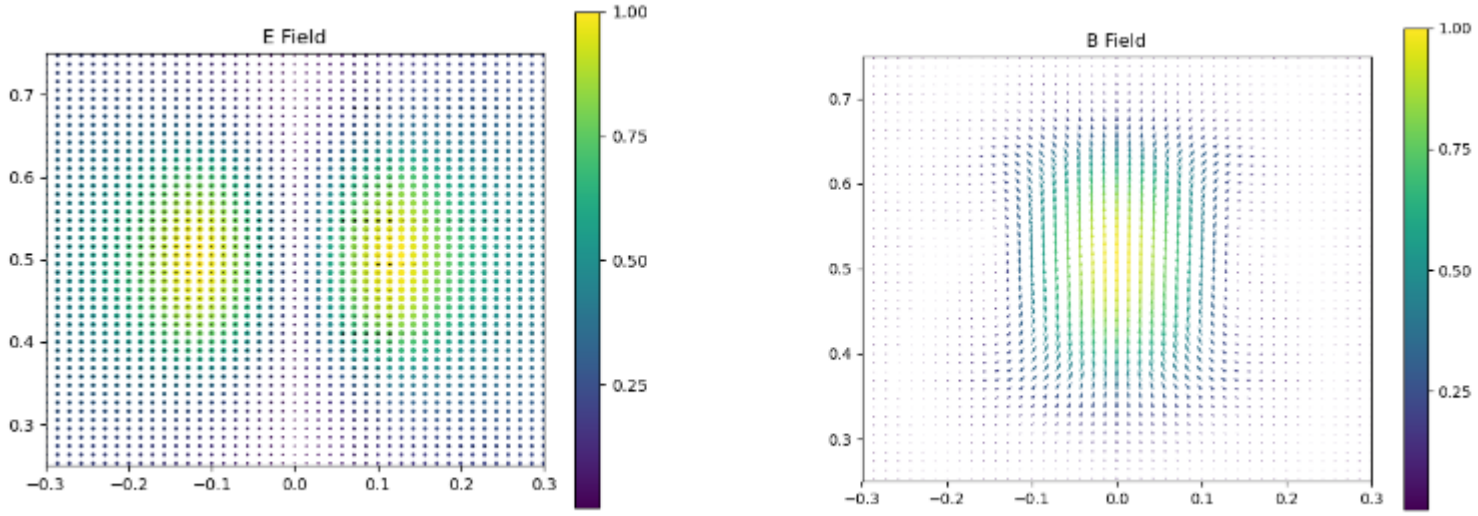

Figure 3.1 Cross-section showing electric and magnetic field lines for a resonator with an aspect ratio of 1, and a relative permittivity of 40. This simulation was performed at a resolution of 20.

Starting with the B field plot, we should expect to see a total downward flux equal to the total upward flux when summing across all space, in accordance with Gauss's law for magnetism. However, the plot clearly shows a net upward flux. This is because in reality this resonant mode extends out into all space, so even though the fields approach zero infinitely far away, if you were to sum over all space, you would see a net flux of zero. This is why using perfectly-matched layer boundaries in our model is so important: had we not used those, our B field plot would look significantly different, as the entirety of the field would be enclosed in the simulation space.

We also see that the electric field is not exactly zero at $r = 0$, as one should expect to see for this mode. This is a form of numerical error caused by the finite-difference method, specifically from how the simulation truncates the field values near the magnetic field point source [12]. The magnitude of these erroneous values along the z-axis is dependent largely on the specified resolution of the simulation: the higher the resolution, the closer to zero these values become.

To quantify this, we found the maximum value of the electric field along the r-axis for a typical

configuration, and divided that value by the value of the electric field at the origin. We then repeated this for a variety of different resolutions. What we found was that in almost every case, the max-to-min field ratio was very nearly equal to the resolution. So, for a resolution of 50, the field at the origin was around 1/50 of the maximum electric field value. Because we know this field value should actually be zero, we can estimate that the maximum error of our field will have a lower bound equal to the inverse of the resolution. So, a simulation with a resolution of 50 will have a field error of at least 2%, and a simulation with a resolution of 10 will have a field error of at least 10%. This is less than ideal, given that the finite element method approach used in our previous work was significantly more accurate than this. This of course can be mitigated by using even higher resolutions, but this leads to issues with long runtimes, which is what we will now discuss.

