CRITICAL ANALYSIS OF A PRACTICAL FOURTH ORDER FINITE-DIFFERENCE TIME-DOMAIN ALGORITHM FOR THE SOLUTION OF MAXWELL'S EQUATIONS

by

Antonio P. Thomson

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Electrical Engineering).

Golden, Colorado

Date ____________________

Signed: ____________________

Antonio P. Thomson

Signed: ____________________

Dr. Atef Elsherbeni
Thesis Advisor

Golden, Colorado

Date ____________________

Signed: ____________________

Dr. Peter Aaen
Department Head
Department of Electrical Engineering

ABSTRACT

The finite-difference time-domain (FDTD) method is a highly effective numerical method of solving Maxwell's equations in the time domain. Traditionally the approximation of the derivatives in Maxwell's equations is based on a central differencing scheme which is second order accurate (second order). The high complexity of today's electromagnetic problems necessitate a FDTD formulation that can use less computational memory and complete simulations faster than current second order FDTD formulations. Many researchers have studied the benefits of FDTD formulations based on fourth order approximations of the spatial derivatives (fourth order). However, none has presented a complete non-specialized case that leads to the simulation of practical antenna or electromagnetic problems. For this reason, and due to the complexity of the available fourth order formulations and the lack of comprehensive analysis of such FDTD formulations, none of the existing commercial electromagnetic software packages use any fourth order FDTD formulations for solving practical problems.

The goal of this thesis is to implement, validate, and provide performance analysis of a practical FDTD scheme using fourth order accurate central differencing derivative approximations in space and second order accurate central differencing derivative approximations in time. The simplicity of the fourth order formulation presented in this thesis comes from that fact that it is derived from Taylor series expansions of a general function. The formulation of the FDTD updating equations is developed for general mediums as well as lumped circuit elements (voltage sources, resistors, capacitors, inductors, and diodes). Additionally, updating equations for fourth order convolutional perfectly matched layers (CPML) are derived. This formulation is straightforward, advantageous, and provides a practical fourth order FDTD formulation for electromagnetics applications.

Verification and simulation accuracy of the developed fourth order formulation are confirmed through the application of Gaussian propagation, a cavity resonator, the radiation from a dipole antenna, antenna arrays, and the radar cross section calculation of a dielectric cube. Simulations of discontinuous boundaries are also explored in detail through the simulation of PEC objects and high permittivity objects. Various different methods of special fourth order updating equations are thoroughly tested at these boundaries and the results are analyzed. The computational advantages of the developed fourth order FDTD formulation are explored and results show reduced memory usage up to a factor of 6.97 and reduced simulation time up to a factor of 8.70 compared to the traditional second order FDTD formulation.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

General Nomenclature

Electric field	E
Magnetic field	H
The wavelength of an electromagnetic wave	λ
Current density	J
Current	I
Material permeability	μ
Permeability of free space	μ_0
Relative permeability	μ_r
Material permittivity	ϵ
Permittivity of free space	ϵ_0
Relativel permittivity	ϵ_r
Material conductivity	σ
Capacitance	C
Inductance	L
Resistance	R

LIST OF ABBREVIATIONS

Colorado School of Mines	CSM
Convolutional Perfectly Matched Layer	CPML
Core Processing Unit	CPU
Courant-Friedrich-Lewy	CFL
Electromagnetic	EM
Finite Difference Time Domain	FDTD
Graphics Processing Unit	GPU
Method of Moments	MoM
Perfect Electric Conductor	PEC
Perfectly Matched Layer	PML
Radar Cross Section	RCS
Transverse Electric	TE
Transverse Magnetic	TM

ACKNOWLEDGMENTS

I would like to thank everyone that made this thesis a possibility. This project would not have suceeded without Dr. Atef Z. Elsherbeni's contiued support and helpful advice thoughout. I would also like to thank Dr. Mohammed F. Hadi for his guidance involving the basics of higher order FDTD, Jennifer Thomson for her grammatical review, and Katie Thomson, my wife, for her support while working on this project. The research in this Thesis was partially supported by the gift funds from Futurewei Technology Inc., New Jersey Research Center, Bridgewater, NJ, USA.

CHAPTER 1
INTRODUCTION

The finite difference time domain (FDTD) method is a highly effective method of numerically solving Maxwell's equations in the time domain [1]. The standard derivative approximation is a second order accurate central differencing scheme for all derivatives in Maxwell's equations (second order) [1]. This thesis will present the implementation for a practical FDTD scheme using fourth order accurate central differencing derivative approximations in space and second order accurate central differencing derivative approximations in time (fourth order). Fourth order FDTD simulations allow the cell size of the simulation to grow while maintaining necessary solution accuracy. Larger cell sizes are imperative when geometries become electrically large and the computational memory becomes too excessive. Many other papers have studied the benefits of fourth order FDTD, but none have simulated practical antenna problems with a simple formulation [2], [3], [4], [5], [6]. The goal of this paper is to present a fourth order formulation that is straightforward and at the same time can simulate practical problems.

1.1 Maxwell's Equations

The starting place to solve any problem involving electrodynamics is Maxwell's equations. In computational electromagnetics, Maxwell's equations are solved by approximating them in such a way to allow a computer to evaluate them. Equations 1.1, 1.2, 1.3 and 1.4 show the mathematical form of Maxwell's equations in the time domain

$$\bigtriangledown \times \overrightarrow{H} = \frac{\partial \overrightarrow{D}}{\partial t} + \overrightarrow{J} \tag{1.1}$$

$$\bigtriangledown \times \overrightarrow{E} = -\frac{\partial \overrightarrow{B}}{\partial t} + \overrightarrow{M} \tag{1.2}$$

$$\bigtriangledown \cdot \overrightarrow{D} = \rho_e \tag{1.3}$$

$$\bigtriangledown \cdot \overrightarrow{B} = \rho_m \tag{1.4}$$

where $\overrightarrow{E}$ is the electric field vector in volts per meter, $\overrightarrow{D}$ is the electric displacement vector in coulombs per square meter, $\overrightarrow{H}$ is the magnetic field strength vector, $\overrightarrow{B}$ is the magnetic flux density vector in Tesla, $\overrightarrow{J}$ is the electric current density in amperes per square meter, $\overrightarrow{M}$ is the magnetic current density vector in volts per square meter, ρ_e is the electric charge density, and ρ_m is the magnetic charge density.

Material properties of certain mediums for which electromagnetic fields pass through are accounted for with the following relations:

$$\overrightarrow{D} = \epsilon \overrightarrow{E} \tag{1.5}$$

$$\overrightarrow{B} = \mu \overrightarrow{H} \tag{1.6}$$

where ϵ is the permitivity of the material and μ is the permeability of the material. Vacuum permitivity and permeability are as follows:

$$\epsilon_0 = 8.854 \times 10^{-12} F/m \tag{1.7}$$

$$\mu_0 = 4\pi \times 10^{-12} H/m. \tag{1.8}$$

Furthermore, $\overrightarrow{J}$ is the sum of the conduction current ($\overrightarrow{J_c} = \sigma^e \overrightarrow{E}$) and the impressed current J_i. Similarly, $\overrightarrow{M}$ is the sum of the conduction magnetic current ($\overrightarrow{M_c} = \sigma^m \overrightarrow{H}$) and M_i. Using these equalities, the final Maxwell equations involving curl are presented as equations 1.9 and 1.10.

For the FDTD method, only the Maxwell equations involving curl are needed. From the vector form (as shown in equations 1.9 to 1.16) it is necessary to break them up into component form so they can be solved numerically.

$$\bigtriangledown \times \overrightarrow{H} = \sigma^e \overrightarrow{E} + \epsilon \frac{\partial \overrightarrow{E}}{\partial t} + \overrightarrow{J}_i \tag{1.9}$$

$$\bigtriangledown \times \overrightarrow{E} = -\sigma^m \overrightarrow{H} - \mu \frac{\partial \overrightarrow{H}}{\partial t} - \overrightarrow{M}_i \tag{1.10}$$

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right) \tag{1.11}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon_y}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma_y^e E_y - J_{iy}\right) \tag{1.12}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon_z}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma_z^e E_z - J_{iz}\right) \tag{1.13}$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right) \tag{1.14}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu_y}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - \sigma_y^m H_y - M_{iy}\right) \tag{1.15}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right) \tag{1.16}$$

In the FDTD method, equations 1.11 to 1.16 are solved directly by using numerical derivative approximations. These numerical derivative approximations are applied over a domain discretized by a finite number of points in space. In order for the derivative approximations to be accurate, the distance between these discrete points needs to be small. For electrically large 3D problems, the number of points can be large and require extensive computational resources.

1.2 Advantages of FDTD

FDTD is advantageous for solving Maxwell's equations because it provides a full-wave time-domain solution. Many other EM solving techniques solve Maxwell's equations in the frequency domain, such as the Method of Moments (MoM). MoM is advantageous in that is requires less computational resources than FDTD, but it only solves the problem one frequency at a time. Many MoM simulations must be completed to characterize a problem in a wide frequency band. Since FDTD produces a time-domain solution, only one FDTD simulation is needed and then a Fourier transform of the results can be taken to evaluate the results for many frequencies of interest.

1.3 The Need for Fourth Order FDTD

As noted, the major disadvantage of FDTD compared to other EM solvers is the computation resources FDTD requires. Problems in the RF and microwave industry today are becoming increasingly complicated and electrically large [7], which demand incredible amounts of computing resources. The limitation of the computer that is the most difficult to overcome is memory resources. In this thesis, the largest problem simulated is 150 by 150 by 150 cells, which is a total of 3.375 million cells. There are 6 arrays of electric and magnetic field components as well as 6 arrays of material property arrays that will each be 3.375 million elements large. Not to mention the few additional arrays needed for pre-processing and post-processing the simulation.

Of course, if the number of cells in the domain can be reduced, the size of these arrays can be reduced, and thus the computational memory is also reduced. Another way to think of reducing the number of cells is to increase the cell size. Increasing the cell size will decrease the accuracy of the geometric discretization and the derivative approximations. There is not much that can be done about the geometric discretization error, but the derivative approximation error can be reduced with a higher order derivative approximation. Most FDTD formulations use second order accurate derivative approximations in space and time. An improvement upon this would be second order derivative approximations in time and fourth order derivative approximations in space. A formulation such as this would be more accurate and could withstand a bigger cell size (and require less computational memory) than the standard second order formulation.

1.4 Previous Work in Higher Order FDTD

FDTD is still an active area of research. Even though the advancement of computers in recent years has made simulating electromagnetic problems with FDTD more practical, there is still a need to improve the FDTD formulation to decrease computational resources. In order to understand where FDTD research is going, an understanding of the current research is imperative. A comprehensive formulation and implementation of second order FDTD is presented in [1]. This book fully describes the derivation of second order updating equations, building objects on the Yee grid, sources and lumped element formulations, and near and far field output parameters. Additionally, the code detailed in [1] serves as the second order code that the fourth order code in this thesis is built from. The second order code in [1] also serves as the baseline second order code to compare the performance of the fourth order code to.

General fourth order FDTD formulations have been the focus of numerous other research efforts. One of the first and most commonly known fourth order FDTD formulations is presented by Fang [2]. The formulation and general updating equations are described, and the accuracy of the fourth order formulations is analyzed [2]. However, the updating equations are very complicated and difficult to integrate with an existing second order code. Additionally, the general update equations are not derived in a way that easily allows for the updating equations of sources, CPML, and lumped elements to be found.

Building on this initial research by Fang, Hwang and Cangellaris analyzed the accuracy of Fang's formulations and compared them to Yee's second order formulation [3]. The research also presented a one-sided fourth order formulation to handle an infinite PEC boundary [3].

Similarly, a simple fourth order formulation is presented in [8] and [9]. This fourth order formulation matches the formulation presented in this thesis but does not derive updating equations for general media, lumped circuit elements, or CPML. However, [9] does present some practical simulations using the fourth order FDTD formulation. They are able to accurately simulate and array of current sources and a cavity resonator problem with fourth order FDTD. In order to simulate more accurate problems with fourth order FDTD, they implement a fine mesh of second order FDTD to handle any antenna geometry [9]. Using this hybrid course fourth order mesh and fine second order mech technique, they are able to simulate the radiation of two monopole antennas on a ground plane [9]. In a third similar work, the hybrid course fourth order mech and fine second order mesh is sued to simulate the shielding effectiveness of a scaled Boeing 757 model [10].

Another general fourth order study building off of the work done by Fang is presented by Adel Abdin [11]. In this dissertation the equations formulated by Fang are again validated through the application of a cavity resonator and free space propagation [11]. The higher order simulation of microstrip structures are also explored by Abdin [11].

Other fourth order schemes have been formulated to optimize certain parameters. One such study by Hadi and Picket-May presents a fourth order FDTD formulation that can be tuned to significantly decrease phase error at a certain frequency [4]. This paper analyzes the modified fourth order formulation and shows that is it advantageous over the standard second and forth order formulations presented by Fang. However, the advantages of the formulation presented in [4] are most significant over a narrow frequency bandwidth. As one of the advantages of FDTD in general is the ability to solve wide bandwidth problems, optimizing it for a specific frequency illuminates a lot of the reason to use FDTD in the first place. Finally, the fourth order formulation in [4] does not show in detail the formulation for fourth order sources, lumped elements, or CPML. In the paper, second order buffers are used to simulate the region where any geometry of the problem is defined and to simulate the domain terminating PML region. A similar FDTD formulation is presented by El-Hefnawi et. al. in [12]. This formulation uses the same general updating equations as [4] and leaves similar areas for improvement.

A detailed analysis of higher order FDTD schemes was done by Kantartzis and Tsiboukis [5]. Their book summarizes many different fourth order schemes. Many of them are practical for certain applications, such as reducing phase error at a certain frequency, modeling dielectric boundaries, working in non-Cartesian coordinate systems, and even applications of subgridding [5]. However, most of the formulations in this book are rather specialized for a specific application, overly complicated, or a combination of both. None of the higher order FDTD formulations existing in literature are general enough to handle all kinds of electromagnetic problems demanded by the RF and microwave industry.

One of the key challenges with any fourth order formulation is the handling of PEC and dielectric boundaries. Since fourth order updating equations have a larger "mask" or "stencil" than second order equations, boundaries are not represented correctly. To solve this issue, Yang et. al. presented a "symmetric image" approach to ensure the larger stencil of the fourth order updating equations do not cause numerical errors at boundaries [13]. The approach presented in [13] is tailored for the same fourth order derivative approximation used in this thesis.

Other higher order FDTD formulations have been presented by Hadi that handle the material boundaries in different ways [14]. This work by Hadi details a method of handling PEC objects while using the specialized "M24" algorithm [14]. Another approach to handle PEC objects with higher order methods is presented by Shang et. al. [15]. This work derives a higher order FDTD scheme using compact differencing to avoid the larger mask of the fourth order formulation presented in this thesis [15]. However, this is a new derivation of FDTD and will not simply integrate with the existing second order FDTD formulations that have been previously fully defines [1].

Perhaps one of the most promising techniques for handling PEC and other interfaces in a higher order FDTD formulation is one-sided differences [16]. Prokopidis and Tsboukis present an elegant way to use fourth order forward or backward differencing, rather than central differencing, at PEC and material interfaces [16]. A similar work, again by Prokopidis and Tsboukis details the accuracy of using simulating lossy dielectrics with a fourth order formulation employing one sided differences [17]. Another study by Yefet and Petropoulos presents a very similar one-sided difference formulation for the handling of PEC and dielectric interfaces [18]. The analysis in [18] is also of interest as it uses a matrix approach to updating the FDTD fied components. Although these techiques are promising, they are not easy to integrate into a FDTD code capable of simulating many different geometries such as presented in [1]. This thesis will attempt a special treatment of PEC objects in the fourth order FDTD code that only slightly modifies the derived updating equations.

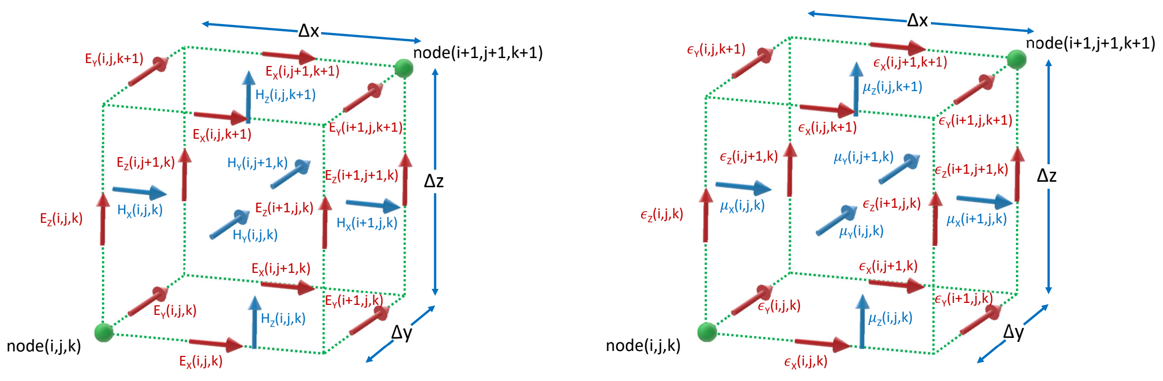
1.5 Goals of this Thesis

The goal of this thesis is to validate and provide performance analysis of a straightforward, advantageous, and practical fourth order FDTD formulation. The formulation presented in this thesis is very simple and can be easily integrated into an existing second order FDTD code. The fourth order updating equations take the same form as the second order FDTD formulation presented in [1]. The only difference between the second and fourth order updating equations is the derivative approximations. Once the fourth order FDTD formulation is complete, analysis comparing the fourth order formulation to the second order simulation will be completed. The goal of such an analysis is to show fourth order FDTD is computationally advantageous due to its increased accuracy. Since the accuracy of the fourth order simulation is higher, the desired response is a larger cell size can be used with the fourth order FDTD simulations compared to the cell size of the second order FDTD simulations. This increase in cell size by using this fourth order FDTD formulation would decrease computational time and computational memory requirements of the simulation. Finally, the ability of the fourth order FDTD formulation presented in this thesis to simulate practical electromagnetic problems will be evaluated. Problems such as radar cross section, thin wire dipole antennas, and antenna arrays will be simulated with the fourth order FDTD formulation.

CHAPTER 2

FOURTH ORDER FDTD UPDATING EQUATIONS FORMULATION

The formulation of 3D updating equations for fourth order FDTD start with understanding the Yee grid [19]. Just as in second order FDTD, the simulation domain must be descritized in a finite mannor. Both the electromagnetic field components and the material properties within the domain need to be descritized. In this paper, the FDTD domain is descritized according to the Yee grid, which is composesd of many Yee cells. Figure 2.1 shows how the field components and material properties are positioned on the Yee cell.



(a) (b)

Figure 2.1 The field component locations and the material component locations in a Yee cell.

Figure 2.1(a) shows the component locations and Figure 2.1(b) shows the material component locations in a Yee cell. It should be noted that many of these Yee cells are tiled together to form the Yee grid. 2D sections of a full Yee grid are shown in Figure 2.11 and Figure 2.14.

2.1 Finite Difference Approximations

In order to solve Maxwell's equations numerically, a general method for solving differential equations numerically must be used. In this thesis, finite differencing (specifically central differencing) is used to numerically approximate derivatives. It has been shown that simple central differencing is second order accurate and can be formulated to be fourth order accurate [1]. In sections 2.1.3 and 2.1.4 the derivations for general fourth order accurate central differencing and fourth order accurate central differencing specific to the FDTD code are shown. In section 2.1.5 the increased accuracy of fourth order central differencing is demonstrated.

2.1.1 General Second Order Accuracy

The starting place for the second order accurate finite differencing scheme is the general Taylor series expansion of a function. The most simple second order accurate finite differencing scheme is known as central differencing. Central differencing uses points on either side of the point of interest to approximate the function derivative. For this reason, Taylor series for $+\Delta x$ and for $-\Delta x$ are both used. Figure 2.2 explains in detail:

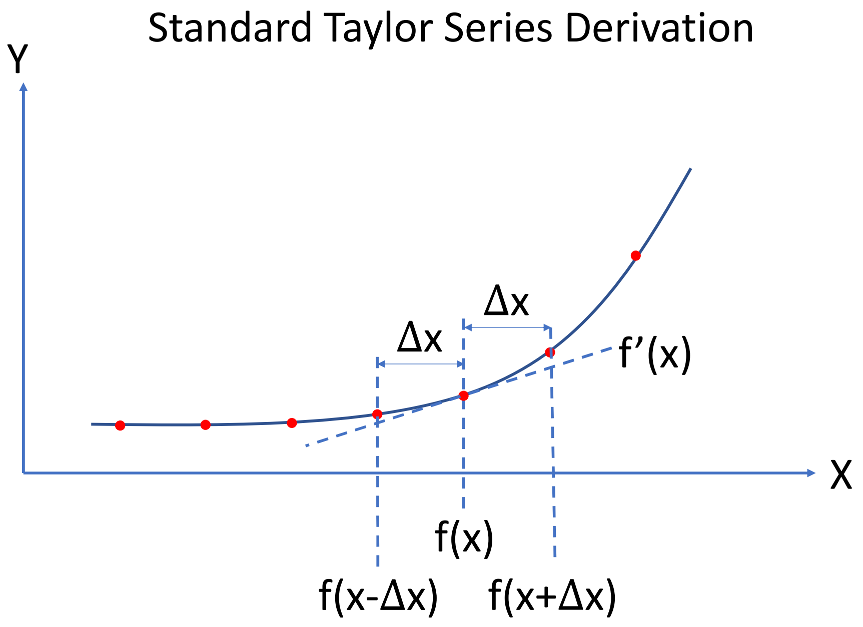


Figure 2.2 General Taylor series centered at x and a spacing of Δx from the center.

The general Taylor series for $+\Delta x$ is:

$$f(x+\Delta x) = f(x) + \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) + \frac{(\Delta x)^3}{6} f'''(x) + \frac{(\Delta x)^4}{24} f^{iv}(x) + \frac{(\Delta x)^5}{120} f^{v}(x) + ... \tag{2.1}$$

while the general Taylor Series for $-\Delta x$ is:

$$f(x-\Delta x) = f(x) - \Delta x f'(x) + \frac{(\Delta x)^2}{2} f''(x) - \frac{(\Delta x)^3}{6} f'''(x) + \frac{(\Delta x)^4}{24} f^{iv}(x) - \frac{(\Delta x)^5}{120} f^{v}(x) + ... \tag{2.2}$$

by subtracting the general Taylor Series for $+\Delta x$ from the general Taylor Series for $-\Delta x$ we have:

$$f(x-\Delta x)-f(x+\Delta x)=(f(x)-f(x))+(-\Delta xf'(x)-\Delta xf'(x))+(\frac{(\Delta x)^2}{2}f''(x)-\frac{(\Delta x)^2}{2}f''(x))$$
$$+(-\frac{(\Delta x)^3}{6}f'''(x)-\frac{(\Delta x)^3}{6}f'''(x))+(\frac{(\Delta x)^4}{24}f^{iv}(x)-\frac{(\Delta x)^4}{24}f^{iv}(x))+(-\frac{(\Delta x)^5}{120}f^{v}(x)-\frac{(\Delta x)^5}{120}f^{v}(x))+...... \tag{2.3}$$

by collecting like terms and solving for $f'(x)$ we get,

$$f(x-\Delta x)-f(x+\Delta x)=-2\Delta xf'(x)-\frac{(\Delta x)^3}{3}f'''(x))-\frac{(\Delta x)^5}{60}f^{v}(x)+... \tag{2.4}$$

$$f'(x)=\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}-\frac{(\Delta x)^2}{6\Delta x}-\frac{(\Delta x)^4}{120\Delta x}f^{v}(x)+... \tag{2.5}$$

after recognizing the $-\frac{(\Delta x)^2}{6\Delta x}-\frac{(\Delta x)^4}{120\Delta x}f^{v}(x)+...$ terms as error, the final approximation for $f'(x)$ is:

$$f'(x)\approx\frac{f(x+\Delta x)-f(x-\Delta x)}{2\Delta x}. \tag{2.6}$$

Equation 2.6 is second order accurate because the lowest order of Δx in the error terms is squared. Since Δx is small, $(\Delta x)^2$ is smaller and $(\Delta x)^4$ is even smaller. Since $(\Delta x)^2$ has the lowest exponent in the "error terms" the equation is said to be second order accurate.

2.1.2 Second Order Accuracy for FDTD

To derive the second order derivative approximations for FDTD, the general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $+\Delta x/2$ and a Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $-\Delta x/2$ will be subtracted from one another. Figure 2.2 shows how these points are located.

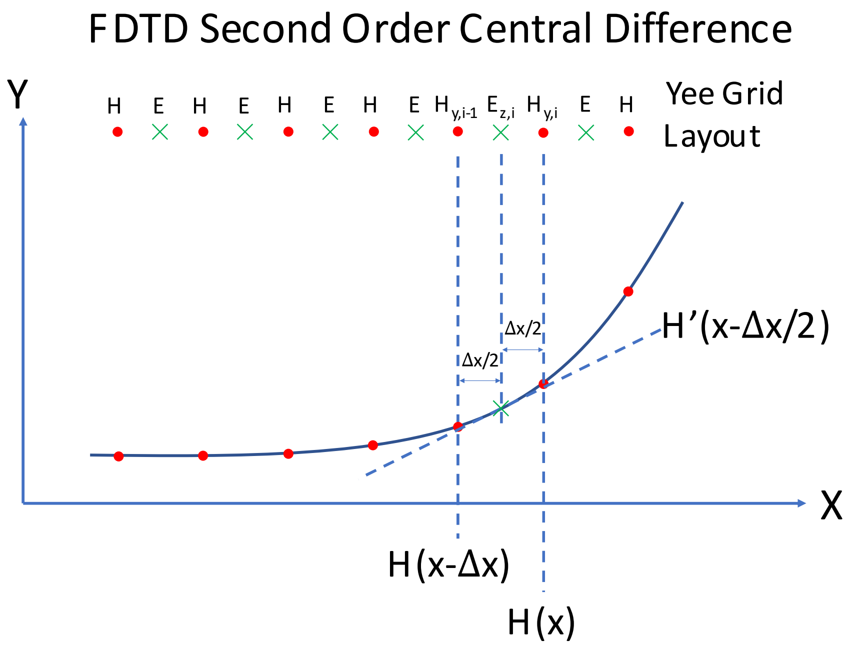

FDTD Second Order Central Difference
Y
H E H E H E H E H$_{y,i-1}$ E$_{z,i}$ H$_{y,i}$ E H
Yee Grid Layout
H'(x-Δx/2)
Δx/2
Δx/2
X
H(x-Δx)
H(x)

Figure 2.3 FDTD compliant Taylor series centered at $x + \Delta x/2$ and a spacing of $\Delta x/2$ from the center.

The general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $+\Delta x/2$ is as follows:

$$\begin{aligned} f\left(x-\frac{\Delta x}{2}+\frac{\Delta x}{2}\right) &= f\left(x-\frac{\Delta x}{2}\right)+\frac{\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right) \\ &+\frac{\left(\frac{\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\ldots \end{aligned} \tag{2.7}$$

$$\begin{aligned} f\left(x\right) &= f\left(x-\frac{\Delta x}{2}\right)+\frac{\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right) \\ &+\frac{\left(\frac{\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\ldots \end{aligned} \tag{2.8}$$

The general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $-\Delta x/2$ is as follows:

$$\begin{aligned} f\left(x-\frac{\Delta x}{2}-\frac{\Delta x}{2}\right) &= f\left(x-\frac{\Delta x}{2}\right)-\frac{\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right) \\ &-\frac{\left(\frac{\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)-\frac{\left(\frac{\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\ldots \end{aligned} \tag{2.9}$$

$$
\begin{aligned}
f(x-\Delta x) = f\left(x-\frac{\Delta x}{2}\right) - \frac{\Delta x}{2} f'\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^2}{2} f''\left(x-\frac{\Delta x}{2}\right) \\
-\frac{\left(\frac{\Delta x}{2}\right)^3}{6} f'''\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^4}{24} f^{iv}\left(x-\frac{\Delta x}{2}\right) - \frac{\left(\frac{\Delta x}{2}\right)^5}{120} f^{v}\left(x-\frac{\Delta x}{2}\right) + ...
\end{aligned} \tag{2.10}
$$

Now we can perform the following equation subtraction:

$$
\begin{aligned}
f(x) - f(x-\Delta x) = & \left[f\left(x-\frac{\Delta x}{2}\right) - f\left(x-\frac{\Delta x}{2}\right)\right] + \left[\frac{\Delta x}{2} f'\left(x-\frac{\Delta x}{2}\right) + \frac{\Delta x}{2} f'\left(x-\frac{\Delta x}{2}\right)\right] \\
& + \left[\frac{\left(\frac{\Delta x}{2}\right)^2}{2} f''\left(x-\frac{\Delta x}{2}\right) - \frac{\left(\frac{\Delta x}{2}\right)^2}{2} f''\left(x-\frac{\Delta x}{2}\right)\right] \\
& + \left[\frac{\left(\frac{\Delta x}{2}\right)^3}{6} f'''\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^3}{6} f'''\left(x-\frac{\Delta x}{2}\right)\right] \\
& + \left[\frac{\left(\frac{\Delta x}{2}\right)^4}{24} f^{iv}\left(x-\frac{\Delta x}{2}\right) - \frac{\left(\frac{\Delta x}{2}\right)^4}{24} f^{iv}\left(x-\frac{\Delta x}{2}\right)\right] \\
& + \left[\frac{\left(\frac{\Delta x}{2}\right)^5}{120} f^{v}\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^5}{120} f^{v}\left(x-\frac{\Delta x}{2}\right)\right] + ...
\end{aligned} \tag{2.11}
$$

$$
f(x) - f(x-\Delta x) = \Delta x f'\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^3}{3} f'''\left(x-\frac{\Delta x}{2}\right) + \frac{\left(\frac{\Delta x}{2}\right)^5}{60} f^{v}\left(x-\frac{\Delta x}{2}\right) + ... \tag{2.12}
$$

$$
\frac{f(x) - f(x-\Delta x)}{\Delta x} = f'\left(x-\frac{\Delta x}{2}\right) + \frac{\Delta x^2}{2^3 * 3} f'''\left(x-\frac{\Delta x}{2}\right) + \frac{\Delta x^4}{2^5 * 60} f^{v}\left(x-\frac{\Delta x}{2}\right) + ... \tag{2.13}
$$

Recognizing the $\frac{\Delta x^2}{2^3 * 3} f'''\left(x-\frac{\Delta x}{2}\right) + \frac{\Delta x^4}{2^5 * 60} f^{v}\left(x-\frac{\Delta x}{2}\right) + ...$ terms as error, the final second order derivative approximation can be written as follows:

$$
f'(x-\frac{\Delta x}{2}) \approx \frac{f(x) - f(x-\Delta x)}{\Delta x}. \tag{2.14}
$$

Equation 2.14 will be used to calculate the derivative of the magnetic field in order to update the electric field as shown in Figure 2.3. A similar analysis can be performed to calculate the derivative of the electric fields in order to update the magnetic fields. Equation 2.15 shows the derivative approximation in this case.

$$
f'(x+\frac{\Delta x}{2}) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x}. \tag{2.15}
$$

2.1.3 General Fourth Order Accuracy

The starting place for the fourth order accurate finite differencing scheme is the general Taylor series expansion of a function. The most simple fourth order accurate finite differencing scheme is known as central differencing. Fourth order central differencing uses two points on either side of the point of interest to approximate the function derivative. For this reason, Taylor series for $+2\Delta x$, $+\Delta x$, $-\Delta x$, and for $-2\Delta x$ are all used. FigureFigure 2.5 explains in detail:

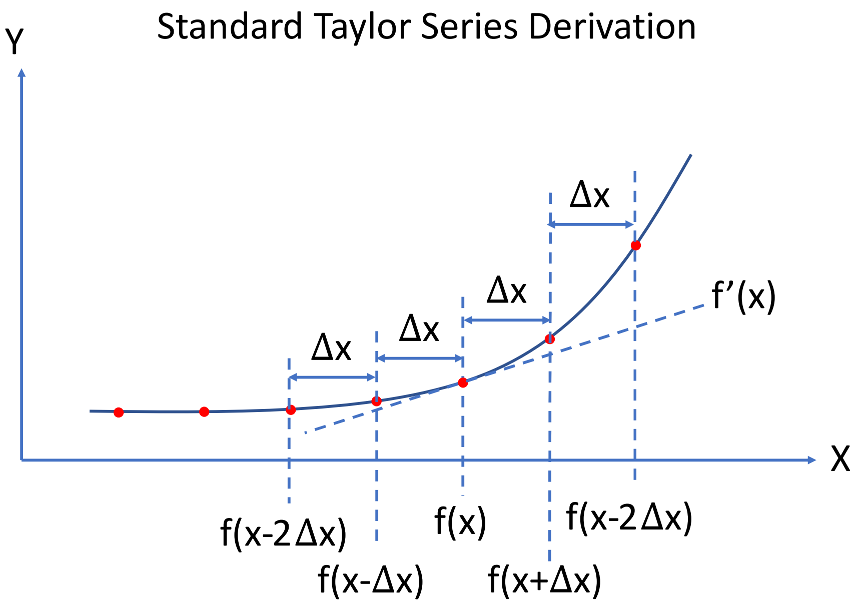


Figure 2.4 General Taylor series centered at x and a spacing of Δx and $2\Delta x$ from the center.

As shown in equation 2.3, the general central differencing equation can be written as follows for a total secant length of $2\Delta x$:

$$f(x-\Delta x) - f(x+\Delta x) = -2\Delta x f'(x) - \frac{(\Delta x)^3}{3} f'''(x)) - \frac{(\Delta x)^5}{60} f^v(x)... \; . \tag{2.16}$$

The general central differencing equation can be written as follows for a total secant length of $4\Delta x$:

$$f(x-2\Delta x) - f(x+2\Delta x) = -2*2\Delta x f'(x) - \frac{(2\Delta x)^3}{3} f'''(x)) - \frac{(2\Delta x)^5}{60} f^v(x)... \tag{2.17}$$

$$f(x-2\Delta x) - f(x+2\Delta x) = -4\Delta x f'(x) - \frac{8(\Delta x)^3}{3} f'''(x)) - \frac{32(\Delta x)^5}{60} f^v(x)... \tag{2.18}$$

Multiplying equation 2.16 by -8 gives:

$$-8f(x-\Delta x) + 8f(x+\Delta x) = +16\Delta x f'(x) + \frac{8(\Delta x)^3}{3} f'''(x)) + \frac{8(\Delta x)^5}{60} f^v(x)... \tag{2.19}$$

and by adding equations 2.18 and 2.19 we can cancel out the $(\Delta x)^3$ term, such that

$$-8f(x-\Delta x) + 8f(x+\Delta x) + f(x-2\Delta x) - f(x+2\Delta x) = +12\Delta x f'(x) - \frac{24(\Delta x)^5}{60} f^v(x)... \tag{2.20}$$

then solving for $f'(x)$ we finally arrive at an equation for the first derivative of f:

$$f'(x) = \frac{-8f(x-\Delta x) + 8f(x+\Delta x) + f(x-2\Delta x) - f(x+2\Delta x)}{12\Delta x} + \frac{24(\Delta x)^4}{60} f^v(x) + \ldots \, . \tag{2.21}$$

Rearranging a bit and recognizing the $\frac{24(\Delta x)^4}{60} f^v(x) + \ldots$ terms as error, the approximate first derivative is:

$$f'(x) \approx \frac{-f(x+2\Delta x) + 8f(x+\Delta x) - 8f(x-\Delta x) + f(x-2\Delta x)}{12\Delta x}. \tag{2.22}$$

Assuming delta x is small, the higher the order of Δx, the smaller the magnitude will be. Since the fourth order of Δx is the lowest order of Δx in the "error terms," the approximation for the first derivative of f in this equation is said to be fourth order accurate.

2.1.4 Fourth Order Accuracy for FDTD

To derive the fourth order derivative approximations for FDTD, the $\frac{\Delta x^2}{2^3 * 3} f''' \left(x - \frac{\Delta x}{2}\right)$ term in equation 2.13 must be eliminated. To do this we need another equation that we can add to or subtract from equation 2.13 in such a way as to perform this elimination. The equation used will in fact be a combination of the general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $+3\Delta x/2$ and a Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $-3\Delta x/2$. Figure 2.5 shows how these points are located.

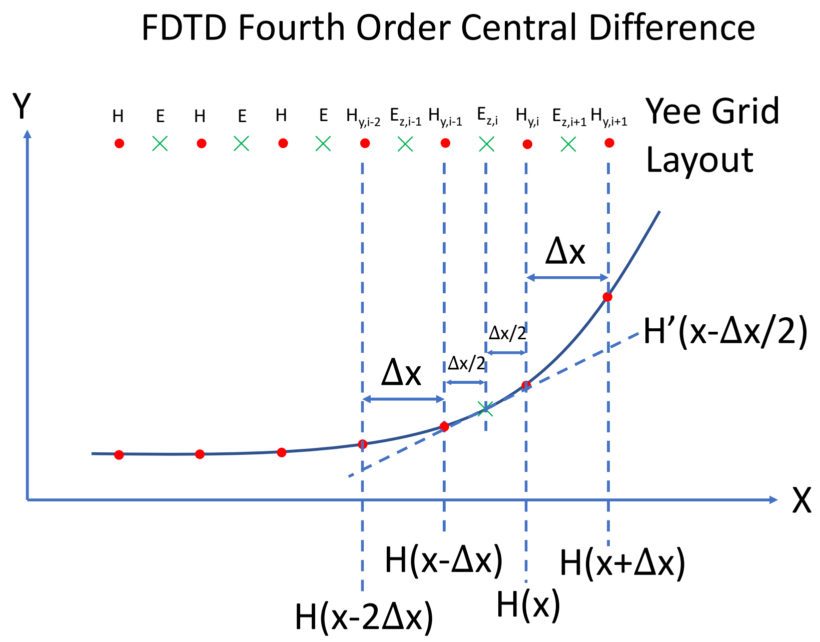

FDTD Fourth Order Central Difference
Y
H E H E H E $H_{y,i-2}$ $E_{z,i-1}$ $H_{y,i-1}$ $E_{z,i}$ $H_{y,i}$ $E_{z,i+1}$ $H_{y,i+1}$
Yee Grid Layout
Δx
Δx/2
H'(x-Δx/2)
Δx
Δx/2
X
H(x-Δx)
H(x+Δx)
H(x)
H(x-2Δx)

Figure 2.5 FDTD compliant Taylor series centered at $x + \Delta x/2$ and a spacing of $\Delta x/2$ and $3\Delta x/2$ from the center.

The general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $+3\Delta x/2$ is as follows:

$$f\left(x-\frac{\Delta x}{2}+\frac{3\Delta x}{2}\right)=f\left(x-\frac{\Delta x}{2}\right)+\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right)$$
$$+\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\ldots \tag{2.23}$$

$$f\left(x+\Delta x\right)=f\left(x-\frac{\Delta x}{2}\right)+\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right)$$
$$+\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\ldots \tag{2.24}$$

The general Taylor series for a point located at $-\Delta x/2$ wth a spacing of $\Delta x/2$ and offset by $-3\Delta x/2$ is as follows:

$$\begin{aligned} f\left(x-\frac{\Delta x}{2}+\frac{-3\Delta x}{2}\right) &= f\left(x-\frac{\Delta x}{2}\right)-\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right) \\ &-\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)-\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+... \end{aligned} \tag{2.25}$$

$$\begin{aligned} f\left(x-2\Delta x\right) &= f\left(x-\frac{\Delta x}{2}\right)-\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right) \\ &-\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)-\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+... \end{aligned} \tag{2.26}$$

Now we can perform the following equation subtraction:

$$\begin{aligned} f\left(x+\Delta x\right)-f\left(x-2\Delta x\right) = &\left[f\left(x-\frac{\Delta x}{2}\right)-f\left(x-\frac{\Delta x}{2}\right)\right]+\left[\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)+\frac{3\Delta x}{2}f'\left(x-\frac{\Delta x}{2}\right)\right] \\ &+\left[\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right)-\frac{\left(\frac{3\Delta x}{2}\right)^2}{2}f''\left(x-\frac{\Delta x}{2}\right)\right] \\ &+\left[\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^3}{6}f'''\left(x-\frac{\Delta x}{2}\right)\right] \\ &+\left[\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)-\frac{\left(\frac{3\Delta x}{2}\right)^4}{24}f^{iv}\left(x-\frac{\Delta x}{2}\right)\right] \\ &+\left[\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^5}{120}f^{v}\left(x-\frac{\Delta x}{2}\right)\right] \end{aligned} \tag{2.27}$$

$$f\left(x+\Delta x\right)-f\left(x-2\Delta x\right)=3\Delta x f'\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^3}{3}f'''\left(x-\frac{\Delta x}{2}\right)+\frac{\left(\frac{3\Delta x}{2}\right)^5}{60}f^{v}\left(x-\frac{\Delta x}{2}\right)+... \tag{2.28}$$

$$\frac{f\left(x+\Delta x\right)-f\left(x-2\Delta x\right)}{\Delta x}=f'\left(x-\frac{\Delta x}{2}\right)+3^3\frac{\Delta x^2}{2^3*3}f'''\left(x-\frac{\Delta x}{2}\right)+3^5\frac{\Delta x^4}{2^5*60}f^{v}\left(x-\frac{\Delta x}{2}\right)+... \tag{2.29}$$

Now performing (equation 2.29) - 3^3 (equation 2.13) we have:

$$\begin{aligned}\frac{f(x+\Delta x)-f(x-2\Delta x)}{\Delta x}-3^3\frac{f(x)-f(x-\Delta x)}{\Delta x} = &\left[3f'\left(x-\frac{\Delta x}{2}\right)-3^3f'\left(x-\frac{\Delta x}{2}\right)\right]\\ &+\left[3^3\frac{\Delta x^2}{2^3*3}f'''\left(x-\frac{\Delta x}{2}\right)-3^3\frac{\Delta x^2}{2^3*3}f'''\left(x-\frac{\Delta x}{2}\right)\right]\\ &+\left[3^5\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)-3^3\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)\right]+...\end{aligned} \tag{2.30}$$

$$\frac{f(x+\Delta x)-f(x-2\Delta x)}{\Delta x}-3^3\frac{f(x)-f(x-\Delta x)}{\Delta x}=-24f'\left(x-\frac{\Delta x}{2}\right)+(3^5-3^3)\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)+... \tag{2.31}$$

$$\frac{f(x+\Delta x)-f(x-2\Delta x)-27f(x)+27f(x-\Delta x)}{\Delta x}=-24f'\left(x-\frac{\Delta x}{2}\right)+(3^5-3^3)\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)+... \tag{2.32}$$

Finally we have:

$$f'\left(x-\frac{\Delta x}{2}\right)=\frac{-f(x+\Delta x)+27f(x)-27f(x-\Delta x)+f(x-2\Delta x)}{24\Delta x}+(3^5-3^3)\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)+... \tag{2.33}$$

Recognizing the $(3^5-3^3)\frac{\Delta x^4}{2^5*60}f^v\left(x-\frac{\Delta x}{2}\right)+...$ terms as error the final fourth order derivative approximation can be written as follows:

$$f'\left(x-\frac{\Delta x}{2}\right)\approx\frac{-f(x+\Delta x)+27f(x)-27f(x-\Delta x)+f(x-2\Delta x)}{24\Delta x} \tag{2.34}$$

Equation 2.34 will be used to calculate the derivative of the magnetic field in order to update the electric field as shown in Figure 2.5. A similar analysis can be performed to calculate the derivative of the electric field in order to update the magnetic field. Equation 2.35 shows the derivative approximation in this case.

$$f'\left(x+\frac{\Delta x}{2}\right)\approx\frac{-f(x+2\Delta x)+27f(x+\Delta x)-27f(x)+f(x-\Delta x)}{24\Delta x} \tag{2.35}$$

2.1.5 Finite Differencing Error Analysis

Figure 2.6 shows the advantage of using fourth order central differencing in terms of error. Figure 2.6 is generated by using four different kinds of finite differencing techniques (forward, backward, second order central, and fourth order central) to approximate the derivative of a given function. The number of points used to discretize the function is varied as the independent variable in Figure 2.6. The approximated derivative for the function is compared to the exact function derivative to find the maximum error value between the exact derivative and a given finite difference approximation for a given number of descretization points. Equation 2.36 is the function who's derivative is approximated and equation 2.37 is the exact derivative of equation 2.36. Code listing Listing 2.1 details how Figure 2.6 is generated in MATLAB.

$$f = e^{-\alpha x^2}\cos(\beta x) \tag{2.36}$$

$$\frac{df}{dx} = -2x\alpha e^{-\alpha x^2}\cos(\beta x) - \beta e^{-\alpha x^2}\sin(\beta x) \tag{2.37}$$

Where, $\alpha = 1.5$ and $\beta = 5$

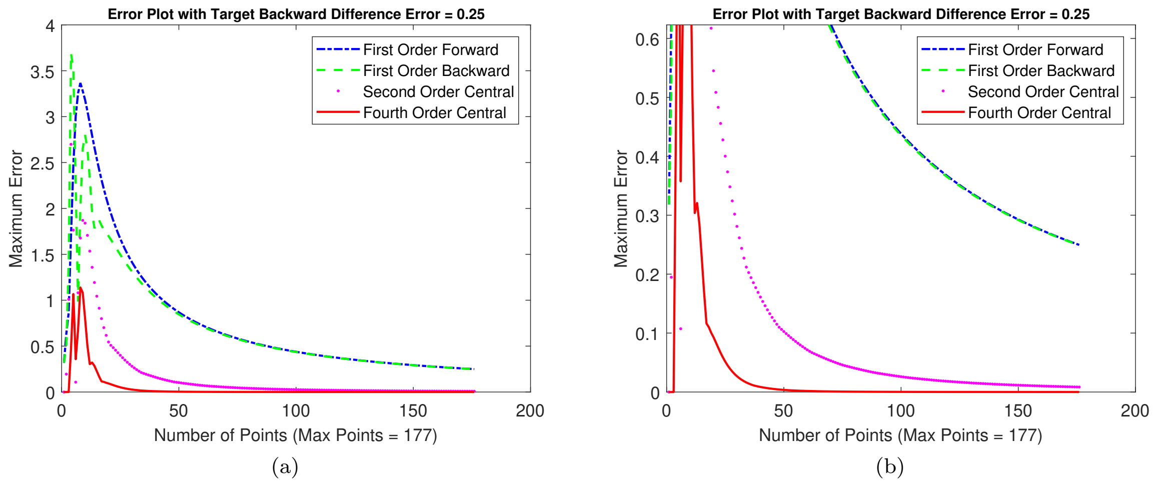


Figure 2.6 Maximum value of error for different finite difference schemes as a function of discretization resolution.