3.2 Single Resonator Configuration Runtimes

Because the simulation resolution corresponds to the number of discrete divisions per unit length, and our simulation space is two dimensional, the total number of points in our simulation grid is proportional to the resolution squared. This means that at best, we can hope that the time it takes for a simulation to run to completion would also be roughly proportional to the resolution squared. For example, if you doubled the resolution, you would expect the runtime to increase by about a factor of four. Unfortunately, this is not the case. Instead, the runtime increasese with resolution by a polynomial factor closer to 4. To show this, the runtimes for a typical configuration tested at a variety of different resolutions are plotted in Fig 3.2. This behavior puts an upper limit on the accuracy we can achieve, as with a resolution of 60 a single simulation takes over 2 hours to complete on a node of the BYU supercomputer. Given that our goal is to potentially run thousands of different configurations, any resolution above 60 is simply untenable, even with a supercomputer than can run many simulations simultaneously.

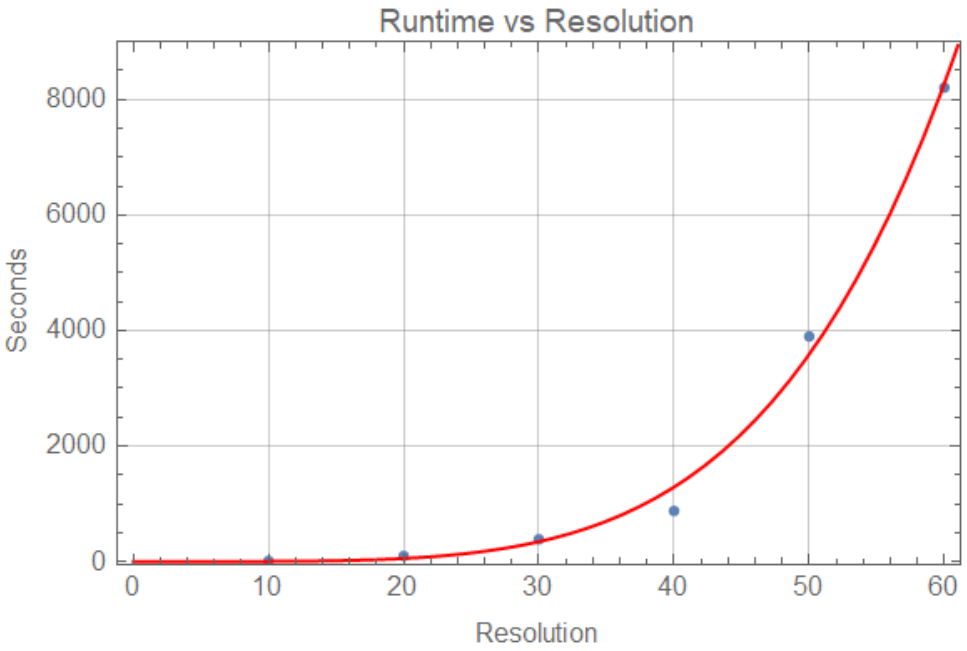

Figure 3.2 Runtime plotted against resolution for a resonator with an aspect ratio of 1 and a relative permittivity of 30. The red curve is a simple polynomial function fitted to the data, which has the form $t = 0.0000577x^{4.58731}$. The runtime for the resolution set equal to 60 was 2 hours and 16 minutes.

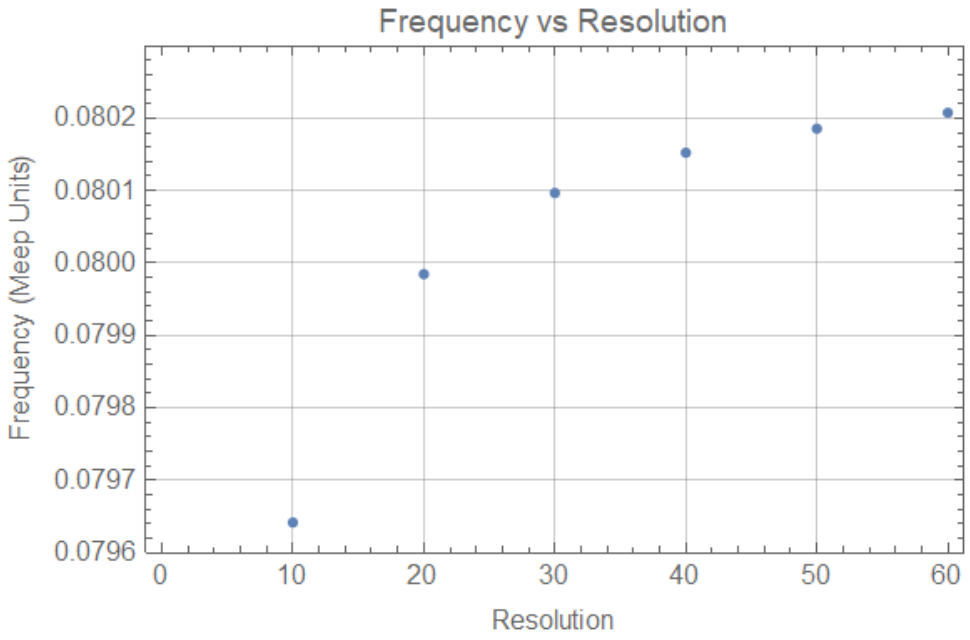

Figure 3.3 Calculated frequency values for different resolutions for a resonator with an aspect ratio of 1 and a relative permittivity of 30.

3.3 Single Resonator Frequency Accuracy

Equally, if not more important than finding the electric and magnetic fields, is finding the resonant frequency. Ideally, we would like to see the simulation resolution have no impact on the calculated frequency, but unfortunately this is not quite the case. What we find is that typically the simulation tends to slightly undershoot what the real frequency should be, and as resolution is increased it asymptotically approaches this true value. Fig 3.3 shows this with the calculated resonant frequency (in Meep units–see Chapter 2) for the same configuration and resolutions as before. The difference is very slight–between resolutions 30 and 60, for example, there is only a difference of 0.13%.

Now, it would be possible to fit these points to an exponential curve to estimate the calculated frequency as resolution approaches infinity. However, we found this approach to be unwieldy and difficult to implement across many different resonator configurations. Instead, we elected to use a technique called Aitken extrapolation, which is a method that allows one to estimate the value at

which a geometric sequence converges [14]. The formula is given as follows:

$$S_\infty = S_{n+2} - \frac{(S_{n+2} - S_{n+1})^2}{S_n - 2S_{n+1} + S_{n+2}} \tag{3.1}$$

where S_n, S_{n+1}, and S_{n+2} are consecutive terms in a geometric sequence. Since terms in a geometric sequence must increase by a constant multiplicative factor, we can use the calculated frequency values at resolutions of 10, 20, and 40 for our S_n, S_{n+1}, and S_{n+2}, respectively. Inputting these into the above formula will then give us an estimation of the resonant frequency as if it were calculated at a resolution of infinity.

To determine whether or not this was an accurate approach, we simulated the same configuration as before using these resolutions; however, this time we removed the PML boundary layers to simulate a dielectric resonator inside of a conductive cavity. This allowed us to perform the same simulation in FEniCS, and then compare the resulting frequencies. The results of this are plotted in Fig 3.4. What we found was that the Aitken extrapolated Meep frequency is in almost exact agreement with the FEniCS frequency, with a percent difference of only 0.008% between the two. Because of this, we believe our Meep code combined with Aitken extrapolation can be used to obtain accurate frequency estimates.

3.4 Frequency Results Across Parameter Space

Given that we have now established the limitations of our simulation code, we can now move on to testing a variety of different resonator configurations, both within and beyond the ranges that our approximate formula (Eq. 2.5) is said to be accurate within. To do this, we varied our aspect ratio from 0.25 to 4, and our relative permittivity from 20 to 50, and performed simulations at a large sampling across this space. All configurations were simulated at resolutions of 10, 20, and 40, and then Aitken extrapolation was used to obtain the final values. A plot of all the calculated