Figure 2.6 clearly shows the fourth order accurate central differencing scheme has the lowest error and therefore is the most accurate of the four techniques plotted. The forward and backward differencing schemes are considered first order accurate because the largest error term in the residual error is proportional to the descretization size ($\propto \Delta x$). As shown in Figure 2.6 the forward and backward differencing schemes have roughly the same and the highest error. More information about forward and backward finite differencing schemes can be found in section 1.2 of [1]. Second order central differencing, sometimes just referred to as central differecing, is considered to be second order accurate because the largest error term is proportional to the descretization size squared ($\propto (\Delta x)^2$). More information about this and second order accurate central differenceing can be found in section 2.1.1. As can be inferred by its name, fourth order central differencing is fourth order accurate meaning the largest error term is proportional to the descretization size raised to the fourth power ($\propto (\Delta x)^4$). More information about the fourth order finite differencing scheme can be found in section 2.1.3. Since Δx is considered to be less than one, relative to the largest usable wavelength of the source waveform, $\Delta x > (\Delta x)^2 > (\Delta x)^4$ and the fourth order accurate central differencing is mathematically the most accurate. Figure 2.6 confirms this mathematical relationship.

Listing 2.1: Finite Difference Errors for Figure 2.6

```
alpha = 1.5;
beta = 5;
maxError = 0.25;

%initialize arrays to hold max error values
forward_error  = [];
backward_error  = [];
central_error  = [];
fourth_central_error  = [];

%main loop to change the number of points of the function
numPoints = 1;
while 1
    numPoints = numPoints + 1;
    %Create x vector and set dx value
    x_exact = linspace(0,pi,numPoints);
    dx = pi/(numPoints-1); %accurate dx calculation because linspace counts "0" as a
         point

    % create exact function and its derivative
    f_exact = exp(-alpha*x_exact.^2).*cos(beta*x_exact);
    f_derivative_exact = -2*x_exact*alpha.*exp(-alpha*x_exact.^2).*cos(beta*x_exact)
        ...
    -exp(-alpha*x_exact.^2).*beta.*sin(beta*x_exact);

%backward differenceing
    %initialize and calculate derivative
    f_derivative_backward  = zeros(1,numPoints);
    f_derivative_backward(2:numPoints) = ...
        (f_exact(2:numPoints)-f_exact(1:numPoints-1))/(dx);

    %calculate error
    f_derivative_error_a  = zeros(1,numPoints);
    f_derivative_error_a(2:numPoints) = f_derivative_backward(2:numPoints)-
        f_derivative_exact(2:numPoints);

    %store error of for current number of points
    backward_error =  [backward_error, max(abs(f_derivative_error_a))];

%forward differenceing
    %initialize and calculate derivative
    f_derivative_forward  = zeros(1,numPoints);
    f_derivative_forward(1:numPoints-1) = ...
        (f_exact(2:numPoints)-f_exact(1:numPoints-1))/(dx);

    %calculate error
    f_derivative_error_c  = zeros(1,numPoints);
    f_derivative_error_c(1:numPoints-1) = f_derivative_forward(1:numPoints-1)-
        f_derivative_exact(1:numPoints-1);

    %store error of for current number of points
    forward_error = [forward_error, max(abs(f_derivative_error_c))];

%2nd order central differencing
    %initialize and calculate derivative
    f_derivative_central_2  = zeros(1,numPoints);
    f_derivative_central_2(2:numPoints-1) = ...
        (f_exact(3:numPoints)-f_exact(1:numPoints-2))/(2*dx);


    %calculate error
    f_derivative_error_d  = zeros(1,numPoints);
    f_derivative_error_d(2:numPoints-1) = f_derivative_central_2(2:numPoints-1)-
        f_derivative_exact(2:numPoints-1);

    %store error of for current number of points
    central_error = [central_error, max(abs(f_derivative_error_d))];

%4th order central differencing
    %initialize and calculate derivative
    f_derivative_central_4  = zeros(1,numPoints);
    f_derivative_central_4(3:numPoints-2) = ...
        (-f_exact(5:numPoints)+8*f_exact(4:numPoints-1)-8*f_exact(2:numPoints-3)+
            f_exact(1:numPoints-4))/(12*dx);

    %calculate error
    f_derivative_error_b  = zeros(1,numPoints);
    f_derivative_error_b(3:numPoints-2) = f_derivative_central_4(3:numPoints-2)-
        f_derivative_exact(3:numPoints-2);

    %store error of for current number of points
    fourth_central_error = [fourth_central_error, max(abs(f_derivative_error_b))];

    if max(abs(f_derivative_error_a)) < maxError
        break;
    end
end

pointSet = 1:numPoints-1;

figure(11); %6
plot(pointSet,forward_error,'b-.',pointSet,backward_error,'g--',pointSet,
    central_error,'m.',pointSet,fourth_central_error,'r-','linewidth',1.5);
set (gca,'FontSize',12,'fontweight','demi');
xlabel(['Number of Points (Max Points = ', num2str(numPoints),')'],'fontsize',12)
ylabel('Maximum Error','FontSize',12);
title(['Error Plot with Target Backward Difference Error = ', num2str(maxError)],'
    fontsize',10)
legend('First Order Forward','First Order Backward','Second Order Central','Fourth
    Order Central')
```

2.2 3D FDTD Formulation

The goal of this section is to present the three dimensional fourth order FDTD formulation. The discrete material updating equations are derived from Maxwell's equations in such a way as to allow any discrete derivative approximation to be used. By deriving the updating equations in this way, the fourth order and second order accurate updating equations are very similar. The coefficients are the same, and the only change is the derivative approximation itself. These material updating equations are then extended to derive the updating equations for sources and lumped circuit elements. These derivations are also formatted to allow any discrete derivative approximation to be used with the same given coefficients. Derivative approximations of any order accuracy are mathematically possible [20]. Modifications would need to be made to the results in [20] to apply them to the Yee staggard grid FDTD approach, but the thoery is good. Other research has been done regarding sixth order FDTD [21] and [22]. An n_{th} order derivative approximation is also prsented in [21] which would be compliant with the updating equation coefficients presented in this thesis.

2.2.1 Material Updating Equations

To find the updating equation for E_x, we must start at the component level Ampere-Maxwell equation for E_x (also see equation 1.11):

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right). \tag{2.38}$$

The goal here is to use central differencing to approximate each partial derivative. We want to use central differencing because it has high accuracy for little computational load. However, central differencing is a little tricky to see in this case. In order for the magnetic field and electric field updating equations to work together to solve a problem using FDTD, the time derivatives for updating the electric fields look a lot like a forward or backward differencing. However, the derivative is evaluated at $(n+\frac{1}{2})\Delta t$, making it truly a central difference. Equation 2.39 illustrates further:

$$\left.\frac{\partial E_x^{n+\frac{1}{2}}}{\partial t}\right|_{(i,j,k)} = \frac{E_x^{n+1}(i,j,k) - E_x^{n}(i,j,k)}{\Delta t}. \tag{2.39}$$

Since the time derivative in E_x is evaluated at $(n+\frac{1}{2})\Delta t$, all other spacial derivatives in the equation must also be evaluated at $(n+\frac{1}{2})\Delta t$. Another key point is that these fourth order accurate central difference approximations for the spacial differential equations must be evaluated at the same location in space as E_x. Using the Yee cell to get the correct locations, the following derivatives are approximated (see section 2.1.2 and 2.1.4). For this derivation the spacial derivatives are represented as α and β for clarity.

$$\alpha = \left.\frac{\partial H_z^{n+\frac{1}{2}}}{\partial y}\right|_{(i,j,k)}, \tag{2.40}$$

$$\beta = \left.\frac{\partial H_y^{n+\frac{1}{2}}}{\partial z}\right|_{(i,j,k)}. \tag{2.41}$$

And more simply, E_x and J_x can be written as:

$$J_x^{n+\frac{1}{2}}\Big|_{(i,j,k)} = J_x^{n+\frac{1}{2}}(i,j,k) \tag{2.42}$$

$$E_x^{n+\frac{1}{2}}\Big|_{(i,j,k)} = E_x^{n+\frac{1}{2}}(i,j,k) \tag{2.43}$$

However, since we are updating E_x in this equation, E_x must only be evaluated at integer values, so we make the approximation:

$$E_x^{n+\frac{1}{2}}\Big|_{(i,j,k)} \approx \frac{E_x^{n+1}(i,j,k) + E_x^{n}(i,j,k)}{2}. \tag{2.44}$$

Putting equations 2.39 2.40, 2.41, 2.42, and 2.44 into equation 2.38 we have:

$$\begin{aligned}\frac{E_x^{n+1}(i,j,k) - E_x^{n}(i,j,k)}{\Delta t} &= \frac{1}{\epsilon_x(i,j,k)}\alpha - \frac{1}{\epsilon_x(i,j,k)}\beta \\ &-\frac{\sigma_x^e(i,j,k)}{\epsilon_x(i,j,k)}\left[\frac{E_x^{n+1}(i,j,k) + E_x^{n}(i,j,k)}{2}\right] \\ &-\frac{1}{\epsilon_x(i,j,k)}J_x^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.45}$$

$$\begin{aligned}\frac{E_x^{n+1}(i,j,k)}{\Delta t} + \frac{\sigma_x^e(i,j,k)}{\epsilon_x(i,j,k)}\frac{E_x^{n+1}(i,j,k)}{2} &= \frac{1}{\epsilon_x(i,j,k)}\alpha - \frac{1}{\epsilon_x(i,j,k)}\beta \\ &+\frac{E_x^{n}(i,j,k)}{\Delta t} - \frac{\sigma_x^e(i,j,k)}{\epsilon_x(i,j,k)}\frac{E_x^{n}(i,j,k)}{2} \\ &-\frac{1}{\epsilon_x(i,j,k)}J_x^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.46}$$

$$\begin{aligned}\frac{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}{2\Delta t\epsilon_x(i,j,k)}E_x^{n+1}(i,j,k) &= \frac{1}{\epsilon_x(i,j,k)}\alpha - \frac{1}{\epsilon_x(i,j,k)}\beta \\ &+\frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k)}{2\Delta t\epsilon_x(i,j,k)}E_x^{n}(i,j,k) \\ &-\frac{1}{\epsilon_x(i,j,k)}J_x^{n+\frac{1}{2}}(i,j,k).\end{aligned} \tag{2.47}$$

The final updating equation for E_x is as follows:

$$\begin{aligned}E_x^{n+1}(i,j,k) &= \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}\alpha \\ &-\frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}\beta \\ &+\frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}E_x^{n}(i,j,k) \\ &-\frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}J_x^{n+\frac{1}{2}}(i,j,k).\end{aligned} \tag{2.48}$$

Equation 2.48 and the proceeding alegbra show that α and β carry through the updating equation algebra. Since α and β generally represent the spacial derivative approximations and are separable from the coefficients in equation 2.48, α and β can represent derivative approximations of any order accuracy. They can also represent any differencing scheme, meaning they could be central, forward, or backward differencing. In the case of this research, the second order updating equations are compared to the fourth order updating equations. Equation 2.49 shows the final second order updating equation and 2.50 shows the fourth order updating equation by substituting α and β for their appropriate derivative approximations.

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{(2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k))}\frac{H_z^{n+\frac{1}{2}}(i,j,k)-H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\ & -\frac{2\Delta t}{(2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k))}\frac{H_y^{n+\frac{1}{2}}(i,j,k)-H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\ & +\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}E_x^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}J_y^{n+\frac{1}{2}}(i,j,k). \end{aligned} \tag{2.49}$$

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\frac{-H_z^{n+\frac{1}{2}}(i,j+1,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i,j-1,k)+H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\frac{-H_y^{n+\frac{1}{2}}(i,j,k+1)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i,j,k-1)+H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\ & +\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}E_x^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}J_x^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.50}$$

The second order updating equations for E_y, E_z, H_x, H_y, and H_z can be derived in a similar way and are presented in [1]. The fourth order updating equations for E_y, E_z, H_x, H_y, and H_z are derived in a similar mannor and shown in equations 2.51 through 2.55

$$\begin{aligned} E_y^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)}\frac{-H_x^{n+\frac{1}{2}}(i,j,k+1)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j,k-1)+H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\ & -\frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)}\frac{-H_z^{n+\frac{1}{2}}(i+1,j,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i-1,j,k)+H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\ & +\frac{2\epsilon_y(i,j,k)-\Delta t\sigma_y^e(i,j,k)}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)}E_y^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)}J_y^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.51}$$

$$\begin{aligned}E_z^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)}\frac{-H_y^{n+\frac{1}{2}}(i+1,j,k)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i-1,j,k)+H_y^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\ & -\frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)}\frac{-H_x^{n+\frac{1}{2}}(i,j+1,k)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j-1,k)+H_x^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\ & +\frac{2\epsilon_z(i,j,k)-\Delta t\sigma_z^e(i,j,k)}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)}E_z^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)}J_z^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.52}$$

$$\begin{aligned}H_x^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\mu_x(i,j,k)+\Delta t\sigma_x^m(i,j,k)}\frac{-E_y^{n+\frac{1}{2}}(i,j,k+2)+27E_y^{n+\frac{1}{2}}(i,j,k+1)-27E_y^{n+\frac{1}{2}}(i,j,k)+E_y^{n+\frac{1}{2}}(i,j,k-1)}{24\Delta z} \\ & -\frac{2\Delta t}{2\mu_x(i,j,k)+\Delta t\sigma_x^m(i,j,k)}\frac{-E_z^{n+\frac{1}{2}}(i,j+2,k)+27E_z^{n+\frac{1}{2}}(i,j+1,k)-27E_z^{n+\frac{1}{2}}(i,j,k)+E_z^{n+\frac{1}{2}}(i,j-1,k)}{24\Delta y} \\ & +\frac{2\mu_x(i,j,k)-\Delta t\sigma_x^m(i,j,k)}{2\mu_x(i,j,k)+\Delta t\sigma_x^m(i,j,k)}H_x^n(i,j,k) \\ & -\frac{2\Delta t}{2\mu_x(i,j,k)+\Delta t\sigma_x^m(i,j,k)}M_x^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.53}$$

$$\begin{aligned}H_y^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}\frac{-E_z^{n+\frac{1}{2}}(i+2,j,k)+27E_z^{n+\frac{1}{2}}(i+1,j,k)-27E_z^{n+\frac{1}{2}}(i,j,k)+E_z^{n+\frac{1}{2}}(i-1,j,k)}{24\Delta x} \\ & -\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}\frac{-E_x^{n+\frac{1}{2}}(i,j,k+2)+27E_x^{n+\frac{1}{2}}(i,j,k+1)-27E_x^{n+\frac{1}{2}}(i,j,k)+E_x^{n+\frac{1}{2}}(i,j,k-1)}{24\Delta z} \\ & +\frac{2\mu_y(i,j,k)-\Delta t\sigma_y^m(i,j,k)}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}H_y^n(i,j,k) \\ & -\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}M_y^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.54}$$

$$\begin{aligned}H_z^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\mu_z(i,j,k)+\Delta t\sigma_z^m(i,j,k)}\frac{-E_x^{n+\frac{1}{2}}(i,j+2,k)+27E_x^{n+\frac{1}{2}}(i,j+1,k)-27E_x^{n+\frac{1}{2}}(i,j,k)+E_x^{n+\frac{1}{2}}(i,j-1,k)}{24\Delta y} \\ & -\frac{2\Delta t}{2\mu_z(i,j,k)+\Delta t\sigma_z^m(i,j,k)}\frac{-E_y^{n+\frac{1}{2}}(i+2,j,k)+27E_y^{n+\frac{1}{2}}(i+1,j,k)-27E_y^{n+\frac{1}{2}}(i,j,k)+E_y^{n+\frac{1}{2}}(i-1,j,k)}{24\Delta x} \\ & +\frac{2\mu_z(i,j,k)-\Delta t\sigma_z^m(i,j,k)}{2\mu_z(i,j,k)+\Delta t\sigma_z^m(i,j,k)}H_z^n(i,j,k) \\ & -\frac{2\Delta t}{2\mu_z(i,j,k)+\Delta t\sigma_z^m(i,j,k)}M_z^{n+\frac{1}{2}}(i,j,k)\end{aligned} \tag{2.55}$$

2.2.2 Updating Equations for Sources and Lumped Circuit Elements

The starting place for deriving the updating equations for sources and lumped circuit elements is also the Ampere Maxwell Equation:

$$\bigtriangledown \times \overrightarrow{H} = \sigma^e \overrightarrow{E} + \epsilon \frac{\partial \overrightarrow{E}}{\partial t} + \overrightarrow{J}_l, \tag{2.56}$$

where the impressed current $\overrightarrow{J}_i$ is represented as the lumped current $\overrightarrow{J}_l$.

Since J represents the current density per unit area, and I represents the total current, if we assume the current density is uniform over a single cell we can relate I and J in the following ways:

$$J_{lx} = \frac{I_{lx}}{\Delta z \Delta y} \tag{2.57}$$

$$J_{ly} = \frac{I_{ly}}{\Delta z \Delta x} \tag{2.58}$$

$$J_{lz} = \frac{I_{lz}}{\Delta x \Delta y}. \tag{2.59}$$

We can deal with the the E_x component Maxwell Ampere equation as follows:

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right), \tag{2.60}$$

after some algebraic manipulation, the final updating equation for E_x is shown in equation 2.50. However as has been shown, the most general updating equation can be written in terms of α and β where α and β represent the numeric approximations for the derivatives:

Substituting equation 2.57 into the most general form of an updating equation (equation 2.48) we have:

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\alpha \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\beta \\ & + \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \frac{I_{lx}^{n+\frac{1}{2}}(i,j,k)}{\Delta z \Delta y}. \end{aligned} \tag{2.61}$$

Even though equation 2.61 is rather complicated, the only "unknown" is I_{lx}. Since the rest of the equation is in terms of E and H, a way to write I_{lx} in terms of E and H must be found. To do this, the I-V characteristics and voltage-field relationships of the voltage source, or lumped element, are used.

2.2.2.1 X Oriented Resistive Voltage Source

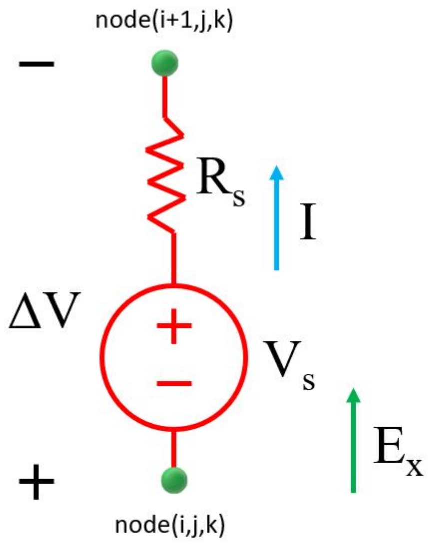


Figure 2.7 A circuit schematic of a resistive voltage source.