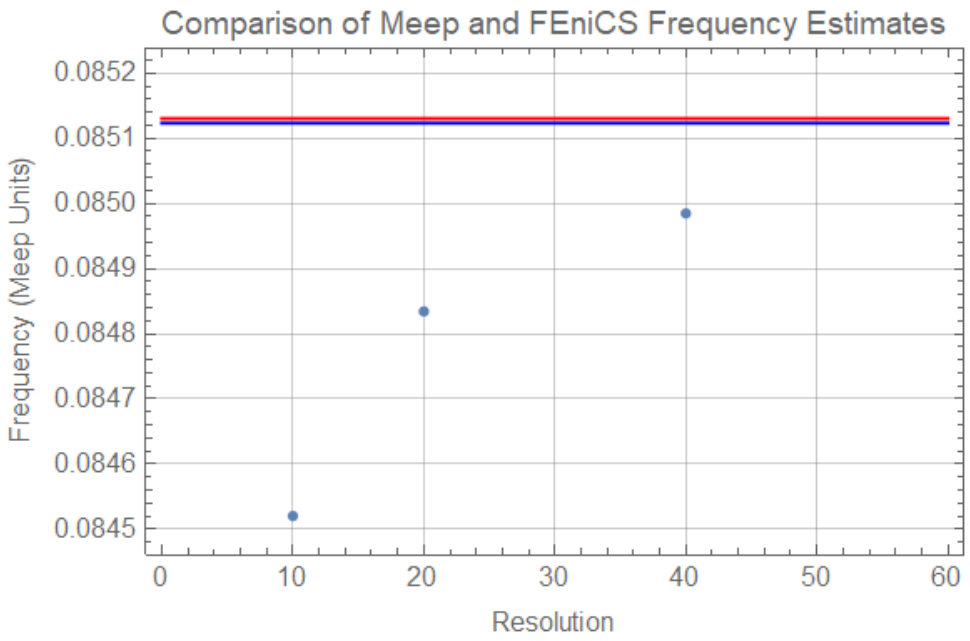

Figure 3.4 Comparison of Meep and FEniCS frequency values for a resonator with an aspect ratio of 1 and a relative permittivity of 30, inside a conductive cavity with a diameter and height of 4. The blue points represent the raw Meep values, and the blue line represents the Aitken extrapolated value at infinity. The red line represents the calculated FEniCS value for the frequency.

resonant frequencies is shown in Fig. 3.5. For clarity, a cross section of this plot is shown in Fig. 3.6. Fig. 3.7 shows the error between the formula and calculated frequencies; here the height above the $z = 0$ plane corresponds to the percent error of the approximate formula from the calculated frequency. At an aspect ratio of 1 and a relative permittivity of 30, which is at the very edge of the range the approximate formula is said to be accurate within, the percent difference is 2.38%, more or less verifying the accuracy claims of this formula. One thing that is concerning, however, is the alternating behavior between overestimation and underestimation seen at higher aspect ratios, as well as the aspect ratio of 1.2 having a much higher error than the surrounding ratios that were tested. We believe this is most likely due to some kind of flaw in our computational approach that has yet to be addressed, or even a mathematical error. Further research is required to determine what the cause of this behavior is.

3.5 Future Work

The work of this thesis was primarily to establish this simulation method as a viable way of determining the fields and resonant frequency of the $TE_{01\delta}$ mode of a dielectric resonator in free space. This lays the groundwork for several avenues of potential work in the future. Firstly, we can use this data to create a more accurate function to estimate the resonant frequency for a single, solid resonator; a function that also could be used across a larger range of relative permittivities and aspect ratios.

More importantly, however, is the potential to use this work in combination with a neural network, similar to the previous work done by this group. This type of simulation can easily be modified to work for a hollow dielectric resonator, or a set of stacked resonators. Doing this introduces new degrees of freedom that makes it impossible to create a simple function that can estimate the resonant frequencies. However, this is the type of task a neural network is well suited