The I-V characteristic for the resistive voltage source shown in Figure 2.7 is derived as follows:

From the passive sign convention we have:

$$\Delta V = -V_s + IR_s, \tag{2.62}$$

solving for I yields the I-V characteristic for a resistive voltage source:

$$I = \frac{\Delta V + V_s}{R_s}. \tag{2.63}$$

From equation 2.63 we must convert all voltages to electric field components that can be plugged into the updating equation. By assuming the electric field is constant between two nodes in the simulation, voltage is simply the electric field times distance. To evaluate the voltage at time step $n+\frac{1}{2}$, a time average of electric field will be used to calculate the voltage. Assuming the shown voltage polarity in the X-direction, equation 2.63 becomes:

$$I_{lx}^{n+\frac{1}{2}} = \frac{\Delta V_x^{n+\frac{1}{2}} + V_s^{n+\frac{1}{2}}}{R_s} \tag{2.64}$$

$$I_{lx}^{n+\frac{1}{2}} = \frac{\Delta x\left(\frac{E_x^{n+1}(i,j,k)+E_x^{n}(i,j,k)}{2}\right) + V_s^{n+\frac{1}{2}}}{R_s}. \tag{2.65}$$

Equation 2.65 is now ready to be substituted into equation 2.61:

$$\begin{aligned} E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\alpha \\ &-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\beta \\ &+\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}E_x^n(i,j,k) \\ &-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\left(\frac{\Delta x\left(\frac{E_x^{n+1}(i,j,k)+E_x^n(i,j,k)}{2}\right)+V_s^{n+\frac{1}{2}}}{R_s\Delta z\Delta y}\right). \end{aligned} \tag{2.66}$$

Next, $E_x^{n+1}(i,j,k)$ must be solved for explicitly. Letting $(2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k))=\gamma$ and $(2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k))=\eta$ and breaking the fraction:

$$\begin{aligned} E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{\gamma}\alpha-\frac{2\Delta t}{\gamma}\beta+\frac{\eta}{\gamma}E_x^n(i,j,k) \\ &-\frac{\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}E_x^{n+1}(i,j,k)-\frac{\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}E_x^n(i,j,k)-\frac{2\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}V_s^{n+\frac{1}{2}} \end{aligned} \tag{2.67}$$

$$\begin{aligned} \left(1+\frac{\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}\right)E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{\gamma}\alpha-\frac{2\Delta t}{\gamma}\beta \\ &+\left(\frac{\eta}{\gamma}-\frac{\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}\right)E_x^n(i,j,k)-\frac{2\Delta t\Delta x}{\gamma R_s\Delta z\Delta y}V_s^{n+\frac{1}{2}} \end{aligned} \tag{2.68}$$

$$\begin{aligned} E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{\gamma+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}\alpha-\frac{2\Delta t}{\gamma+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}\beta \\ &+\frac{\eta-\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{\gamma+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}E_x^n(i,j,k)-\frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{\gamma+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}V_s^{n+\frac{1}{2}}. \end{aligned} \tag{2.69}$$

Substituting back in for η and γ, the most general updating equation for a resistive voltage source oriented in the X-direction is derived:

$$\begin{aligned} E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}\alpha \\ &-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}\beta \\ &+\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)-\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}E_x^n(i,j,k) \\ &-\frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)+\frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}V_s^{n+\frac{1}{2}}. \end{aligned} \tag{2.70}$$

Finally, substituting second order approximate derivatives back in for α and β, the second order updating equation for a resistive voltage source oriented in the X-direction is derived:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \frac{H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\
& - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\
& + \frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_x^n(i,j,k) \\
& - \frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} V_s^{n+\frac{1}{2}}.
\end{aligned}
\tag{2.71}
$$

Substituting fourth order approximate derivatives back in for α and β, the fourth order updating equation for a resistive voltage source oriented in the X-direction is derived:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i,j+1,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i,j-1,k) + H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_y^{n+\frac{1}{2}}(i,j,k+1) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i,j,k-1) + H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& + \frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_x^n(i,j,k) \\
& - \frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} V_s^{n+\frac{1}{2}}.
\end{aligned}
\tag{2.72}
$$

The second order updating equations for E_y and E_z can be derived in a similar way and are presented in [1]. The fourth order updating equations for E_y and E_z are derived in a similar mannor and shown in equations 2.73 and 2.74. The magnetic field updating equations are unaltered and shown in equations 2.53

through 2.55.

$$
\begin{aligned}
E_y^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j,k+1) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j,k-1) + H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& - \frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i+1,j,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i-1,j,k) + H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& + \frac{2\epsilon_y(i,j,k) - \Delta t\sigma_y^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_y^n(i,j,k) \\
& - \frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} V_s^{n+\frac{1}{2}}.
\end{aligned}
\tag{2.73}
$$

$$
\begin{aligned}
E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_y^{n+\frac{1}{2}}(i+1,j,k) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i-1,j,k) + H_y^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& - \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j+1,k) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j-1,k) + H_x^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& + \frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_z^n(i,j,k) \\
& - \frac{2\Delta t\Delta x/R_s\Delta z\Delta y}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} V_s^{n+\frac{1}{2}}.
\end{aligned}
\tag{2.74}
$$

For a resistive voltage source oriented in the Y-direction, the $\frac{\Delta x}{\Delta z\Delta y}$ terms in equations 2.72 through 2.74 is replaced by $\frac{\Delta y}{\Delta x\Delta z}$.

For a resistive voltage source oriented in the Z-direction, the $\frac{\Delta x}{\Delta z\Delta y}$ terms in equations 2.72 through 2.74 is replaced by $\frac{\Delta z}{\Delta x\Delta y}$.

2.2.2.2 X Oriented Resistor

The updating equations for a resistor are the same as equations 2.71 and 2.72 except that $V_s^{n+\frac{1}{2}}$ is set to 0 for all time. Equations 2.75 through 2.77 summarize:

$$\begin{aligned}
E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_z^{n+\frac{1}{2}}(i,j+1,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i,j-1,k) + H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& -\frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_y^{n+\frac{1}{2}}(i,j,k+1) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i,j,k-1) + H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& +\frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_x^n(i,j,k).
\end{aligned} \tag{2.75}$$

$$\begin{aligned}
E_y^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_x^{n+\frac{1}{2}}(i,j,k+1) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j,k-1) + H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& -\frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_z^{n+\frac{1}{2}}(i+1,j,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i-1,j,k) + H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& +\frac{2\epsilon_y(i,j,k) - \Delta t\sigma_y^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_y^n(i,j,k).
\end{aligned} \tag{2.76}$$

$$\begin{aligned}
E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_y^{n+\frac{1}{2}}(i+1,j,k) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i-1,j,k) + H_y^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& -\frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} \\
& *\frac{-H_x^{n+\frac{1}{2}}(i,j+1,k) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j-1,k) + H_x^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& +\frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k) - \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{\Delta t\Delta x}{R_s\Delta z\Delta y}} E_z^n(i,j,k).
\end{aligned} \tag{2.77}$$

For a resistor oriented in the Y-direction, the $\frac{\Delta x}{\Delta z\Delta y}$ terms in equations 2.75 through 2.77 is replaced by $\frac{\Delta y}{\Delta x\Delta z}$.

For a resistor oriented in the Z-direction, the $\frac{\Delta x}{\Delta z \Delta y}$ terms in equations 2.75 through 2.77 is replaced by $\frac{\Delta z}{\Delta x \Delta y}$.

2.2.2.3 X Oriented Capacitor

Figure 2.8 shows the schematic of the capacitor analyzed in this section.

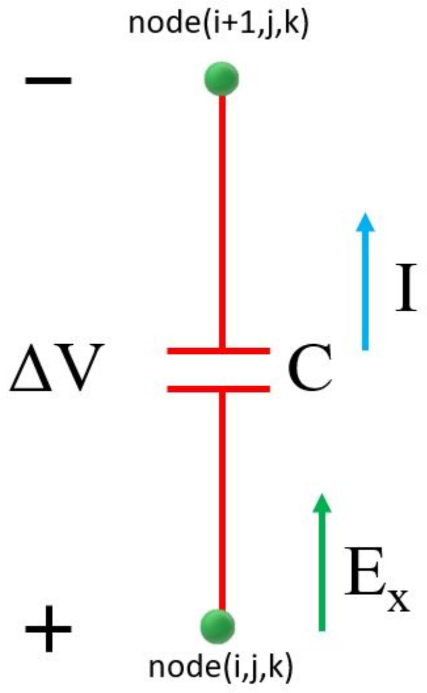


Figure 2.8 A circuit schematic of a capacitor.

The I-V characteristic for a capacitor is derived as follows from:

$$I = C\frac{dV}{dt}. \tag{2.78}$$

The discrete form of this equation is approximated using second order accurate central differencing and can be written as:

$$I_{lx}^{n+\frac{1}{2}} = C\frac{\Delta V^{n+1} - \Delta V^{n}}{\Delta t}, \tag{2.79}$$

from equation 2.79, all voltages must be converted to electric field components that can be plugged into the updating equation. By assuming the electric field is constant between two nodes in the simulation, voltage is simply the electric field times distance. To evaluate the voltage at time step $n + \frac{1}{2}$, a time average of electric field will be used to calculate the voltage. Assuming the shown voltage polarity in the

X-direction, equation 2.79 becomes:

$$I_{lx}^{n+\frac{1}{2}} = C\Delta x \frac{E_x^{n+1} - E_x^n}{\Delta t} \tag{2.80}$$

Equation 2.80 is now ready to be substituted into equation 2.61:

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\alpha \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\beta \\ & + \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \left(\frac{C\Delta x \frac{E_x^{n+1}(i,j,k) - E_x^n(i,j,k)}{\Delta t}}{\Delta z \Delta y} \right) \end{aligned} \tag{2.81}$$

Next, $E_x^{n+1}(i,j,k)$ must be solved for explicitly. Letting $(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)) = \gamma$ and $(2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)) = \eta$ and breaking the fraction we have:

$$E_x^{n+1}(i,j,k) = \frac{2\Delta t}{\gamma}\alpha - \frac{2\Delta t}{\gamma}\beta + \frac{\eta}{\gamma}E_x^n(i,j,k) - \frac{2\Delta t}{\gamma}\left(\frac{C\Delta x E_x^{n+1}(i,j,k)}{\Delta t \Delta z \Delta y} - \frac{C\Delta x E_x^n(i,j,k)}{\Delta t \Delta z \Delta y} \right) \tag{2.82}$$

$$E_x^{n+1}(i,j,k)\left(1 + \frac{2}{\gamma}\frac{C\Delta x}{\Delta z \Delta y}\right) = \frac{2\Delta t}{\gamma}\alpha - \frac{2\Delta t}{\gamma}\beta + E_x^n(i,j,k)\left(\frac{\eta}{\gamma} + \frac{2}{\gamma}\frac{C\Delta x}{\Delta z \Delta y}\right) \tag{2.83}$$

$$E_x^{n+1}(i,j,k)\left(\frac{\gamma + \frac{2C\Delta x}{\Delta z \Delta y}}{\gamma}\right) = \frac{2\Delta t}{\gamma}\alpha - \frac{2\Delta t}{\gamma}\beta + E_x^n(i,j,k)\left(\frac{\eta + \frac{2C\Delta x}{\Delta z \Delta y}}{\gamma}\right) \tag{2.84}$$

$$E_x^{n+1}(i,j,k) = \frac{2\Delta t}{\gamma + \frac{2C\Delta x}{\Delta z \Delta y}}\alpha - \frac{2\Delta t}{\gamma + \frac{2C\Delta x}{\Delta z \Delta y}}\beta + \frac{\eta + \frac{2C\Delta x}{\Delta z \Delta y}}{\gamma + \frac{2C\Delta x}{\Delta z \Delta y}}E_x^n(i,j,k). \tag{2.85}$$

Substituting back in for η and γ, the most general form of an updating equation for a capacitor oriented in the X-direction is derived:

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z \Delta y}}\alpha \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z \Delta y}}\beta \\ & + \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z \Delta y}}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z \Delta y}} E_x^n(i,j,k). \end{aligned} \tag{2.86}$$

Finally, substituting second order approximate derivatives back in for α and β, the second order updating equation for a capacitor oriented in the X-direction is derived:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \frac{H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\
& - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\
& + \frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} E_x^n(i,j,k).
\end{aligned}
\tag{2.87}
$$

Substituting fourth order approximate derivatives back in for α and β, the fourth order updating equation for a capacitor oriented in the X-direction is derived:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i,j+1,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i,j-1,k) + H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_y^{n+\frac{1}{2}}(i,j,k+1) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i,j,k-1) + H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& + \frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} E_x^n(i,j,k).
\end{aligned}
\tag{2.88}
$$

The second order updating equations for E_y and E_z can be derived in a similar way and are presented in [1]. The fourth order updating equations for E_y and E_z are derived in a similar mannor and shown in equations 2.89 and 2.90. The magnetic field updating equations are unaltered and shown in equations 2.53 through 2.55.

$$
\begin{aligned}
E_y^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j,k+1) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j,k-1) + H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& - \frac{2\Delta t}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i+1,j,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i-1,j,k) + H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& + \frac{2\epsilon_y(i,j,k) - \Delta t\sigma_y^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}}{2\epsilon_y(i,j,k) + \Delta t\sigma_y^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} E_y^n(i,j,k).
\end{aligned}
\tag{2.89}
$$

$$
\begin{aligned}
E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_y^{n+\frac{1}{2}}(i+1,j,k) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i-1,j,k) + H_y^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& - \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j+1,k) + 27H_x^{n+\frac{1}{2}}(i,j,k) - 27H_x^{n+\frac{1}{2}}(i,j-1,k) + H_x^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
& + \frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \frac{2C\Delta x}{\Delta z\Delta y}} E_z^n(i,j,k).
\end{aligned}
\tag{2.90}
$$

For a capacitor oriented in the Y-direction, the $\frac{\Delta x}{\Delta z\Delta y}$ terms in equations 2.88 through 2.90 is replaced by $\frac{\Delta y}{\Delta x\Delta z}$.

For a capacitor oriented in the Z-direction, the $\frac{\Delta x}{\Delta z\Delta y}$ terms in equations 2.88 through 2.90 is replaced by $\frac{\Delta z}{\Delta x\Delta y}$.

2.2.2.4 X Oriented Inductor

Figure 2.9 shows the schematic of the inductor analyzed in this section.

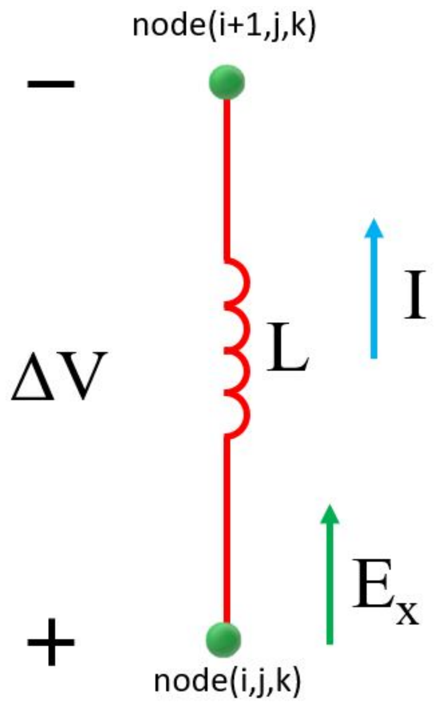


Figure 2.9 A circuit schematic of an inductor.

The I-V characteristic for an inductor is derived as follows from:

$$\Delta V = L\frac{dI}{dt}. \tag{2.91}$$

The discrete form of this equation is approximated using second order accurate central differencing and can be written as:

$$\Delta V^n = L\frac{I_{lx}^{n+\frac{1}{2}} - I_{lx}^{n-\frac{1}{2}}}{\Delta t}, \tag{2.92}$$

$$I_{lx}^{n+\frac{1}{2}} = \frac{\Delta t \Delta V^n}{L} + I_{lx}^{n-\frac{1}{2}}. \tag{2.93}$$

From equation 2.93 we must convert all voltages to electric field components that can be plugged into the updating equation. By assuming the electric field is constant between two nodes in the simulation, voltage is simply the electric field times distance. To evaluate the voltage at time step $n+\frac{1}{2}$, a time average of electric field will be used to calculate the voltage. Assuming the shown voltage polarity in the X-direction, equation 2.93 becomes:

$$I_{lx}^{n+\frac{1}{2}} = \frac{\Delta t \Delta x E_x^n}{L} + I_{lx}^{n-\frac{1}{2}}. \tag{2.94}$$

Equation 2.94 is now ready to be substituted into equation 2.61:

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\alpha \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\beta \\ & + \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \left(\frac{\frac{\Delta t \Delta x E_x^n(i,j,k)}{L} + I_{lx}^{n-\frac{1}{2}}(i,j,k)}{\Delta z \Delta y} \right). \end{aligned} \tag{2.95}$$

Simplifying equation 2.95 to be in terms of the current density, we have:

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\alpha \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\beta \\ & + \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \left(\frac{\Delta t \Delta x E_x^n(i,j,k)}{L \Delta z \Delta y} + J_{lx}^{n-\frac{1}{2}}(i,j,k) \right). \end{aligned} \tag{2.96}$$

Since $E_x^n(i,j,k)$ and $J_{lx}^{n-\frac{1}{2}}(i,j,k)$ are known quantities from the previous time step, equation 2.96 represents the most general updating equation for an inductor.

Finally, substituting second order approximate derivatives back in for α and β we arrive at the second order updating equation for an inductor oriented in the X-direction:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \frac{H_z^{n+\frac{1}{2}}(i,j,k)-H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\
&-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \frac{H_y^{n+\frac{1}{2}}(i,j,k)-H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\
&+\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} E_x^n(i,j,k) \\
&-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \left(\frac{\Delta t\Delta x E_x^n(i,j,k)}{L\Delta z\Delta y}+J_{lx}^{n-\frac{1}{2}}(i,j,k)\right).
\end{aligned} \tag{2.97}
$$

Substituting fourth order approximate derivatives back in for α and β we arrive at the fourth order updating equation for an inductor oriented in the X-direction:

$$
\begin{aligned}
E_x^{n+1}(i,j,k) = &\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \\
&*\frac{-H_z^{n+\frac{1}{2}}(i,j+1,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i,j-1,k)+H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\
&-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \\
&*\frac{-H_y^{n+\frac{1}{2}}(i,j,k+1)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i,j,k-1)+H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
&+\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} E_x^n(i,j,k) \\
&-\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \left(\frac{\Delta t\Delta x E_x^n(i,j,k)}{L\Delta z\Delta y}+J_{lx}^{n-\frac{1}{2}}(i,j,k)\right).
\end{aligned} \tag{2.98}
$$

Note that the coefficients in equations 2.97 and 2.98 are the same as the coefficients for the general updating equations for materials presented in section 2.2.1.

The second order updating equations for E_y and E_z can be derived in a similar way and are presented in [1]. The fourth order updating equations for E_y and E_z are derived in a similar mannor and shown in equations 2.99 and 2.100. The magnetic field updating equations are unaltered and shown in equations

2.53 through 2.55.

$$
\begin{aligned}
E_y^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j,k+1)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j,k-1)+H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& - \frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i+1,j,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i-1,j,k)+H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& + \frac{2\epsilon_y(i,j,k)-\Delta t\sigma_y^e(i,j,k)}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)} E_y^n(i,j,k) \\
& - \frac{2\Delta t}{2\epsilon_y(i,j,k)+\Delta t\sigma_y^e(i,j,k)} \left(\frac{\Delta t \Delta x E_y^n(i,j,k)}{L\Delta z\Delta y} + J_{ly}^{n-\frac{1}{2}}(i,j,k) \right).
\end{aligned}
\tag{2.99}
$$

$$
\begin{aligned}
E_z^{n+1}(i,j,k) = {} & \frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)} \\
& * \frac{-H_x^{n+\frac{1}{2}}(i,j,k+1)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j,k-1)+H_x^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\
& - \frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)} \\
& * \frac{-H_z^{n+\frac{1}{2}}(i+1,j,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i-1,j,k)+H_z^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\
& + \frac{2\epsilon_z(i,j,k)-\Delta t\sigma_z^e(i,j,k)}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)} E_z^n(i,j,k) \\
& - \frac{2\Delta t}{2\epsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)} \left(\frac{\Delta t \Delta x E_z^n(i,j,k)}{L\Delta z\Delta y} + J_{lz}^{n-\frac{1}{2}}(i,j,k) \right).
\end{aligned}
\tag{2.100}
$$

For a capacitor oriented in the Y-direction, the $\frac{\Delta x}{\Delta z \Delta y}$ terms in equations 2.98 through 2.100 is replaced by $\frac{\Delta y}{\Delta x \Delta z}$.

For a capacitor oriented in the Z-direction, the $\frac{\Delta x}{\Delta z \Delta y}$ terms in equations 2.98 through 2.100 is replaced by $\frac{\Delta z}{\Delta x \Delta y}$.

2.2.2.5 X Oriented Diode

Figure 2.10 shows the schematic of the diode analyzed in this section.

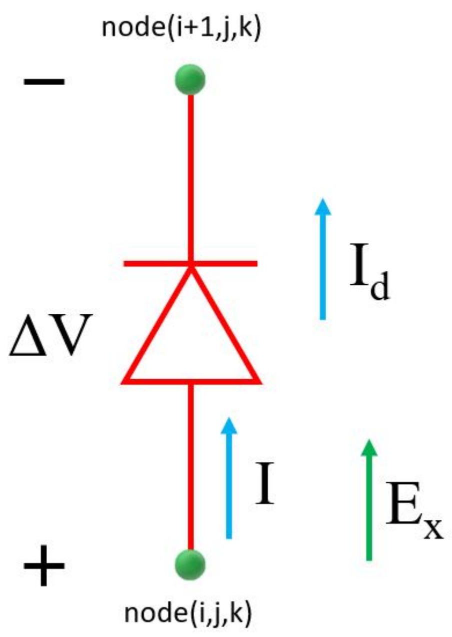


Figure 2.10 A circuit schematic of a diode oriented in the +X direction.

The I-V characteristic of a diode is non linear. Based on the semiconductor properties of the diode, the voltage-current relationship takes an exponential form:

$$I_d = I_0 \left(e^{qV_d/kT} - 1 \right), \tag{2.101}$$

where I_d is the current across the diode, V_d is the voltage across the diode, q is the absolute value of the electron charge in coulombs, k is Boltzman's constant, and T is the temperature in Kelvin.

Since equation 2.101 must be converted to discrete form, and we must solve for the current at the time step of $n + \frac{1}{2}$ in order to substitute equation 2.101 into equation 2.61, the following substitution for V_d is made:

$$V_d^{n+\frac{1}{2}}(i,j,k) = \Delta x E_x^{n+\frac{1}{2}}(i,j,k) = \Delta x \left(\frac{E_x^{n+1}(i,j,k) + E_x^n(i,j,k)}{2} \right). \tag{2.102}$$

Equation 2.102 assumes a V_d in the +X direction. Note that $E_x^{n+\frac{1}{2}}$ is calculated by taking the simple time average of E_x^{n+1} and E_x^n.

Substituting equation 2.102 into equation 2.101 gives the discrete form of the I-V characteristic for a diode:

$$I_d^{n+\frac{1}{2}}(i,j,k) = I_0(i,j,k) \left(e^{q\Delta x \left(\frac{E_x^{n+1}(i,j,k) + E_x^n(i,j,k)}{2} \right) / kT} - 1 \right), \tag{2.103}$$

simplifying we have:

$$I_d^{n+\frac{1}{2}}(i,j,k) = I_0(i,j,k)\left(e^{q\Delta x E_x^{n+1}(i,j,k)/2kT} e^{q\Delta x E_x^{n}(i,j,k)/2kT} - 1\right). \tag{2.104}$$

Finally, equation 2.104 is ready to be substituted into equation 2.61 by realizing $I_d^{n+\frac{1}{2}}(i,j,k) = I_{lx}^{n+\frac{1}{2}}(i,j,k)$

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\alpha \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\beta \\ & +\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}E_x^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)}\frac{I_0(i,j,k)\left(e^{q\Delta x E_x^{n+1}(i,j,k)/2kT} e^{q\Delta x E_x^{n}(i,j,k)/2kT} - 1\right)}{\Delta z\Delta y}. \end{aligned} \tag{2.105}$$

Due to the non linearity of the I-V charectoristic of the diode, E_x in equation 2.105 cannot be solved for explicitly. However, equation 2.105 can be rearranged such that E_x can be solved for using a zero crossing method such as Newton's method. In order to make the algebra more clear, let $(2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)) = \gamma$ and $(2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)) = \eta$.