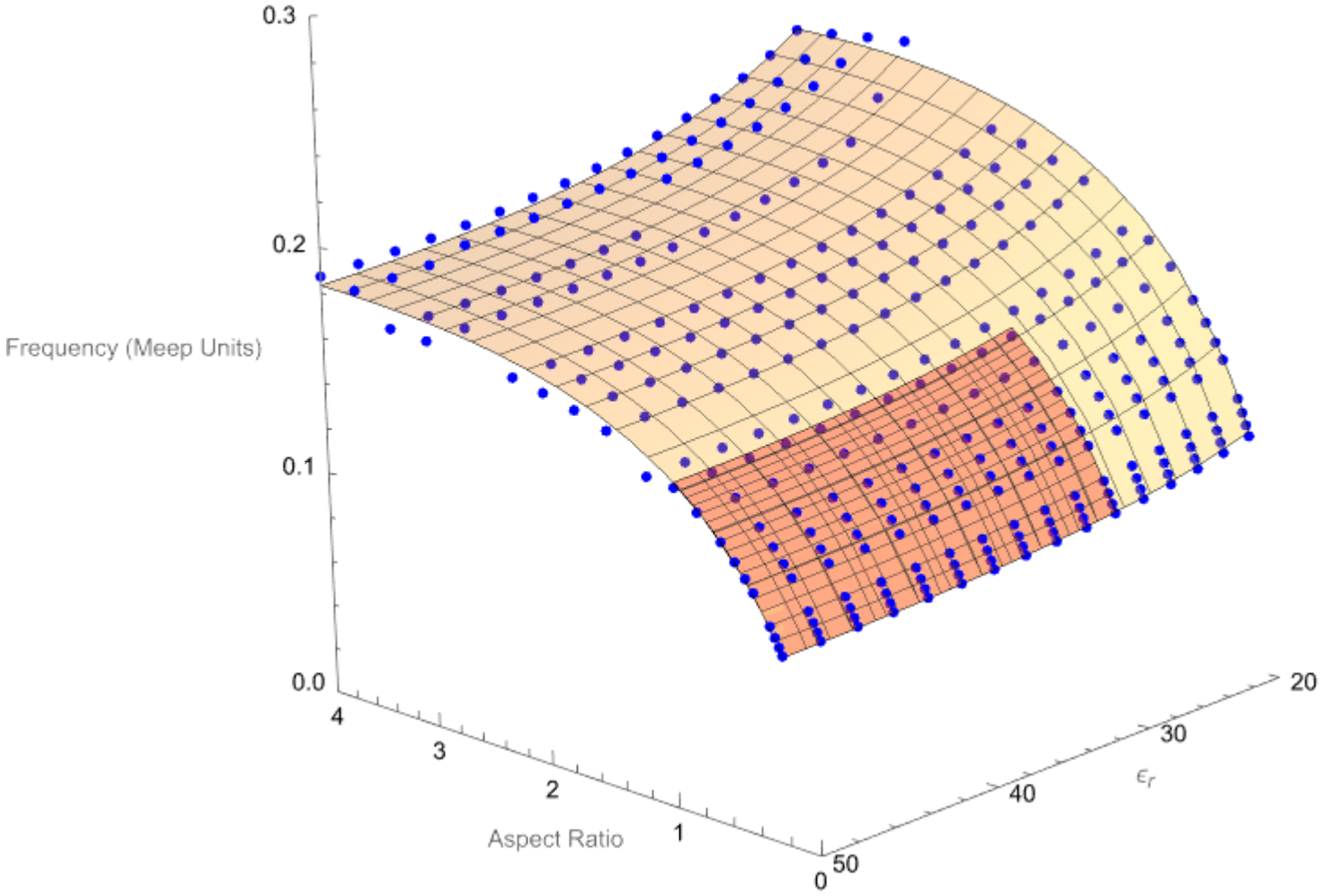

Figure 3.5 3D plot of all calculated frequencies, as a function of both aspect ratio and relative permittivity. The overlaid surface plot is the estimation provided by Eq. 2.5 (with the harmonic mean set to 1), and the red section of the surface plot indicates the region within which the formula is accurate to within 2%.