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{\gamma}\alpha - \frac{2\Delta t}{\gamma}\beta + \frac{\eta}{\gamma}E_x^n(i,j,k) \\ & -\frac{2\Delta t}{\gamma}\frac{I_0(i,j,k)\left(e^{q\Delta x E_x^{n+1}(i,j,k)/2kT} e^{q\Delta x E_x^{n}(i,j,k)/2kT} - 1\right)}{\Delta z\Delta y} \end{aligned} \tag{2.106}$$

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{\gamma}\alpha - \frac{2\Delta t}{\gamma}\beta + \frac{\eta}{\gamma}E_x^n(i,j,k) \\ & -\frac{2\Delta t I_0(i,j,k)e^{\frac{q\Delta x}{2kT}E_x^n(i,j,k)}}{\gamma\Delta z\Delta y}e^{\frac{q\Delta x}{2kT}E_x^{n+1}(i,j,k)} + \frac{2\Delta t I_0(i,j,k)}{\gamma\Delta z\Delta y} \end{aligned} \tag{2.107}$$

Knowing that E_x^{n+1} is the only unknown in equation 2.107, the following substitutions can be made:

$$A = \frac{2\Delta t I_0(i,j,k)e^{\frac{q\Delta x}{2kT}E_x^n(i,j,k)}}{\gamma\Delta z\Delta y} \tag{2.108}$$

$$B = \frac{q\Delta x}{2kT} \tag{2.109}$$

$$C = -\frac{2\Delta t}{\gamma}\alpha + \frac{2\Delta t}{\gamma}\beta - \frac{\eta}{\gamma}E_x^n(i,j,k) - \frac{2\Delta t I_0(i,j,k)}{\gamma\Delta z\Delta y} \tag{2.110}$$

$$x = E_x^{n+1}, \tag{2.111}$$

additionally, substituting back in for γ and η, the most general updating equation for any order accuracy is derived:

$$A = \frac{2\Delta t I_0(i,j,k) e^{\frac{q\Delta x}{2kT} E_x^n(i,j,k)}}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}, \tag{2.112}$$

$$B = \frac{q\Delta x}{2kT}, \tag{2.113}$$

$$\begin{aligned} C = & -\frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\alpha \\ & + \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}\beta \\ & - \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t I_0(i,j,k)}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}, \end{aligned} \tag{2.114}$$

with these substitutions, equation 2.107 can be written as:

$$x = -Ae^{Bx} - C, \tag{2.115}$$

Or,

$$Ae^{Bx} + x + C = 0. \tag{2.116}$$

Finally, "x" can be solved for in equation 2.116 by using Newton's iterative method for zero finding. This iterative process will happen for every time step in order to calculate E_x^{n+1}.

The full forms of A, B, and C for second order are shown in equations 2.117, 2.118 and 2.119 by substituting second order approximate derivatives back in for α and β.

$$A = \frac{2\Delta t I_0(i,j,k) e^{\frac{q\Delta x}{2kT} E_x^n(i,j,k)}}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}, \tag{2.117}$$

$$B = \frac{q\Delta x}{2kT}, \tag{2.118}$$

$$\begin{aligned} C = & -\frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \frac{H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\ & + \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\ & - \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & - \frac{2\Delta t I_0(i,j,k)}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}. \end{aligned} \tag{2.119}$$

The full forms of A, B, and C for fourth order are shown in equations 2.120, 2.121 and 2.122 by substituting fourth order approximate derivatives back in for α and β.

$$A = \frac{2\Delta t I_0(i,j,k) e^{\frac{q\Delta x}{2kT} E_x^n(i,j,k)}}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}, \tag{2.120}$$

$$B = \frac{q\Delta x}{2kT}, \tag{2.121}$$

$$\begin{aligned} C = &- \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \\ &* \frac{-H_z^{n+\frac{1}{2}}(i,j+1,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i,j-1,k) + H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\ &+ \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} \\ &* \frac{-H_y^{n+\frac{1}{2}}(i,j,k+1) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i,j,k-1) + H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\ &- \frac{2\epsilon_x(i,j,k) - \Delta t \sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ &- \frac{2\Delta t I_0(i,j,k)}{(2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k))\Delta z \Delta y}. \end{aligned} \tag{2.122}$$

2.2.3 MATLAB implementation of Fourth Order Updating Equations

In order to apply the fourth order finite differencing derivative approximations to the FDTD code, a deeper understanding of the Yee cell layout is needed. In the following sections, the cross sections of planes in 3D space are presented to help illustrate the physical locations of the field components required to update the electric and magnetic field components. From these physical locations, the forth order updating equations can be easily implemented into the code. Additionally, the limit for Δt needs to be re-evaluated due to the shape of the fourth order mask. The length of Δt for the second order formulation is determined based on the fact that the electromagnetic fields cannot travel faster than the speed of light across one Yee cell length. However, since the forth order code pulls from two Yee cell lengths, the formulation must be re-evaluated. The fourth order Δt formulation is presented after the figures of the field components.

2.2.3.1 Fourth Order E_x Updating Equations

Figure 2.11, Figure 2.12, and Figure 2.13 show the details of how the E_x component of the electric field is updated.

Calculating E-Field dots from H-Field arrows

$$\frac{\partial E_x}{\partial t} = \frac{1}{\varepsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right)$$

Green: Field component updated
with fourth order accuracy

Red: Field component
never updated (PEC layers)

X Range: 1:Nx

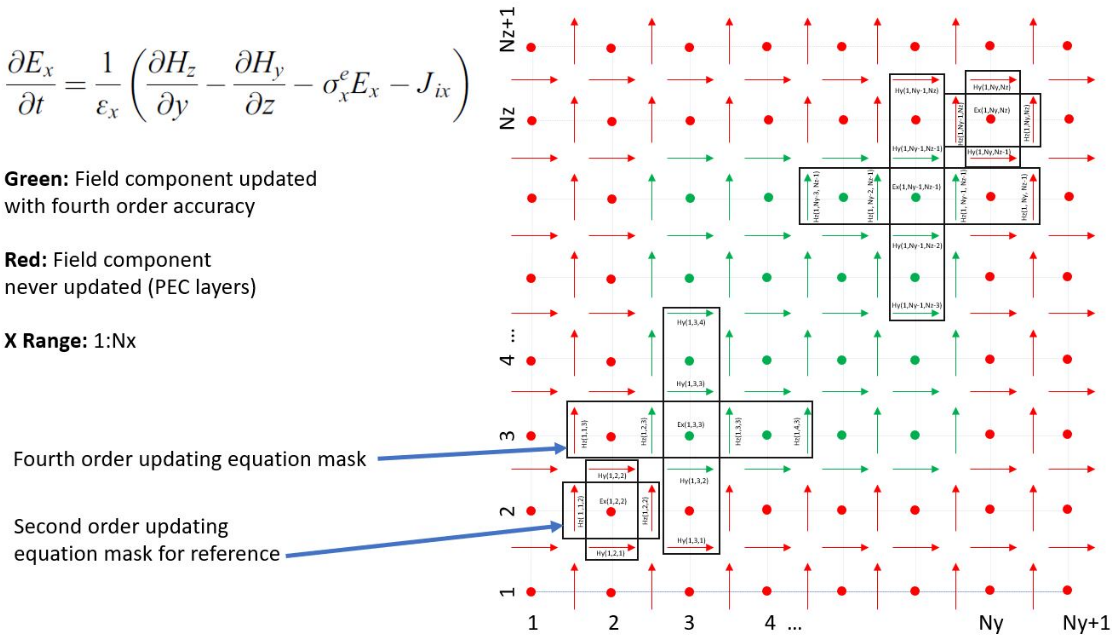


Figure 2.11 Yee cells used to calculate E_x.

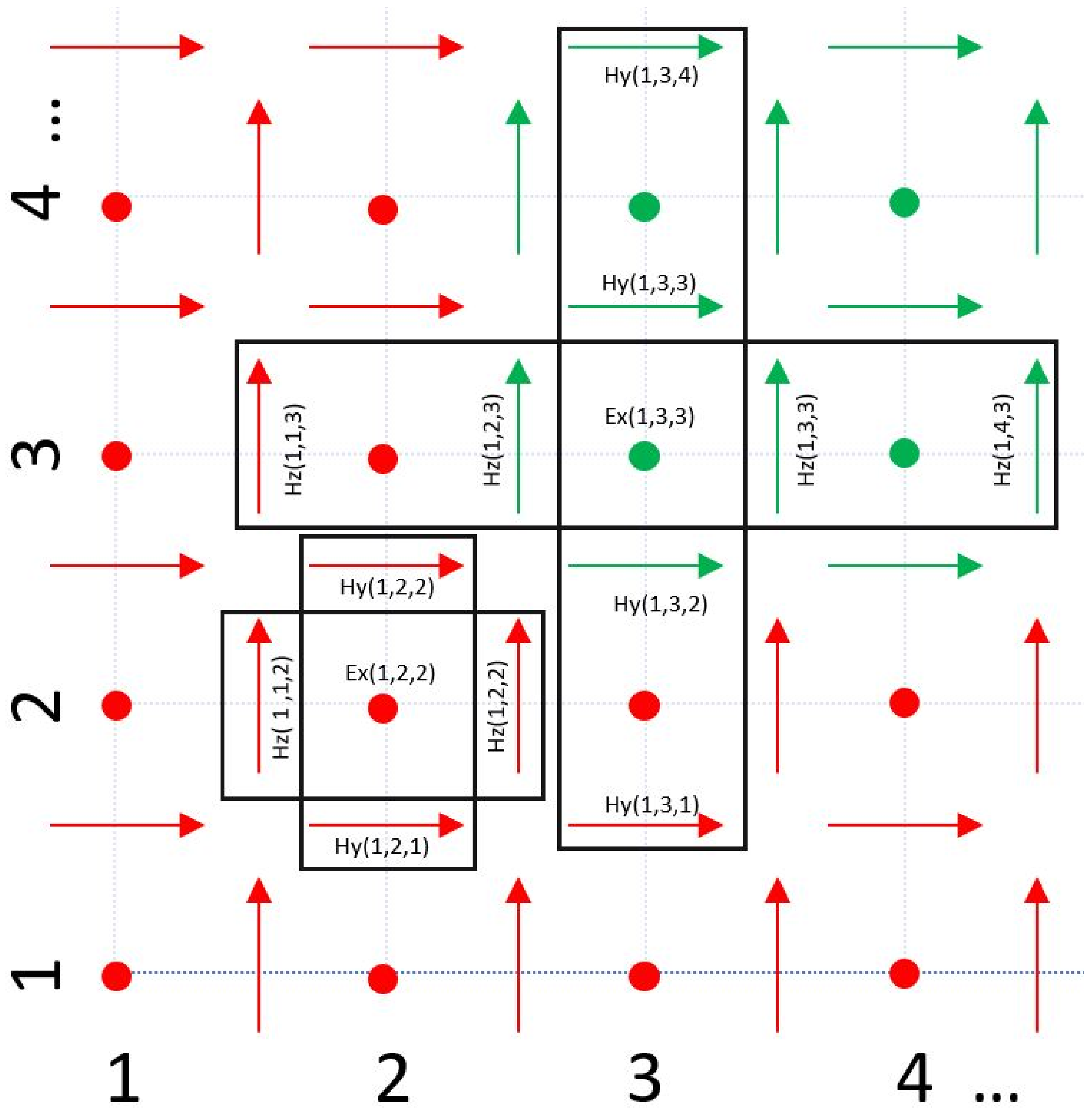


Figure 2.12 Bottom limit of Yee cells used to calculate E_x.

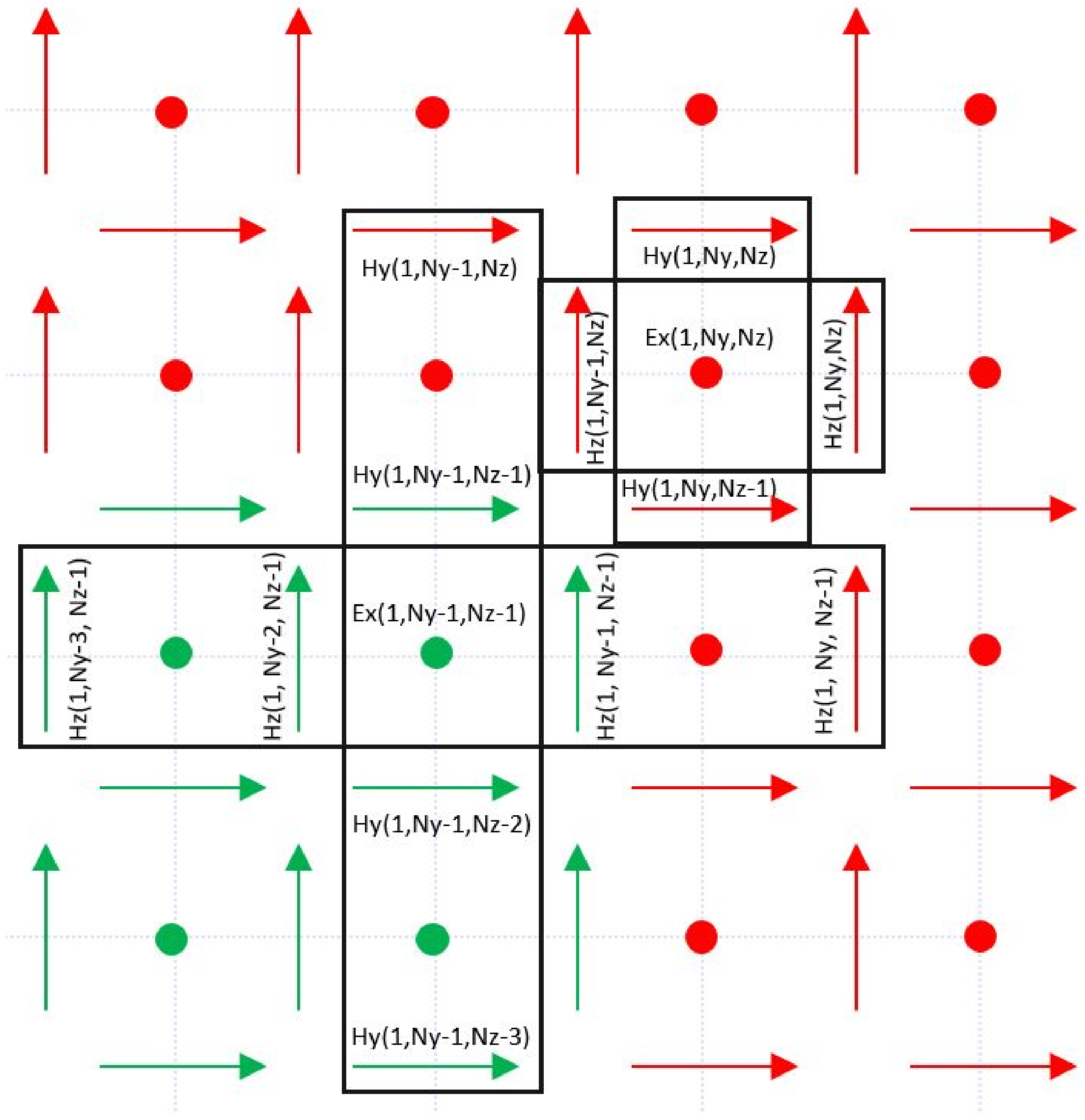


Figure 2.13 Top limit of Yee cells used to calculate E_x.

Even though only E_x is shown in the figures, E_y and E_z can be depicted in the same manor. The final electric field updating equations are as follows.

$$\begin{aligned} Ex(1:nx,3:ny-1,3:nz-1) =& Cexe(1:nx,3:ny-1,3:nz-1).*Ex(1:nx,3:ny-1,3:nz-1)...\\ &+Cexhz(1:nx,3:ny-1,3:nz-1).*...\\ &\ \ (-Hz(1:nx,4:ny,3:nz-1)+27*Hz(1:nx,3:ny-1,3:nz-1)\\ &-27*Hz(1:nx,2:ny-2,3:nz-1)+Hz(1:nx,1:ny-3,3:nz-1))/24...\\ &+Cexhy(1:nx,3:ny-1,3:nz-1).*...\\ &\ \ (-Hy(1:nx,3:ny-1,4:nz)+27*Hy(1:nx,3:ny-1,3:nz-1)\\ &-27*Hy(1:nx,3:ny-1,2:nz-2)+Hy(1:nx,3:ny-1,1:nz-3))/24; \end{aligned} \tag{2.123}$$

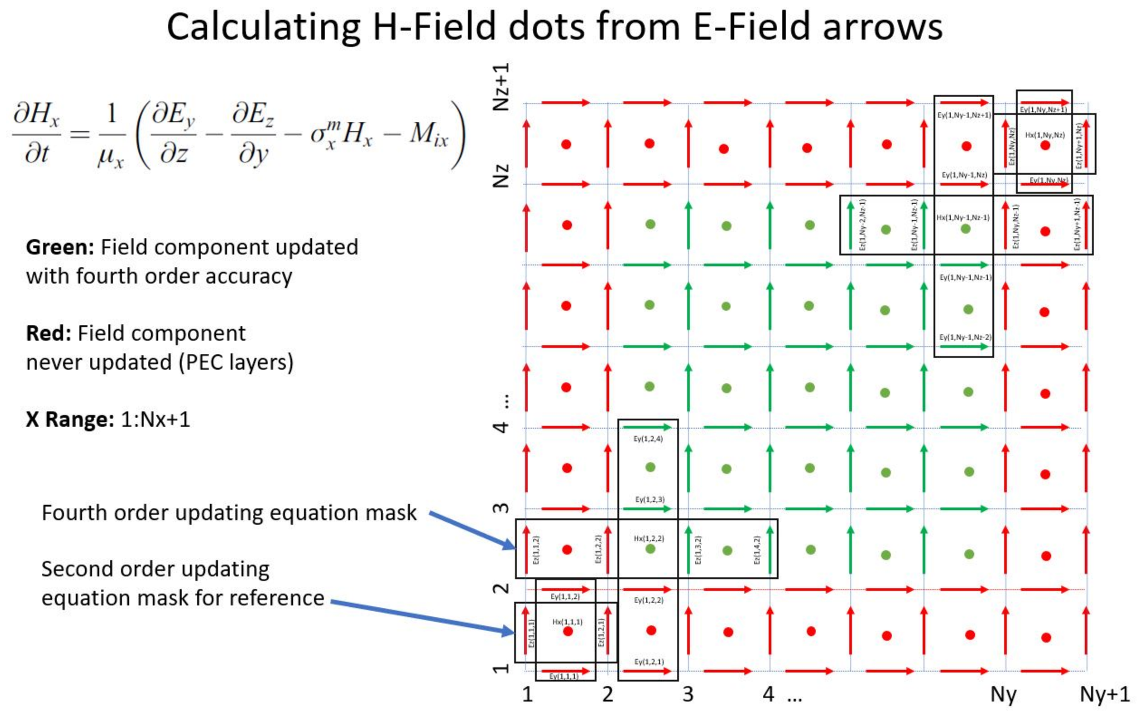
Note that the indexing of E_x in equation 2.123 is unique. The range of E_x values that are able to be updated with fourth order FDTD does not encompass the total number of E_x components in the domain. The range of E_x components that can be updated with fourth order updating equations is limited by the size of the fourth order updating equation mask. As shown in Figure 2.12, the lowest Y and Z index of E_x that can be updated with the fourth order mask is 3. As shown in Figure 2.13, the highest Y and Z index of E_x that can be updated with fourth order mask is $N_y - 1$ and $N_z - 1$. Since the fourth order updating mask for E_x only extends in the Y and Z directions, the range of the X index of E_x can range from 1 to N_x. Similar reasoning applies to indexes shown in equations 2.124 and 2.125.

$$\begin{aligned} Ey(3:nx-1,1:ny,3:nz-1) =& Ceye(3:nx-1,1:ny,3:nz-1).*Ey(3:nx-1,1:ny,3:nz-1)...\\ &+Ceyhx(3:nx-1,1:ny,3:nz-1).*...\\ &\ (-Hx(3:nx-1,1:ny,4:nz)+27*Hx(3:nx-1,1:ny,3:nz-1)\\ &-27*Hx(3:nx-1,1:ny,2:nz-2)+Hx(3:nx-1,1:ny,1:nz-3))/24...\\ &+Ceyhz(3:nx-1,1:ny,3:nz-1).*...\\ &\ (-Hz(4:nx,1:ny,3:nz-1)+27*Hz(3:nx-1,1:ny,3:nz-1)\\ &-27*Hz(2:nx-2,1:ny,3:nz-1)+Hz(1:nx-3,1:ny,3:nz-1))/24; \end{aligned} \tag{2.124}$$

$$\begin{aligned} Ez(3:nx-1,3:ny-1,1:nz) =& Ceze(3:nx-1,3:ny-1,1:nz).*Ez(3:nx-1,3:ny-1,1:nz)...\\ &+Cezhy(3:nx-1,3:ny-1,1:nz).*...\\ &\ (-Hy(4:nx,3:ny-1,1:nz)+27*Hy(3:nx-1,3:ny-1,1:nz)\\ &-27*Hy(2:nx-2,3:ny-1,1:nz)+Hy(1:nx-3,3:ny-1,1:nz))/24...\\ &+Cezhx(3:nx-1,3:ny-1,1:nz).*...\\ &\ (-Hx(3:nx-1,4:ny,1:nz)+27*Hx(3:nx-1,3:ny-1,1:nz)\\ &-27*Hx(3:nx-1,2:ny-2,1:nz)+Hx(3:nx-1,1:ny-3,1:nz))/24; \end{aligned} \tag{2.125}$$

2.2.3.2 Fourth Order H_x Updating Equations

Figure 2.14, Figure 2.15, and Figure 2.16 show the details of how the H_x component of the magnetic field is updated.