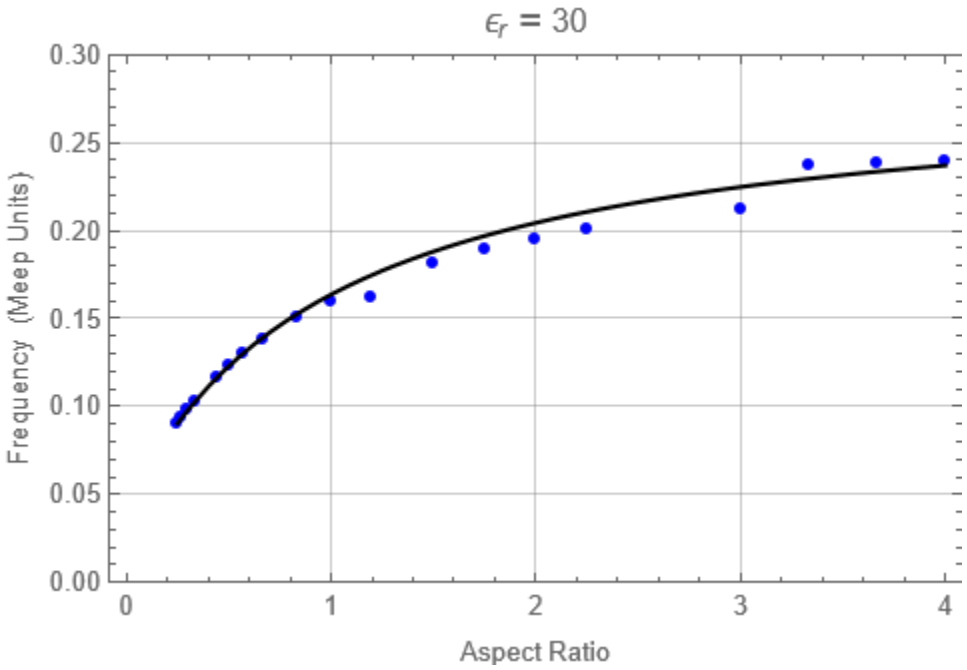

Figure 3.6 Cross section of Figure 3.5, for a relative permittivity of 30.

to handle, given a large enough dataset to train with, which is what we can provide using the BYU supercomputer.

To pursue these avenues, we will also attempt to better quantify and address the sources of error in our simulations, in addition to testing resolutions at a wider variety of configurations, and also finding established frequency values in literature with which we can compare our results with. We are confident that with continued work, we can achieve similar accuracy in our results to our previous work with resonators enclosed inside conductive cavities, across potentially an even wider range of configurations. Given these possibilities, we are excited to continue to explore this project, and find ways to contribute and expand our collective understanding of these materials.