Calculating H-Field dots from E-Field arrows

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right)$$

Green: Field component updated with fourth order accuracy

Red: Field component never updated (PEC layers)

X Range: 1:Nx+1

Fourth order updating equation mask

Second order updating equation mask for reference

Figure 2.14 Yee cells used to calculate H_x.

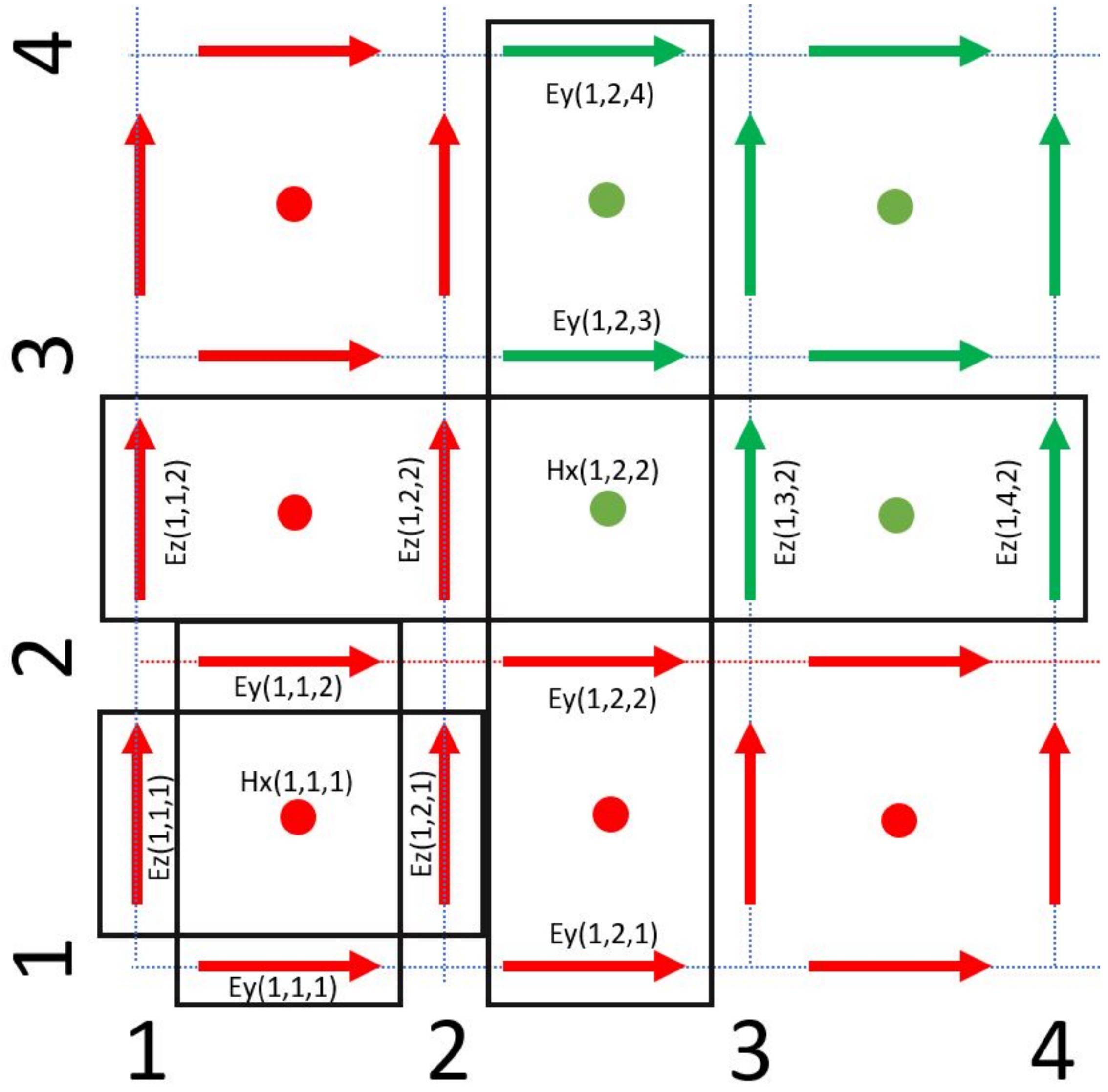


Figure 2.15 Bottom limit of Yee cells used to calculate H_x.

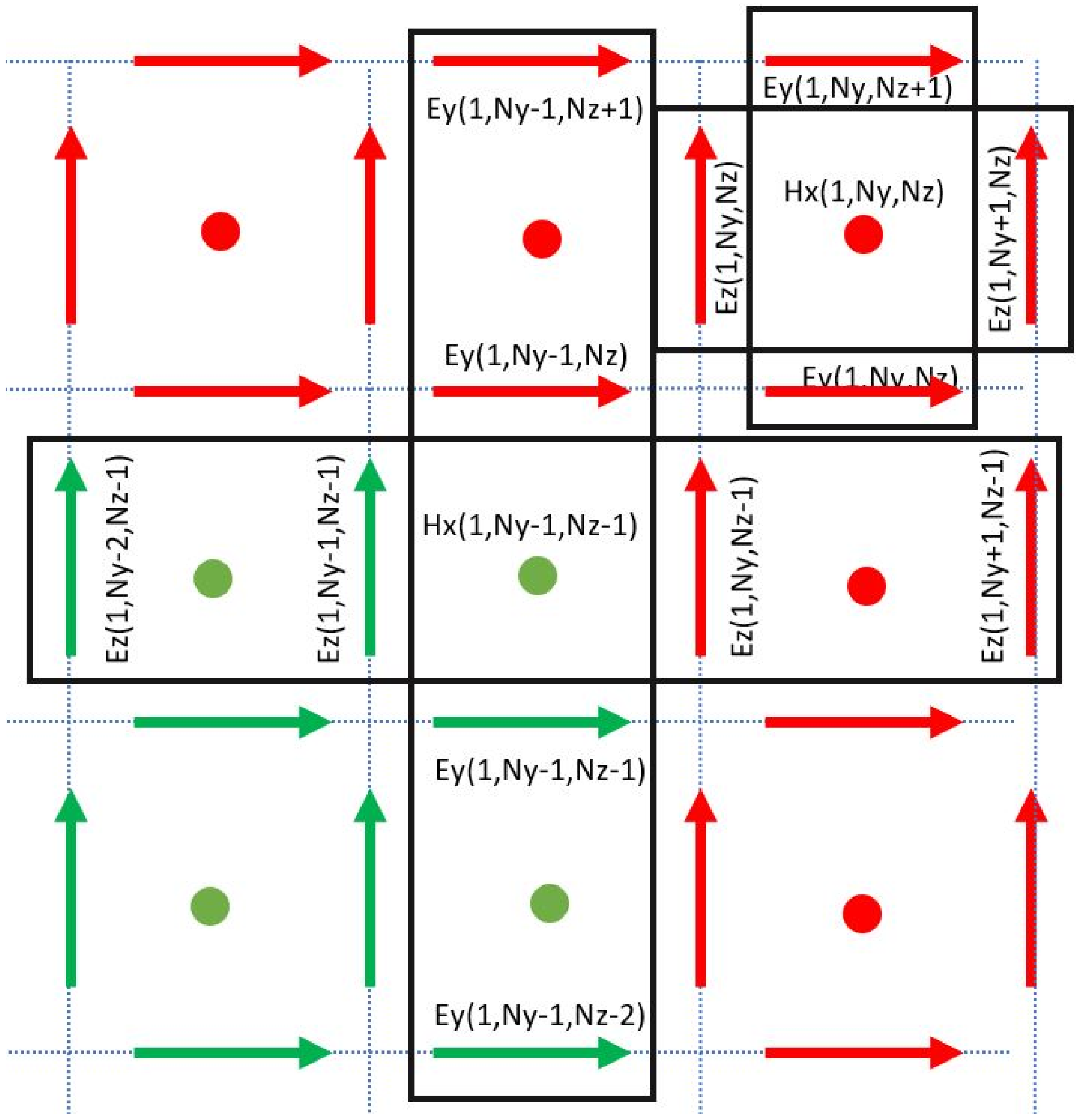


Figure 2.16 Top limit of Yee cells used to calculate H_x.

Even though only H_x is shown in the figures, H_y and H_z can be depicted in the same manor. The final magnetic field updating equations are as follows.

$$\begin{aligned}
Hx(1:nx+1,2:ny-1,2:nz-1) =& Chxh(1:nx+1,2:ny-1,2:nz-1).*Hx(1:nx+1,2:ny-1,2:nz-1)...\\
&+Chxey(1:nx+1,2:ny-1,2:nz-1).*...\\
&\;(-Ey(1:nx+1,2:ny-1,4:nz+1)+27*Ey(1:nx+1,2:ny-1,3:nz)\\
&-27*Ey(1:nx+1,2:ny-1,2:nz-1)+Ey(1:nx+1,2:ny-1,1:nz-2))/24...\\
&+Chxez(1:nx+1,2:ny-1,2:nz-1).*...\\
&\;(-Ez(1:nx+1,4:ny+1,2:nz-1)+27*Ez(1:nx+1,3:ny,2:nz-1)\\
&-27*Ez(1:nx+1,2:ny-1,2:nz-1)+Ez(1:nx+1,1:ny-2,2:nz-1))/24;
\end{aligned} \tag{2.126}$$

Note that the indexing of H_x in equation 2.126 is unique. The range of H_x values that are able to be updated with fourth order FDTD does not encompass the total number of H_x components in the domain. The range of H_x components that can be updated with fourth order updating equations is limited by the size of the fourth order updating equation mask. As shown in Figure 2.15, the lowest Y and Z index of H_x that can be updated with the fourth order mask is 2. As shown in Figure 2.16, the highest Y and Z index of H_x that can be updated with fourth order mask is $N_y - 1$ and $N_z - 1$. Since the fourth order updating mask for H_x only extends in the Y and Z directions, the range of the X index of H_x can range from 1 to $N_x + 1$. Similar reasoning applies to indexes shown in equations 2.127 and 2.128.

$$\begin{aligned} Hy(2:nx-1,1:ny+1,2:nz-1) =& Chyh(2:nx-1,1:ny+1,2:nz-1).*Hy(2:nx-1,1:ny+1,2:nz-1)...\\ &+Chyez(2:nx-1,1:ny+1,2:nz-1).*...\\ &\ (-Ez(4:nx+1,1:ny+1,2:nz-1)+27*Ez(3:nx,1:ny+1,2:nz-1)\\ &-27*Ez(2:nx-1,1:ny+1,2:nz-1)+Ez(1:nx-2,1:ny+1,2:nz-1))/24...\\ &+Chyex(2:nx-1,1:ny+1,2:nz-1).*...\\ &\ (-Ex(2:nx-1,1:ny+1,4:nz+1)+27*Ex(2:nx-1,1:ny+1,3:nz)\\ &-27*Ex(2:nx-1,1:ny+1,2:nz-1)+Ex(2:nx-1,1:ny+1,1:nz-2))/24; \end{aligned} \tag{2.127}$$

$$\begin{aligned} Hz(2:nx-1,2:ny-1,1:nz+1) =& Chzh(2:nx-1,2:ny-1,1:nz+1).*Hz(2:nx-1,2:ny-1,1:nz+1)...\\ &+Chzex(2:nx-1,2:ny-1,1:nz+1).*...\\ &\ (-Ex(2:nx-1,4:ny+1,1:nz+1)+27*Ex(2:nx-1,3:ny,1:nz+1)\\ &-27*Ex(2:nx-1,2:ny-1,1:nz+1)+Ex(2:nx-1,1:ny-2,1:nz+1))/24...\\ &+Chzey(2:nx-1,2:ny-1,1:nz+1).*...\\ &\ (-Ey(4:nx+1,2:ny-1,1:nz+1)+27*Ey(3:nx,2:ny-1,1:nz+1)\\ &-27*Ey(2:nx-1,2:ny-1,1:nz+1)+Ey(1:nx-2,2:ny-1,1:nz+1))/24; \end{aligned} \tag{2.128}$$

2.2.3.3 Fourth Order Stability Criteria

According to the Courant-Friedrich-Lewy (CFL) condition [1], the constraint on Δt for a second order simulation is shown in equations 2.129 and 2.130

$$\Delta t \leq \frac{1}{c}\frac{1}{\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}}} \tag{2.129}$$

or

$$c\Delta t\sqrt{\frac{1}{\Delta x^2}+\frac{1}{\Delta y^2}+\frac{1}{\Delta z^2}} \leq 1 \tag{2.130}$$

In 1D this relation can be written as:

$$\Delta t \leq \frac{\Delta x}{c} \tag{2.131}$$

or

$$c\Delta t \leq \Delta x \tag{2.132}$$

Ultimately, the CFL condition for a second order simulation states that during one time step a wave cannot travel more than one step in space and ensures the wave does not travel faster than the speed of light.

The fourth order CFL condition is much more complicated to derive. Pages 153-154 of [23] show the full derivation. For the fourth order simulation, the stability criterion is [23].

$$\Delta t \leq \frac{1}{c} \frac{6/7}{\sqrt{\frac{1}{\Delta x^2} + \frac{1}{\Delta y^2} + \frac{1}{\Delta z^2}}}. \tag{2.133}$$

Furthermore the optimum Δt is obtained by [24].

$$\Delta t_{optimum} = \frac{\Delta t_{max}}{0.335 N_c + 0.040}, \tag{2.134}$$

where N_c is the number of cells per wavelength of the maximum frequency of interest.

Equation 2.134 is not explored in detail in this thesis. However, section 4.1.1 explores the effect of the Courant factor on the accuracy of the second and fourth order simulations.

2.2.4 Fourth Order Thin Wire Formulation

The improved thin wire formulation presented in [1] does not need to be modified extensively to implement a fourth order accurate thin wire simulation. The thin wire formulation in [1] implements the thin wire formulation by changing the coefficients on the second order magnetic field updating equations. Figure 2.17 explains further:

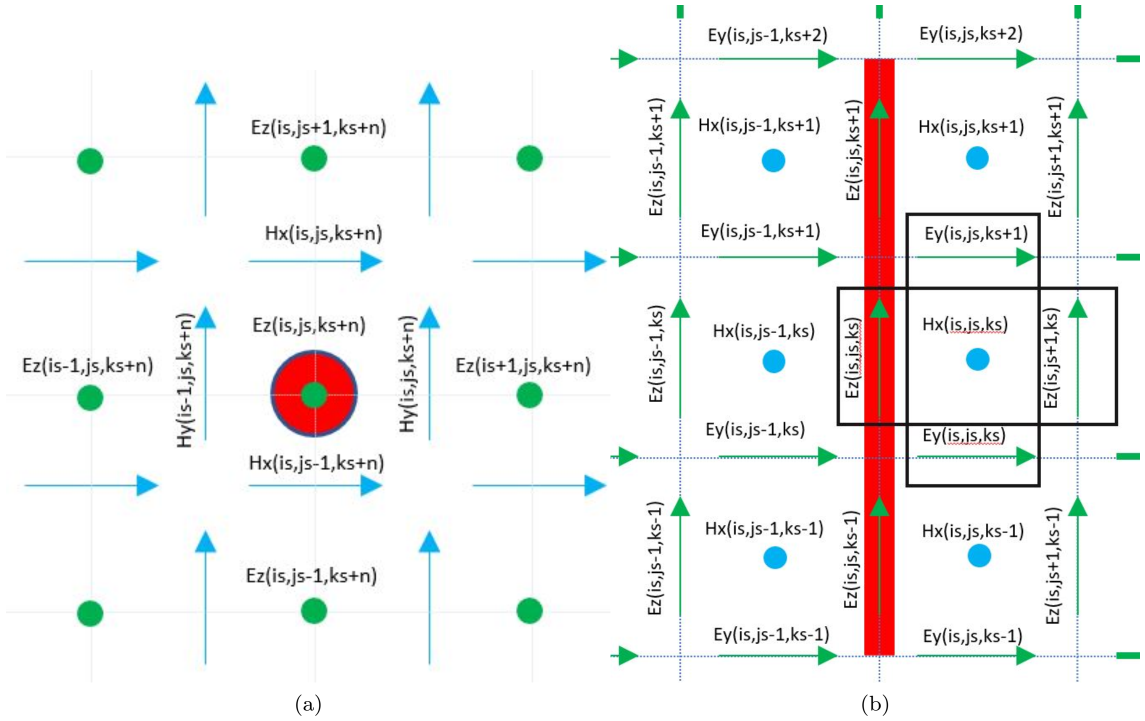


Figure 2.17 Field indices near a thin wire oriented in the Z direction.

Figure 2.17(a) shows the field indices in the XY-plane where the Z direction is coming out of the page. The E_z components of the electric field are shown as green dots while the magnetic field components in the X and Y directions are shown as blue arrows. The thin wire is shown as a large red dot. The updating of the E_z component of the electric field depends solely on the H_x and H_y magnetic fields. The updating of the electric fields is not directly effected by the thin wire, meaning the fourth order updating equations for E_z still hold around a thin wire. In fact, all electric field components near a thin wire can be updated with unaltered fourth order updating equations.

Even though the electric fields are not directly effected by the thin wire, the magnetic fields used to calculate the electric fields are directly effected. This means the fourth order updating equations for the magnetic fields near the thin wire need to account for the thin wire. The fourth order updating equation mask uses two electric field components on either side of a magnetic field component. When the magnetic field component is only one electric field component away from the thin wire, the fourth order updating equation mask should not be able to use the second electric field on the other side of the thin wire. This is due to the fact that the thin wire is PEC and any field component in the wire will be zero, creating a discontinuous electric field. As section 4.8.1 shows, fourth order updating equations do not handle discontinuities correctly. Additionally, the PEC of the thin wire will mask the effects of the electric field component on the other side of the thin wire on the magnetic field component being updated. For these reasons, the magnetic field components perpendicular and directly adjacent to the thin wire are updated with second order updating equations. Figure 2.17(b) shows the resulting second order updating mask used near the thin wire. By using this technique, the improved thin wire formulation presented in [1] does not need to change at all.

Collapsing the fourth order magnetic field updating equations to second order near the thin wire requires a careful look into the field indices near the thin wire as shown in Figure 2.17(b). In Figure 2.17(b), the H_x components of the magnetic field are oriented out of the page and shown as blue dots. The E_z and E_y components of the electric field are shown as green arrows. The thin wire is shown as a thick red line. The black boxes are meant to illustrate the mask of the second order updating equation for the H_x field component. A diagram outlining the H_y, E_x, and E_z field components showing how the H_y field component to be updated would be almost identical to Figure 2.17(b). As described in [1] the only components that are specially updated due to the thin wire are the magnetic field components perpendicular and directly next to the thin wire. These are the H_x components shown in Figure 2.17(b). Furthermore, as shown by the second order updating mask, the E_z components in the Y direction are effected by the presence of the thin wire. This means only the $\approx \frac{\partial E_z}{\partial y}$ in the updating equation for H_x will be effected by the thin wire. The $\approx \frac{\partial E_y}{\partial z}$ in the updating equation for H_x will still be collapsed to second order but will not be effected by the thin wire. Listing Listing 2.2 shows the details of this methodology for thin wires oriented in different directions.

Listing 2.2: initialize thin wire updating coefficients

```
disp('initializing thin wire updating coefficients');

dtm = dt/mu_0;

for ind = 1:number_of_thin_wires
    is = round((thin_wires(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((thin_wires(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((thin_wires(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((thin_wires(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((thin_wires(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((thin_wires(ind).max_z - fdtd_domain.min_z)/dz)+1;
    r_o = thin_wires(ind).radius;

    switch (thin_wires(ind).direction(1))
        case 'x'
            Cexe (is:ie-1,js,ks) = 0;
            Cexhy(is:ie-1,js,ks) = 0;
            Cexhz(is:ie-1,js,ks) = 0;
            Chyh (is:ie-1,js,ks-1:ks) = 1;
            Chyez(is:ie-1,js,ks-1:ks) = dtm ...
                ./ (mu_r_y(is:ie-1,js,ks-1:ks) * dx);
            Chyex(is:ie-1,js,ks-1:ks) = -2 * dtm ...
                ./ (mu_r_y(is:ie-1,js,ks-1:ks) * dz * log(dz/r_o));
            Chzh( is:ie-1,js-1:js,ks) = 1;
            Chzex(is:ie-1,js-1:js,ks) = 2 * dtm ...
                ./ (mu_r_z(is:ie-1,js-1:js,ks) * dy * log(dy/r_o));
            Chzey(is:ie-1,js-1:js,ks) = -dtm ...
                ./ (mu_r_z(is:ie-1,js-1:js,ks) * dx);
        case 'y'
            Ceye (is,js:je-1,ks) = 0;
            Ceyhx(is,js:je-1,ks) = 0;
            Ceyhz(is,js:je-1,ks) = 0;
            Chzh (is-1:is,js:je-1,ks) = 1;
            Chzex(is-1:is,js:je-1,ks) = dtm ...
                ./ (mu_r_z(is-1:is,js:je-1,ks) * dy);
            Chzey(is-1:is,js:je-1,ks) = -2 * dtm ...
                ./ (mu_r_z(is-1:is,js:je-1,ks) * dx * log(dx/r_o));
            Chxh (is,js:je-1,ks-1:ks) = 1;
            Chxey(is,js:je-1,ks-1:ks) = 2 * dtm ...
                ./ (mu_r_x(is,js:je-1,ks-1:ks) * dz * log(dz/r_o));
            Chxez(is,js:je-1,ks-1:ks) = -dtm ...
                ./ (mu_r_x(is,js:je-1,ks-1:ks) * dy);
        case 'z'
            khx = (dy/dx)*atan(dx/dy);
            khy = (dx/dy)*atan(dy/dx);
            Ceze (is,js,ks:ke-1) = 0;
            Cezhx(is,js,ks:ke-1) = 0;
            Cezhy(is,js,ks:ke-1) = 0;
            Chxh (is,js-1:js,ks:ke-1) = 1;
%             Chxey(is,js-1:js,ks:ke-1) = -2*dtm*khx ...
%                 ./ (mu_r_x(is,js-1:js,ks:ke-1) * dz);
            Chxez(is,js-1:js,ks:ke-1) = -2 * dtm * khx...
                ./ (mu_r_x(is,js-1:js,ks:ke-1) * dy * log(dy/r_o));
            Chyh (is-1:is,js,ks:ke-1) = 1;
            Chyez(is-1:is,js,ks:ke-1) = 2 * dtm * khy...
                ./ (mu_r_y(is-1:is,js,ks:ke-1) * dx * log(dx/r_o));
%             Chyex(is-1:is,js,ks:ke-1) = -dtm ...
%                 ./ (mu_r_y(is-1:is,js,ks:ke-1) * dz);
    end
end
```

The simplest way to implement second order updating near the thin wire, and fourth order updating everywhere else, is to update the entire domain with fourth order first. Then, go back over the field components near the thin wire with second order updating. The magnetic field components near the thin wire must be corrected to second order every time step before the electric fields are updated. This ensures that the electric fields are not calculated with the incorrect magnetic field components.

Listing Listing 2.3 shows the modified update_magnetic_fields code. The last line of listing Listing 2.3 calls the code that will go back over the field components near the thin wire and correct them to be second order updating equations. Correcting a magnetic field component to second order updating after it has already been updated with fourth order updating is complicated. Both the second and fourth order updating equations use the field component as a parameter to update the same field component, as would be expected by the name "updating equation". The code uses the previously stored value for the field component and over-writes it with the new value every time it is updated. This means that by the time the thin_wire_B code is called, the previously stored value of the field component has been inaccurately updated with fourth order updating and cannot be used to update itself using second order updating.

The easiest way to restore the field component to its original value (undo the fourth order updating) is to simply perform the fourth order updating equation in reverse. After this is done the field component can be correctly updated with second order updating. Listing Listing 2.4 shows the details of this method.

Listing 2.3: update magnetic fields (thin wire)

```
% update magnetic fields
current_time  = current_time + dt/2;

%second order
% Hx = Chxh.*Hx+Chxey.*(Ey(1:nxp1,1:ny,2:nzp1)-Ey(1:nxp1,1:ny,1:nz)) ...
%      + Chxez.*(Ez(1:nxp1,2:nyp1,1:nz)-Ez(1:nxp1,1:ny,1:nz));
%
% Hy = Chyh.*Hy+Chyez.*(Ez(2:nxp1,1:nyp1,1:nz)-Ez(1:nx,1:nyp1,1:nz)) ...
%      + Chyex.*(Ex(1:nx,1:nyp1,2:nzp1)-Ex(1:nx,1:nyp1,1:nz));
%
% Hz = Chzh.*Hz+Chzex.*(Ex(1:nx,2:nyp1,1:nzp1)-Ex(1:nx,1:ny,1:nzp1))   ...
%      + Chzey.*(Ey(2:nxp1,1:ny,1:nzp1)-Ey(1:nx,1:ny,1:nzp1));

%Fourth Order
%+2, +1, 0, -1

HxXb = 1+pecTony; %1
HxXt = nx+1-pecTony; %nx+1
HxYb = 2+pecTony;
HxYt = ny-1-pecTony;
HxZb = 2+pecTony;
HxZt = nz-1-pecTony;

Hx(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt) = Chxh(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*Hx(HxXb:
    HxXt,HxYb:HxYt,HxZb:HxZt)...
    + Chxey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
    (-Ey(HxXb:HxXt,HxYb:HxYt,HxZb+2:HxZt+2)+27*Ey(HxXb:HxXt,HxYb:HxYt,HxZb+1:HxZt+1)
        -27*Ey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)+Ey(HxXb:HxXt,HxYb:HxYt,HxZb-1:HxZt-1))
        /24 ...%dz
    + Chxez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
    (-Ez(HxXb:HxXt,HxYb+2:HxYt+2,HxZb:HxZt)+27*Ez(HxXb:HxXt,HxYb+1:HxYt+1,HxZb:HxZt)
        -27*Ez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)+Ez(HxXb:HxXt,HxYb-1:HxYt-1,HxZb:HxZt))
        /24; %dy

HyXb = 2+pecTony;
HyXt = nx-1-pecTony;
HyYb = 1+pecTony; %1
HyYt = ny+1-pecTony; %ny+1
HyZb = 2+pecTony;
HyZt = nz-1-pecTony;

Hy(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt) = Chyh(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*Hy(HyXb:
    HyXt,HyYb:HyYt,HyZb:HyZt)...
    + Chyez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
    (-Ez(HyXb+2:HyXt+2,HyYb:HyYt,HyZb:HyZt)+27*Ez(HyXb+1:HyXt+1,HyYb:HyYt,HyZb:HyZt)
        -27*Ez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)+Ez(HyXb-1:HyXt-1,HyYb:HyYt,HyZb:HyZt))
        /24 ... %dx
    + Chyex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
    (-Ex(HyXb:HyXt,HyYb:HyYt,HyZb+2:HyZt+2)+27*Ex(HyXb:HyXt,HyYb:HyYt,HyZb+1:HyZt+1)
        -27*Ex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)+Ex(HyXb:HyXt,HyYb:HyYt,HyZb-1:HyZt-1))
        /24; %dz

HzXb = 2+pecTony;
HzXt = nx-1-pecTony;
HzYb = 2+pecTony;
HzYt = ny-1-pecTony;
HzZb = 1+pecTony; %1
HzZt = nz+1-pecTony; %nz+1

Hz(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt) = Chzh(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt).*Hz(HzXb:
    HzXt,HzYb:HzYt,HzZb:HzZt)...
    + Chzex(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt).*...
    (-Ex(HzXb:HzXt,HzYb+2:HzYt+2,HzZb:HzZt)+27*Ex(HzXb:HzXt,HzYb+1:HzYt+1,HzZb:HzZt)
        -27*Ex(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt)+Ex(HzXb:HzXt,HzYb-1:HzYt-1,HzZb:HzZt))
        /24 ... %dy
    + Chzey(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt).*...
    (-Ey(HzXb+2:HzXt+2,HzYb:HzYt,HzZb:HzZt)+27*Ey(HzXb+1:HzXt+1,HzYb:HzYt,HzZb:HzZt)
        -27*Ey(HzXb:HzXt,HzYb:HzYt,HzZb:HzZt)+Ey(HzXb-1:HzXt-1,HzYb:HzYt,HzZb:HzZt))
        /24; %dx

thin_wire_B;
```

Listing 2.4: thin wire B

```

%find regions of affected field componants for S24 updating
for ind = 1:number_of_thin_wires

    % convert coordinates to node indices on the FDTD grid
    is = round((thin_wires(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((thin_wires(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((thin_wires(ind).min_z - fdtd_domain.min_z)/dz)+1;

    ie = round((thin_wires(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((thin_wires(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((thin_wires(ind).max_z - fdtd_domain.min_z)/dz)+1;

        HxXb = is; %1
        HxXt = is; %nx+1
        HxYb = js-1;
        HxYt = js;
        HxZb = ks;
        HxZt = ke-1;
        %Remove the badness that the S24 updating equation did.
        Hx(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt) = (Hx(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)...
            -(Chxey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
            (-Ey(HxXb:HxXt,HxYb:HxYt,HxZb+2:HxZt+2)+27*Ey(HxXb:HxXt,HxYb:HxYt,HxZb
                +1:HxZt+1)-27*Ey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)+Ey(HxXb:HxXt,HxYb:
                HxYt,HxZb-1:HxZt-1))/24 ...%dz
            + Chxez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
            (-Ez(HxXb:HxXt,HxYb+2:HxYt+2,HxZb:HxZt)+27*Ez(HxXb:HxXt,HxYb+1:HxYt+1,
                HxZb:HxZt)-27*Ez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)+Ez(HxXb:HxXt,HxYb-1:
                HxYt-1,HxZb:HxZt))/24))./Chxh(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt); %dy
        %update Hx using second order
        Hx(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt) = Chxh(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*Hx(
            HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)...
            + Chxey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
            (Ey(HxXb:HxXt,HxYb:HxYt,HxZb+1:HxZt+1)-Ey(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)
                ) ...%dz   %%
            + Chxez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt).*...
            (Ez(HxXb:HxXt,HxYb+1:HxYt+1,HxZb:HxZt)-Ez(HxXb:HxXt,HxYb:HxYt,HxZb:HxZt)
                ); %dy

        HyXb = is-1;
        HyXt = is;
        HyYb = js; %1
        HyYt = js; %ny+1
        HyZb = ks;
        HyZt = ke-1;
        %Remove the badness that the S24 updating equation did.
        Hy(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt) = Hy(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)...
            -((Chyez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
            (-Ez(HyXb+2:HyXt+2,HyYb:HyYt,HyZb:HyZt)+27*Ez(HyXb+1:HyXt+1,HyYb:HyYt,
                HyZb:HyZt)-27*Ez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)+Ez(HyXb-1:HyXt-1,HyYb
                :HyYt,HyZb:HyZt))/24 ... %dx
            + Chyex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
            (-Ex(HyXb:HyXt,HyYb:HyYt,HyZb+2:HyZt+2)+27*Ex(HyXb:HyXt,HyYb:HyYt,HyZb
                +1:HyZt+1)-27*Ex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)+Ex(HyXb:HyXt,HyYb:
                HyYt,HyZb-1:HyZt-1))/24))./Chyh(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt); %dz
        %update Hy using second order
        Hy(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt) = Chyh(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*Hy(
            HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)...
            + Chyez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
            (Ez(HyXb+1:HyXt+1,HyYb:HyYt,HyZb:HyZt)-Ez(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)
                ) ... %dx
            + Chyex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt).*...
            (Ex(HyXb:HyXt,HyYb:HyYt,HyZb+1:HyZt+1)-Ex(HyXb:HyXt,HyYb:HyYt,HyZb:HyZt)
                ); %dz

end
```

2.2.5 Fourth Order Plane Wave Implementation

The implementation of a fourth order simulation with a plane wave source only requires minor changes to the second order implementation shown in [1]. Given a working second order code, only the update_electric_fields and update_magnetic_fields require changes to achieve a fourth order simulation. As shown in section 2.2.1, the fourth order derivative approximation can be easily substituted for the second order derivative approximation without changing the coefficients or form of the updating equation. This simplicity also applies when implementing the plane wave formulation in a fourth order simulation. The following code listings explain further.

The first step in implementing a plane wave source in the FDTD code is defining the attributes of the plane wave (see Listing 2.5). This code most importantly defines the incident angle of the plane wave and the waveform of the pane wave.

Listing 2.5: define sources and lumped elements

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];
incident_plane_wave = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% Define incident plane wave, angles are in degrees
incident_plane_wave.E_theta = 1;
incident_plane_wave.E_phi = 0;
incident_plane_wave.theta_incident = 45;
incident_plane_wave.phi_incident = 30;
incident_plane_wave.waveform_type = 'gaussian';
incident_plane_wave.waveform_index = 1;
```

The next step is to initialize the updating coefficients for the plane wave updating equations. Code listings Listing 2.6 and Listing 2.7 show how the updating coefficients are calculated. Note that these code listings match the codes presented in [1] as the fourth order updating equations are simple enough not to effect the updating coefficients. Code listing Listing 2.8 shows how more parameters of the plane wave are calculated. Again none of these calculations are effected by the switch to fourth order updating equations.

Listing 2.6: initialize updating coefficients

```
disp('initializing general updating coefficients');

% General electric field updating coefficients
% Coeffiecients updating Ex
Cexe  =  (2*eps_r_x*eps_0 - dt*sigma_e_x) ...
      ./(2*eps_r_x*eps_0 + dt*sigma_e_x);
Cexhz =  (2*dt/dy)./(2*eps_r_x*eps_0 + dt*sigma_e_x);
Cexhy = -(2*dt/dz)./(2*eps_r_x*eps_0 + dt*sigma_e_x);

% Coeffiecients updating Ey
Ceye  =  (2*eps_r_y*eps_0 - dt*sigma_e_y) ...
      ./(2*eps_r_y*eps_0 + dt*sigma_e_y);
Ceyhx =  (2*dt/dz)./(2*eps_r_y*eps_0 + dt*sigma_e_y);
Ceyhz = -(2*dt/dx)./(2*eps_r_y*eps_0 + dt*sigma_e_y);

% Coeffiecients updating Ez
```

```
Ceze  =  (2*eps_r_z*eps_0 - dt*sigma_e_z) ...
     ./(2*eps_r_z*eps_0 + dt*sigma_e_z);
Cezhy =  (2*dt/dx)./(2*eps_r_z*eps_0 + dt*sigma_e_z);
Cezhx = -(2*dt/dy)./(2*eps_r_z*eps_0 + dt*sigma_e_z);

% General magnetic field updating coefficients
% Coeffiecients updating Hx
Chxh  =  (2*mu_r_x*mu_0 - dt*sigma_m_x) ...
     ./(2*mu_r_x*mu_0 + dt*sigma_m_x);
Chxez = -(2*dt/dy)./(2*mu_r_x*mu_0 + dt*sigma_m_x);
Chxey =  (2*dt/dz)./(2*mu_r_x*mu_0 + dt*sigma_m_x);

% Coeffiecients updating Hy
Chyh  =  (2*mu_r_y*mu_0 - dt*sigma_m_y) ...
     ./(2*mu_r_y*mu_0 + dt*sigma_m_y);
Chyex = -(2*dt/dz)./(2*mu_r_y*mu_0 + dt*sigma_m_y);
Chyez =  (2*dt/dx)./(2*mu_r_y*mu_0 + dt*sigma_m_y);

% Coeffiecients updating Hz
Chzh  =  (2*mu_r_z*mu_0 - dt*sigma_m_z) ...
     ./(2*mu_r_z*mu_0 + dt*sigma_m_z);
Chzey = -(2*dt/dx)./(2*mu_r_z*mu_0 + dt*sigma_m_z);
Chzex =  (2*dt/dy)./(2*mu_r_z*mu_0 + dt*sigma_m_z);

% Initialize coeffiecients for lumped element components
initialize_voltage_source_updating_coefficients;
initialize_current_source_updating_coefficients;
initialize_resistor_updating_coefficients;
initialize_capacitor_updating_coefficients;
initialize_inductor_updating_coefficients;
initialize_diode_updating_coefficients;
initialize_incident_field_updating_coefficients;
```

Listing 2.7: initialize incident field updating coefficients

```
% initialize incident field updating coefficients

if incident_plane_wave.enabled == false
    return;
end

% Coeffiecients updating Ex
Cexeic= (2*(1-eps_r_x)*eps_0-dt*sigma_e_x) ...
     ./(2*eps_r_x*eps_0+dt*sigma_e_x);
Cexeip=-(2*(1-eps_r_x)*eps_0+dt*sigma_e_x) ...
     ./(2*eps_r_x*eps_0+dt*sigma_e_x);

% Coeffiecients updating Ey
Ceyeic= (2*(1-eps_r_y)*eps_0-dt*sigma_e_y) ...
     ./(2*eps_r_y*eps_0+dt*sigma_e_y);
Ceyeip=-(2*(1-eps_r_y)*eps_0+dt*sigma_e_y) ...
     ./(2*eps_r_y*eps_0+dt*sigma_e_y);

% Coeffiecients updating Ez
Cezeic= (2*(1-eps_r_z)*eps_0-dt*sigma_e_z) ...
     ./(2*eps_r_z*eps_0+dt*sigma_e_z);
Cezeip=-(2*(1-eps_r_z)*eps_0+dt*sigma_e_z) ...
```

```
    ./(2*eps_r_z*eps_0+dt*sigma_e_z);

% Coeffiecients updating Hx
Chxhic= (2*(1-mu_r_x)*mu_0-dt*sigma_m_x)./(2*mu_r_x*mu_0+dt*sigma_m_x);
Chxhip=-(2*(1-mu_r_x)*mu_0+dt*sigma_m_x)./(2*mu_r_x*mu_0+dt*sigma_m_x);

% Coeffiecients updating Hy
Chyhic= (2*(1-mu_r_y)*mu_0-dt*sigma_m_y)./(2*mu_r_y*mu_0+dt*sigma_m_y);
Chyhip=-(2*(1-mu_r_y)*mu_0+dt*sigma_m_y)./(2*mu_r_y*mu_0+dt*sigma_m_y);

% Coeffiecients updating Hz
Chzhic=(2*(1-mu_r_z)*mu_0-dt*sigma_m_z)./(2*mu_r_z*mu_0+dt*sigma_m_z);
Chzhip=-(2*(1-mu_r_z)*mu_0+dt*sigma_m_z)./(2*mu_r_z*mu_0+dt*sigma_m_z);
```

Listing 2.8: initialize sources and lumped elements

```
% initialize incident plane wave
if isfield(incident_plane_wave,'E_theta')
    incident_plane_wave.enabled = true;
else
    incident_plane_wave.enabled = false;
end

if incident_plane_wave.enabled
    % create incident field arrays for current time step
    Hxic = zeros(nxp1,ny,nz);
    Hyic = zeros(nx,nyp1,nz);
    Hzic = zeros(nx,ny,nzp1);
    Exic = zeros(nx,nyp1,nzp1);
    Eyic = zeros(nxp1,ny,nzp1);
    Ezic = zeros(nxp1,nyp1,nz);
    % create incident field arrays for previous time step
    Hxip = zeros(nxp1,ny,nz);
    Hyip = zeros(nx,nyp1,nz);
    Hzip = zeros(nx,ny,nzp1);
    Exip = zeros(nx,nyp1,nzp1);
    Eyip = zeros(nxp1,ny,nzp1);
    Ezip = zeros(nxp1,nyp1,nz);

    % calculate the amplitude factors for field components
    theta_incident = incident_plane_wave.theta_incident*pi/180;
    phi_incident = incident_plane_wave.phi_incident*pi/180;
    E_theta = incident_plane_wave.E_theta;
    E_phi = incident_plane_wave.E_phi;
    eta_0 = sqrt(mu_0/eps_0);
    Exi0 = E_theta * cos(theta_incident) * cos(phi_incident) ...
        - E_phi * sin(phi_incident);
    Eyi0 = E_theta * cos(theta_incident) * sin(phi_incident) ...
        + E_phi * cos(phi_incident);
    Ezi0 = -E_theta * sin(theta_incident);
    Hxi0 = (-1/eta_0)*(E_phi * cos(theta_incident) ...
        * cos(phi_incident) + E_theta * sin(phi_incident));
    Hyi0 = (-1/eta_0)*(E_phi * cos(theta_incident) ...
        * sin(phi_incident) - E_theta * cos(phi_incident));
    Hzi0 = (1/eta_0)*(E_phi * sin(theta_incident));

    % Create position arrays indicating the coordinates of the nodes
    x_pos = zeros(nxp1,nyp1,nzp1);
    y_pos = zeros(nxp1,nyp1,nzp1);
    z_pos = zeros(nxp1,nyp1,nzp1);
    for ind = 1:nxp1
        x_pos(ind,:,:) = (ind - 1) * dx + fdtd_domain.min_x;
    end
    for ind = 1:nyp1
        y_pos(:,ind,:) = (ind - 1) * dy + fdtd_domain.min_y;
    end
    for ind = 1:nzp1
        z_pos(:,:,ind) = (ind - 1) * dz + fdtd_domain.min_z;
    end

    % calculate spatial shift, l_0, required for incident plane wave
    r0 =[fdtd_domain.min_x fdtd_domain.min_y fdtd_domain.min_z;
    fdtd_domain.min_x fdtd_domain.min_y fdtd_domain.max_z;
    fdtd_domain.min_x fdtd_domain.max_y fdtd_domain.min_z;
    fdtd_domain.min_x fdtd_domain.max_y fdtd_domain.max_z;
    fdtd_domain.max_x fdtd_domain.min_y fdtd_domain.min_z;
    fdtd_domain.max_x fdtd_domain.min_y fdtd_domain.max_z;
    fdtd_domain.max_x fdtd_domain.max_y fdtd_domain.min_z;
    fdtd_domain.max_x fdtd_domain.max_y fdtd_domain.max_z;];

    k_vec_x =  sin(theta_incident)*cos(phi_incident);
    k_vec_y =  sin(theta_incident)*sin(phi_incident);
    k_vec_z =  cos(theta_incident);

    k_dot_r0 = k_vec_x * r0(:,1) ...
        + k_vec_y * r0(:,2) ...
        + k_vec_z * r0(:,3);

    l_0 = min(k_dot_r0)/c;

    % calculate k.r for every field component
    k_dot_r_ex = ((x_pos(1:nx,1:nyp1,1:nzp1)+dx/2) * k_vec_x ...
        + y_pos(1:nx,1:nyp1,1:nzp1) * k_vec_y ...
        + z_pos(1:nx,1:nyp1,1:nzp1) * k_vec_z)/c;

    k_dot_r_ey = (x_pos(1:nxp1,1:ny,1:nzp1) * k_vec_x ...
        + (y_pos(1:nxp1,1:ny,1:nzp1)+dy/2) * k_vec_y ...
        + z_pos(1:nxp1,1:ny,1:nzp1) * k_vec_z)/c;

    k_dot_r_ez = (x_pos(1:nxp1,1:nyp1,1:nz) * k_vec_x ...
        + y_pos(1:nxp1,1:nyp1,1:nz) * k_vec_y ...
        + (z_pos(1:nxp1,1:nyp1,1:nz)+dz/2) * k_vec_z)/c;

    k_dot_r_hx = (x_pos(1:nxp1,1:ny,1:nz) * k_vec_x ...
        + (y_pos(1:nxp1,1:ny,1:nz)+dy/2) * k_vec_y ...
        + (z_pos(1:nxp1,1:ny,1:nz)+dz/2) * k_vec_z)/c;

    k_dot_r_hy = ((x_pos(1:nx,1:nyp1,1:nz)+dx/2) * k_vec_x ...
        + y_pos(1:nx,1:nyp1,1:nz) * k_vec_y ...
        + (z_pos(1:nx,1:nyp1,1:nz)+dz/2) * k_vec_z)/c;

    k_dot_r_hz = ((x_pos(1:nx,1:ny,1:nzp1)+dx/2) * k_vec_x ...
        + (y_pos(1:nx,1:ny,1:nzp1)+dy/2) * k_vec_y ...
        + z_pos(1:nx,1:ny,1:nzp1) * k_vec_z)/c;

    % embed spatial shift in k.r
    k_dot_r_ex = k_dot_r_ex - l_0;
    k_dot_r_ey = k_dot_r_ey - l_0;
    k_dot_r_ez = k_dot_r_ez - l_0;
    k_dot_r_hx = k_dot_r_hx - l_0;
    k_dot_r_hy = k_dot_r_hy - l_0;
    k_dot_r_hz = k_dot_r_hz - l_0;

    % store the waveform
    wt_str = incident_plane_wave.waveform_type;
    wi_str = num2str(incident_plane_wave.waveform_index);
    eval_str = ['a_waveform = waveforms.' ...
        wt_str '(' wi_str ').waveform;'];
    eval(eval_str);
    incident_plane_wave.waveform = a_waveform;

    clear x_pos y_pos z_pos;
end
```

Finally, the time marching loop (Listing 2.9) is ready to be ran and the field components updated at every time step. This part of the code also updates the plane wave as the plane wave is also dependent on time. Updating the incident fields (shown in code Listing 2.10) is not effected by changing to fourth order accurate updating equations.

Listing 2.9: run fdtd time marching loop