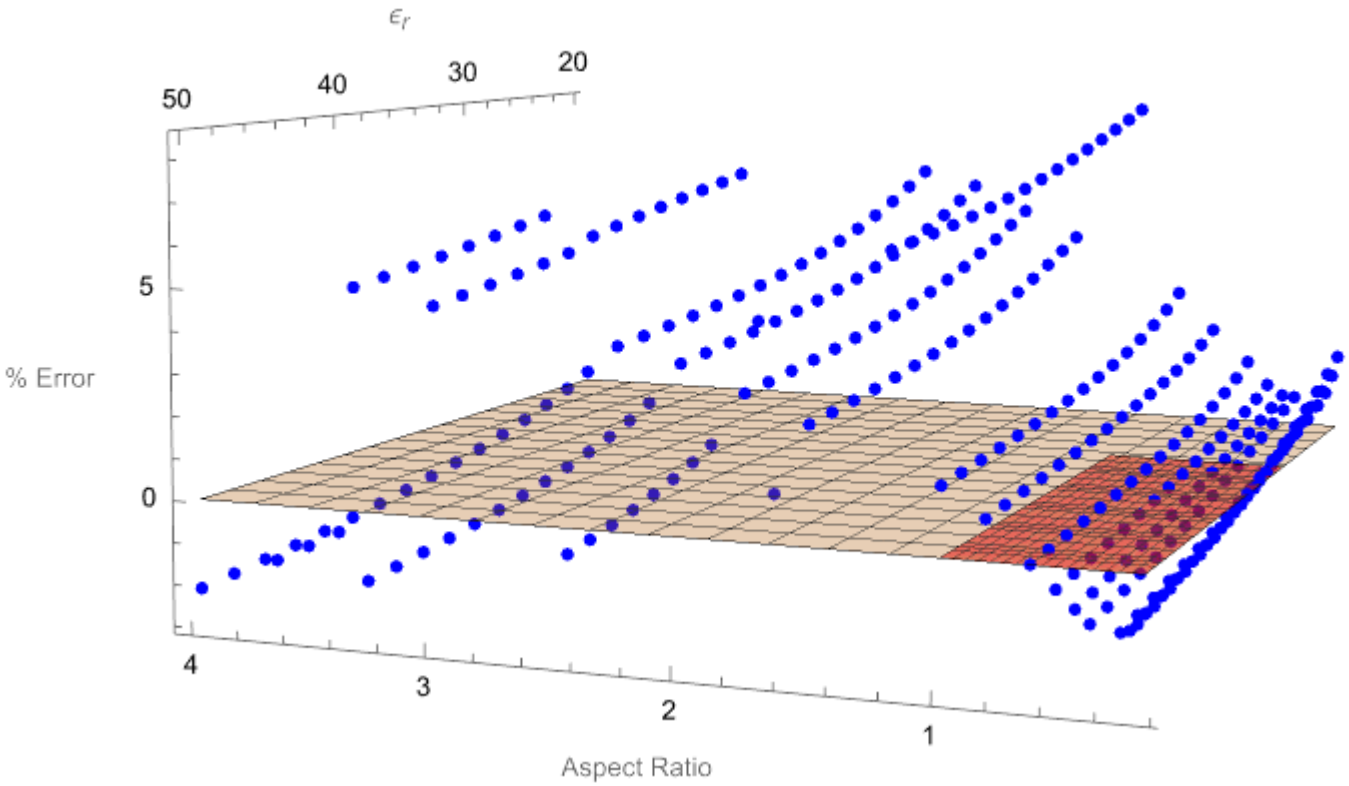

Figure 3.7 Percent difference between calculated frequency and frequency predicted by the approximate formula. The $z = 0$ plane has been displayed for convenience, with the red section corresponding to the region within which the formula is said to be accurate to within 2%.

Appendix

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import meep as mp
import sys
import numpy as np
import timeit
import csv

oR = float(sys.argv[1]) #dielectric outer radius
h = float(sys.argv[2]) #dielectric height
iR = float(sys.argv[3]) #dielectric inner radius
diel = float(sys.argv[4]) #dielectric constant
sim_r = float(sys.argv[5]) #simulation radius
sim_h = float(sys.argv[6]) #simulation height
dpml = float(sys.argv[7]) #PML thickness
resolution = int(sys.argv[8]) #simulation resolution
output_file_base = str(sys.argv[9]) #output file name

if __name__ == "__main__":
    #mp.verbosity(0) #uncomment this command if you want to reduce the amount of log text output

    sr = sim_r + dpml #total simulation radius
    sh = sim_h + 2*dpml #total simulation height

    t0 = timeit.default_timer() #start timer

    max_attempts = 5
    attempt_counter = 0
    is_te011 = False
    while((not is_te011) and (attempt_counter < max_attempts)):

        dimensions = mp.CYLINDRICAL
        cell = mp.Vector3(sr, 0 , sh)
        geometry = [mp.Block(center=mp.Vector3(iR + ((oR-iR)/2), 0, 0),
                             size=mp.Vector3((oR-iR), 1e20, h),
                             material=mp.Medium(epsilon=diel))]
```

```
    pml_layers = [mp.PML(dpml)]
    freq = (0.113413 * (3.45 + oR / h))/(oR * np.sqrt(diel))
    freq = freq - (0.1 * freq * attempt_counter)
    #this guess frequency used for the source is often slightly above the resonant frequency,
    #so if the simulation fails with the first guess frequency,
    #it tries again with a slightly lower guess frequency. This rarely happens though

    src = [mp.Source(mp.ContinuousSource(frequency = freq),
                     component = mp.Bz,
                     center = mp.Vector3(0, 0, 0),
                     size = mp.Vector3(0, 0, 0))]

    sim = mp.Simulation(cell_size = cell,
                        geometry = geometry,
                        boundary_layers = pml_layers,
                        resolution = resolution,
                        sources = src,
                        dimensions = dimensions,
                        m = 0, #Angular mode number, which is 0 for us
                        force_complex_fields = True)
    try:
        sim.init_sim()
        eigfreq = sim.solve_eigfreq(tol=1e-7, maxiters=100,
                                    cwtol=1e-10, cwmaxiters=10000, L=10)
        #these are just the default settings for solve_eigfreq, written out for clarity

        ep_data = sim.get_array(component = mp.Ep).real #get Ephi array

        #trim array to exclude PML boundaries plus a little extra
        ep = np.copy(ep_data[int(resolution*1.5*dpml):int(resolution*sh-resolution*1.5*dpml),
                             :int(resolution*sr-resolution*1.5*dpml)])
        if(np.abs(np.amin(ep)) > np.abs(np.amax(ep))): #flip Ephi if negative
            ep = -1 * ep

        #perform mode check (note: this isn't perfect)
        tol = 0.01
        if ((-np.amin(ep) * tol) < np.amax(ep)): #checks for zero crossings in the E field.
            is_te011 = True


    except Exception as exception:
        eigfreq = -1 #frequency value to report that indicates error
        is_te011 = False
        print(exception)

    attempt_counter += 1

tf = timeit.default_timer() #end timer
total_time = tf - t0
```

```
print("Total time is : ", str(total_time) + " s") #print total time

#generating output array
output = np.zeros(13)
output[0] = h/(2*oR)  #(aspect ratio)
output[1] = oR
output[2] = h
output[3] = iR
output[4] = diel
output[5] = sim_r
output[6] = sim_h
output[7] = dpml
output[8] = resolution
output[9] = attempt_counter
output[10] = total_time

#add frqeuency value to frequency array
if (is_te011):
    output[11] = eigfreq.real
    output[12] = eigfreq.imag
else:
    output[11] = -1
    output[12] = -1

output_file = output_file_base + ".csv"

with open(output_file, 'a') as f: # 'a' = append
    writer = csv.writer(f)
    writer.writerow(output) #array is saved as a single row in the specified .csv file
```

Bibliography

[1] R. D. Richtmyer, "Dielectric Resonators," Journal of Applied Physics **10,** 391–398 (2004).

[2] D. Kajfez and P. Guillon, *Dielectric Resonators*, *Artech House microwave library* (Noble Publishing Corporation, 1998).

[3] J. Plourde and C.-L. Ren, "Application of Dielectric Resonators in Microwave Components," IEEE Transactions on Microwave Theory and Techniques **29,** 754–770 (1981).

[4] C. Lewis, J. Bryan, N. Schwartz, J. Hale, K. Fanning, and J. S. Colton, "Machine Learning to Predict Quasi TE011 Mode Resonances in Double-Stacked Dielectric Cavities," IEEE Transactions on Microwave Theory and Techniques **70,** 2135–2146 (2022).

[5] W. Schoenmaker, in *Computational Electrodynamics : A Gauge Approach with Applications in Microelectronics*, S. Gay and A. Ravara, eds., (River Publishers, Aalborg, 2017), iD: 4874048.

[6] J. N. Reddy, *An Introduction to the Finite Element Method* (McGraw-Hill, New York, 1984).

[7] A. Taflove and S. Hagness, *Computational Electrodynamics: The Finite-difference Time-domain Method*, *Artech House antennas and propagation library* (Artech House, 2005).

[8] J.-P. Berenger, "A perfectly matched layer for the absorption of electromagnetic waves," Journal of Computational Physics **114,** 185–200 (1994).

[9] S. G. Johnson, "Notes on Perfectly Matched Layers (PMLs)," CoRR abs/2108.05348 (2021).

[10] F. Teixeira and W. Chew, "Systematic derivation of anisotropic PML absorbing media in cylindrical and spherical coordinates," IEEE Microwave and Guided Wave Letters **7,** 371–373 (1997).

[11] A. F. Oskooi, D. Roundy, M. Ibanescu, P. Bermel, J. Joannopoulos, and S. G. Johnson, "Meep: A flexible free-software package for electromagnetic simulations by the FDTD method," Computer Physics Communications **181,** 687–702 (2010).

[12] MIT, "Meep Documentation,", https://meep.readthedocs.io/en/latest/, accessed 2023-04-19.

[13] MIT, "Introduction,", https://meep.readthedocs.io/en/latest/Introduction/, accessed 2023-04-19.

[14] K. G. Miller, M. Meehan, R. L. Spencer, and J. S. Colton, "Resonance of Complex Cylindrically Symmetric Cavities Using an Eigenfunction Expansion in Empty Cavity Modes," IEEE Transactions on Microwave Theory and Techniques **64,** 3113–3120 (2016).

Index

analytic continuation, 6

coordinate system, 8

dielectric resonator, 1

FEniCS, 6
finite difference method, 4
finite element method, 3
flux, 13

Meep, 7
Meep Units, 7
microwave cavity, 2

neural network, 20

perfectly matched layers, 5

resolution, 15

source function, 9

$TE_{01\delta}$, 2, 4
transformation into complex space, 6