```
disp (['Starting the time marching loop']);
disp(['Total number of time steps : ' ...
    num2str(number_of_time_steps)]);

start_time = cputime;
current_time = 0;

for time_step = 1:number_of_time_steps
    update_incident_fields;
    update_magnetic_fields;
    update_magnetic_field_CPML_ABC;
    capture_sampled_magnetic_fields;
    capture_sampled_currents;
    update_electric_fields;
    update_electric_field_CPML_ABC;
    update_voltage_sources;
    update_current_sources;
    update_inductors;
    update_diodes;
    capture_sampled_electric_fields;
    capture_sampled_voltages;
    calculate_JandM;
    display_sampled_parameters;
end

end_time = cputime;
total_time_in_minutes = (end_time - start_time)/60;
disp(['Total simulation time is ' ...
    num2str(total_time_in_minutes) ' minutes.']);
```

Listing 2.10: update incident fields

```
% update incident fields for the current time step
if incident_plane_wave.enabled == false
    return;
end

tm = current_time + dt/2;
te = current_time + dt;

% update incident fields for previous time step
Hxip = Hxic; Hyip = Hyic; Hzip = Hzic;
Exip = Exic; Eyip = Eyic; Ezip = Ezic;

wt_str = incident_plane_wave.waveform_type;
wi = incident_plane_wave.waveform_index;

% if waveform is Gaussian waveforms
if strcmp(incident_plane_wave.waveform_type,'gaussian')
    tau = waveforms.gaussian(wi).tau;
    t_0 = waveforms.gaussian(wi).t_0;
    Exic = Exi0 * exp(-((te - t_0 - k_dot_r_ex )/tau).^2);
    Eyic = Eyi0 * exp(-((te - t_0 - k_dot_r_ey )/tau).^2);
    Ezic = Ezi0 * exp(-((te - t_0 - k_dot_r_ez )/tau).^2);
    Hxic = Hxi0 * exp(-((tm - t_0 - k_dot_r_hx )/tau).^2);
    Hyic = Hyi0 * exp(-((tm - t_0 - k_dot_r_hy )/tau).^2);
    Hzic = Hzi0 * exp(-((tm - t_0 - k_dot_r_hz )/tau).^2);
end

% if waveform is derivative of Gaussian
if strcmp(incident_plane_wave.waveform_type,'derivative_gaussian')
    tau = waveforms.derivative_gaussian(wi).tau;
    t_0 = waveforms.derivative_gaussian(wi).t_0;
    Exic = Exi0 * (-sqrt(2*exp(1))/tau)*(te - t_0 - k_dot_r_ex) ...
           .*exp(-((te - t_0 - k_dot_r_ex)/tau).^2);
    Eyic = Eyi0 * (-sqrt(2*exp(1))/tau)*(te - t_0 - k_dot_r_ey) ...
           .*exp(-((te - t_0 - k_dot_r_ey)/tau).^2);
    Ezic = Ezi0 * (-sqrt(2*exp(1))/tau)*(te - t_0 - k_dot_r_ez) ...
           .*exp(-((te - t_0 - k_dot_r_ez)/tau).^2);
    Hxic = Hxi0 * (-sqrt(2*exp(1))/tau)*(tm - t_0 - k_dot_r_hx) ...
           .*exp(-((tm - t_0 - k_dot_r_hx)/tau).^2);
    Hyic = Hyi0 * (-sqrt(2*exp(1))/tau)*(tm - t_0 - k_dot_r_hy) ...
           .*exp(-((tm - t_0 - k_dot_r_hy)/tau).^2);
    Hzic = Hzi0 * (-sqrt(2*exp(1))/tau)*(tm - t_0 - k_dot_r_hz) ...
           .*exp(-((tm - t_0 - k_dot_r_hz)/tau).^2);
end

% if waveform is cosine modulated Gaussian
if strcmp(incident_plane_wave.waveform_type, ...
    'cosine_modulated_gaussian')
    f = waveforms.cosine_modulated_gaussian(wi).modulation_frequency;
    tau = waveforms.cosine_modulated_gaussian(wi).tau;
    t_0 = waveforms.cosine_modulated_gaussian(wi).t_0;
    Exic = Exi0 * cos(2*pi*f*(te - t_0 - k_dot_r_ex)) ...
           .*exp(-((te - t_0 - k_dot_r_ex)/tau).^2);
    Eyic = Eyi0 * cos(2*pi*f*(te - t_0 - k_dot_r_ey)) ...
           .*exp(-((te - t_0 - k_dot_r_ey)/tau).^2);
    Ezic = Ezi0 * cos(2*pi*f*(te - t_0 - k_dot_r_ez)) ...
```

```
2.57             .*exp(-((te - t_0 - k_dot_r_ez)/tau).^2);
2.58      Hxic = Hxi0 * cos(2*pi*f*(tm - t_0 - k_dot_r_hx)) ...
2.59             .*exp(-((tm - t_0 - k_dot_r_hx)/tau).^2);
2.60      Hyic = Hyi0 * cos(2*pi*f*(tm - t_0 - k_dot_r_hy)) ...
2.61             .*exp(-((tm - t_0 - k_dot_r_hy)/tau).^2);
2.62      Hzic = Hyi0 * cos(2*pi*f*(tm - t_0 - k_dot_r_hz)) ...
2.63             .*exp(-((tm - t_0 - k_dot_r_hz)/tau).^2);
2.64 end
```

Code listing Listing 2.11 shows the final form of the fourth order electric field updating equations. Note that lines 48-70 are added to the end of the update_electric_fields code for both the second and fourth order simulations and account for the plane wave moving through the simulation. Note that the variable "pecTony" is set to 8 to ensure the plane wave does not exist inside the CPML boundaries that are 8 cells thick. The update_magnetic_fields code has a similar form.

Listing 2.11: update electric fields

```
2.1  % update electric fields except the tangential components
2.2  % on the boundaries
2.3
2.4  current_time  = current_time + dt/2;
2.5
2.6  %Fourth Order
2.7  %+1, 0, -1, -2
2.8
2.9  ExXb = 1; %1
2.10 ExXt = nx; %nx
2.11 ExYb = 3;
2.12 ExYt = ny-1;
2.13 ExZb = 3;
2.14 ExZt = nz-1;
2.15
2.16 Ex(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) = Cexe(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Ex(ExXb:
         ExXt,ExYb:ExYt,ExZb:ExZt) ...
2.17                              + Cexhz(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*...
2.18                              (-Hz(ExXb:ExXt,ExYb+1:ExYt+1,ExZb:ExZt)+27*Hz(ExXb:ExXt,ExYb:
                                      ExYt,ExZb:ExZt)-27*Hz(ExXb:ExXt,ExYb-1:ExYt-1,ExZb:ExZt)+Hz
                                      (ExXb:ExXt,ExYb-2:ExYt-2,ExZb:ExZt))/24 ... %dy
2.19                              + Cexhy(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*...
2.20                              (-Hy(ExXb:ExXt,ExYb:ExYt,ExZb+1:ExZt+1)+27*Hy(ExXb:ExXt,ExYb:
                                      ExYt,ExZb:ExZt)-27*Hy(ExXb:ExXt,ExYb:ExYt,ExZb-1:ExZt-1)+Hy
                                      (ExXb:ExXt,ExYb:ExYt,ExZb-2:ExZt-2))/24; %dz
2.21
2.22 EyXb = 3;
2.23 EyXt = nx-1;
2.24 EyYb = 1; %1
2.25 EyYt = ny; %ny
2.26 EyZb = 3;
2.27 EyZt = nz-1;
2.28
2.29 Ey(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)=Ceye(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Ey(EyXb:EyXt,
         EyYb:EyYt,EyZb:EyZt) ...
2.30                              + Ceyhx(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*    ...
```

```
                              (-Hx(EyXb:EyXt,EyYb:EyYt,EyZb+1:EyZt+1)+27*Hx(EyXb:EyXt,EyYb:
                                   EyYt,EyZb:EyZt)-27*Hx(EyXb:EyXt,EyYb:EyYt,EyZb-1:EyZt-1)+Hx
                                   (EyXb:EyXt,EyYb:EyYt,EyZb-2:EyZt-2))/24  ...  %dz
                              + Ceyhz(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*    ...
                              (-Hz(EyXb+1:EyXt+1,EyYb:EyYt,EyZb:EyZt)+27*Hz(EyXb:EyXt,EyYb:
                                   EyYt,EyZb:EyZt)-27*Hz(EyXb-1:EyXt-1,EyYb:EyYt,EyZb:EyZt)+Hz
                                   (EyXb-2:EyXt-2,EyYb:EyYt,EyZb:EyZt))/24; %dx

EzXb = 3;
EzXt = nx-1;
EzYb = 3;
EzYt = ny-1;
EzZb = 1; %1
EzZt = nz; %nz

Ez(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)=Ceze(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Ez(EzXb:EzXt,
    EzYb:EzYt,EzZb:EzZt) ...
                              + Cezhy(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*    ...
                              (-Hy(EzXb+1:EzXt+1,EzYb:EzYt,EzZb:EzZt)+27*Hy(EzXb:EzXt,EzYb:
                                   EzYt,EzZb:EzZt)-27*Hy(EzXb-1:EzXt-1,EzYb:EzYt,EzZb:EzZt)+Hy
                                   (EzXb-2:EzXt-2,EzYb:EzYt,EzZb:EzZt))/24  ...  %dx
                              + Cezhx(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*...
                              (-Hx(EzXb:EzXt,EzYb+1:EzYt+1,EzZb:EzZt)+27*Hx(EzXb:EzXt,EzYb:
                                   EzYt,EzZb:EzZt)-27*Hx(EzXb:EzXt,EzYb-1:EzYt-1,EzZb:EzZt)+Hx
                                   (EzXb:EzXt,EzYb-2:EzYt-2,EzZb:EzZt))/24;

ExXb = 1+pecTony; %1
ExXt = nx-pecTony; %nx
ExYb = 3+pecTony;
ExYt = ny-1-pecTony;
ExZb = 3+pecTony;
ExZt = nz-1-pecTony;
EyXb = 3+pecTony;
EyXt = nx-1-pecTony;
EyYb = 1+pecTony; %1
EyYt = ny-pecTony; %ny
EyZb = 3+pecTony;
EyZt = nz-1-pecTony;
EzXb = 3+pecTony;
EzXt = nx-1-pecTony;
EzYb = 3+pecTony;
EzYt = ny-1-pecTony;
EzZb = 1+pecTony; %1
EzZt = nz-pecTony; %nz
if incident_plane_wave.enabled
    Ex(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) = Ex(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) + Cexeic(
        ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) .* Exic(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) +
        Cexeip(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) .* Exip(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt);
    Ey(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt) = Ey(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt) + Ceyeic(
        EyXb:EyXt,EyYb:EyYt,EyZb:EyZt) .* Eyic(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt) +
        Ceyeip(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt) .* Eyip(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt);
    Ez(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt) = Ez(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt) + Cezeic(
        EzXb:EzXt,EzYb:EzYt,EzZb:EzZt) .* Ezic(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt) +
        Cezeip(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt) .* Ezip(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt);
end
```

2.2.6 Fourth Order Near-field to Far-field Transformation

No changes need to be made to the near-field to far-field transformation presented in [1] to achieve a fourth order simulation. The near-field to far-field transformation procedure as implemented for individual frequencies is not a function of time and is calculated after the simulation is complete. Since the fourth order code only changes the updating of field components during the simulation, the post process of calculating the electric and magnetic surface currents for the far field evaluation is not effected.

2.3 2D FDTD Formulation

This section discusses the two dimensional FDTD derivation. In order to go from a 3D derivation to a 2D derivation, the parts of Maxwell's equations involving change in the third dimension can be ignored. Then is it clear two sets of three equations can be de-coupled from each other. These two sets of equations then represent the 2D TE or TM modes of propagation.

2.3.1 Material Updating Equations

From Maxwell's 2 curl equations, there are 6 component level equations:

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right), \tag{2.135}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon_y}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma_y^e E_y - J_{iy}\right), \tag{2.136}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon_z}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma_z^e E_z - J_{iz}\right), \tag{2.137}$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right), \tag{2.138}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu_y}\left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} - \sigma_y^m H_y - M_{iy}\right), \tag{2.139}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right). \tag{2.140}$$

For a two dimensional case where X and Y are the only dimensions that vary, any derivative with respect to Z is zero. Once the approximations are made we are left with four component level equations:

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \sigma_x^e E_x - J_{ix}\right), \tag{2.141}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon_y}\left(-\frac{\partial H_z}{\partial x} - \sigma_y^e E_y - J_{iy}\right), \tag{2.142}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon_z}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma_z^e E_z - J_{iz}\right), \tag{2.143}$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(-\frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right), \tag{2.144}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu_y}\left(\frac{\partial E_z}{\partial x} - \sigma_y^m H_y - M_{iy}\right), \tag{2.145}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right). \tag{2.146}$$

Similar to the 1D case, these 2D Maxwell equations also have two groups of coupled equations. The following group of equations represent a TM_z mode:

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(-\frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right), \tag{2.147}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu_y}\left(\frac{\partial E_z}{\partial x} - \sigma_y^m H_y - M_{iy}\right), \tag{2.148}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon_z}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma_z^e E_z - J_{iz}\right). \tag{2.149}$$

The following group of equations represent a TE_z mode:

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon_x}\left(\frac{\partial H_z}{\partial y} - \sigma_x^e E_x - J_{ix}\right), \tag{2.150}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon_y}\left(-\frac{\partial H_z}{\partial x} - \sigma_y^e E_y - J_{iy}\right), \tag{2.151}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right). \tag{2.152}$$

From here similar steps to section 2.2.1 can be taken to arrive at the 2D updating equations for the appropriate mode. No results for 2D FDTD simulations are presented in this thesis.

2.4 1D FDTD Formulation

Now that the 3D and 2D FDTD formulations are understood, it is of interest to investigate the 1D FDTD formulation. In order to go from a 3D derivation to a 1D derivation, the parts of Maxwell's equations involving change in two dimensions can be ignored. Then is it clear two sets of two equations can be de-coupled from each other. These two pairs of equations then represent the 1D TE or TM modes of plane wave propagation.

2.4.1 Material Updating Equations

For a one dimensional case where X is the only dimension that varies, any derivative with respect to Y or Z are zero. Furthermore, E_x and H_x are set to zero since only plane waves propagate in 1D space and the direction of propagation is in the X-direction. Once the approximations are made to equations 2.135

through 2.140 we are left with four component level equations:

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon_y}\left(-\frac{\partial H_z}{\partial x} - \sigma_y^e E_y - J_{iy}\right), \tag{2.153}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon_z}\left(\frac{\partial H_y}{\partial x} - \sigma_z^e E_z - J_{iz}\right), \tag{2.154}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu_y}\left(\frac{\partial E_z}{\partial x} - \sigma_y^m H_y - M_{iy}\right), \tag{2.155}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(-\frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right). \tag{2.156}$$

By looking at these four equations, it is clear that there are two sets of coupled equations. Equations 2.153 and 2.156 are coupled and equations 2.154 and 2.155 are coupled. These two groups are independent of one another. Equations 2.153 and 2.156 represent a TE_z mode while equations 2.154 and 2.155 represent a TM_z mode. The 1D code analyzed in this thesis models a TM_z mode and thus uses equations 2.154 and 2.155. From here similar steps to section 2.2.1 can be taken to arrive at the 1D FDTD updating equations.

The general 1D E_z updating equation is as follows:

$$\begin{aligned} E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)}\alpha \\ & + \frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k)}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} E_z^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} J_z^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.157}$$

where α is the appropriate numerical derivative for $\frac{\partial H_y}{\partial x}$. The second order E_z updating equation is shown in equation 2.158 and the foruth order updating equation for E_z is shown in equation 2.159.

$$\begin{aligned} E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} \frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)}{\Delta x} \\ & + \frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k)}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} E_z^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} J_z^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.158}$$

$$\begin{aligned} E_z^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} \frac{-H_y^{n+\frac{1}{2}}(i+1,j,k) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i-1,j,k) + H_y^{n+\frac{1}{2}}(i-2,j,k)}{24\Delta x} \\ & + \frac{2\epsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k)}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} E_z^n(i,j,k) \\ & - \frac{2\Delta t}{2\epsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k)} J_z^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.159}$$

The second order H_y updating equation is shown in equation 2.160 and the foruth order updating equation for H_y is shown in equation 2.161.

$$\begin{aligned} H_y^{n+1}(i,j,k) = &\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}\frac{E_z^{n+\frac{1}{2}}(i+1,j,k)-E_z^{n+\frac{1}{2}}(i,j,k)}{\Delta x} \\ &+\frac{2\mu_y(i,j,k)-\Delta t\sigma_y^m(i,j,k)}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}H_y^n(i,j,k) \\ &-\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}M_y^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.160}$$

$$\begin{aligned} H_y^{n+1}(i,j,k) = &\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}\frac{-E_z^{n+\frac{1}{2}}(i+2,j,k)+27E_z^{n+\frac{1}{2}}(i+1,j,k)-27E_z^{n+\frac{1}{2}}(i,j,k)+E_z^{n+\frac{1}{2}}(i-1,j,k)}{24\Delta x} \\ &+\frac{2\mu_y(i,j,k)-\Delta t\sigma_y^m(i,j,k)}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}H_y^n(i,j,k) \\ &-\frac{2\Delta t}{2\mu_y(i,j,k)+\Delta t\sigma_y^m(i,j,k)}M_y^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{2.161}$$

CHAPTER 3
THE ABSORBING BOUNDARY

A good absorbing boundary is paramount to any electromagnetic simulator. In this FDTD code, convolutional perfectly matched layer (CPML) is implemented [25]. Other perfectly matched layer formulations exist [26], but they are not optimized. The CPML absorbing boundary is advantageous because it can effectively absorb evanescent waves. The effective absorption of evanescent waves means fewer air buffer cells are needed between the objects in the simulation domain and the start of the CPML layers. Fewer air buffer cells decrease memory requirements and speed up simulations.

3.1 Fourth Order CPML Mathematical Formulation

The mathematical formulation for fourth order accurate CPML is very similar to the second order accurate CPML formulation presented in [1]. Equation (8.29) of [1] shows the final form of the updating equation for second order accurate CPML:

$$\begin{aligned} E_x^{n+1}(i,j,k) = C_{exe}(i,j,k) \times E_x^n(i,j,k) \\ +C_{exhz}(i,j,k) \times \left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right) \\ +C_{exhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)\right) \\ +C_{\psi exy}(i,j,k) \times \psi_{exy}^{n+\frac{1}{2}}(i,j,k) + C_{\psi exz}(i,j,k) \times \psi_{exz}^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{3.1}$$

where,

$$\psi_{exy}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0ey}(m) \left(H_z^{n-m+\frac{1}{2}}(i,j,k) - H_z^{n-m+\frac{1}{2}}(i,j-1,k)\right), \tag{3.2}$$

$$Z_{0ey}(m) = \frac{\sigma_{pey}}{\Delta y \left(\sigma_{pey}\kappa_{ey} + \alpha_{ey}\kappa_{ey}^2\right)} \left(e^{-\left(\frac{\sigma_{pey}}{\kappa_{ey}}+\alpha_{ey}\right)\frac{\Delta t}{\epsilon_0}} - 1\right) \left(e^{-\left(\frac{\sigma_{pey}}{\kappa_{ey}}+\alpha_{ey}\right)\frac{m\Delta t}{\epsilon_0}}\right), \tag{3.3}$$

$$\psi_{exz}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0ez}(m) \left(H_y^{n-m+\frac{1}{2}}(i,j,k) - H_y^{n-m+\frac{1}{2}}(i,j,k-1)\right), \tag{3.4}$$

$$Z_{0ez}(m) = \frac{\sigma_{pez}}{\Delta z \left(\sigma_{pez}\kappa_{ez} + \alpha_{ez}\kappa_{ez}^2\right)} \left(e^{-\left(\frac{\sigma_{pez}}{\kappa_{ez}}+\alpha_{ez}\right)\frac{\Delta t}{\epsilon_0}} - 1\right) \left(e^{-\left(\frac{\sigma_{pez}}{\kappa_{ez}}+\alpha_{ez}\right)\frac{m\Delta t}{\epsilon_0}}\right), \tag{3.5}$$

$$C_{exe}(i,j,k) = \frac{2\epsilon_x(i,j,k) - \Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)}, \tag{3.6}$$

$$C_{exhz}(i,j,k) = \frac{2\Delta t}{\Delta y \left(2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)\right)}, \tag{3.7}$$

$$C_{exhy}(i,j,k) = \frac{-2\Delta t}{\Delta z \left(2\epsilon_x(i,j,k) + \Delta t\sigma_x^e(i,j,k)\right)}, \tag{3.8}$$

$$C_{\psi exy}(i,j,k) = \frac{2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}, \tag{3.9}$$

$$C_{\psi exz}(i,j,k) = \frac{-2\Delta t}{2\epsilon_x(i,j,k) + \Delta t \sigma_x^e(i,j,k)}. \tag{3.10}$$

Looking at the second and third terms in equation 3.1, it is clear that they match the general second order updating equation for E_x. This implies that the same technique to convert the normal domain updating equations to fourth order can be used here as well.

Looking at the fourth and fifth terms of equation 3.1 and equation 3.2 it is clear that the definition of ψ must alter in order to be fourth order accurate. As shown in equation 3.3 there is a Δy in the denominator of Z_{0ey}. This $1/\Delta y$ combined with the $\left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right)$ term constitutes the second order accurate derivative approximation in full. Therefore, in order to convert the definition of ψ to fourth order, all that is needed is to convert this second order derivative approximation to fourth order.

Since the Δy and Δz values are in the applicable denominators of the coefficients C_{exhz}, C_{exhy}, Z_{0ey}, and Z_{0ez} already, equations 3.1 and 3.2 converted to fourth order accuracy are as follows:

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & C_{exe}(i,j,k) \times E_x^n(i,j,k) \\ & + C_{exhz}(i,j,k) \times \left(\frac{-H_z^{n+\frac{1}{2}}(i,j+1,k) + 27H_z^{n+\frac{1}{2}}(i,j,k) - 27H_z^{n+\frac{1}{2}}(i,j-1,k) + H_z^{n+\frac{1}{2}}(i,j-2,k)}{24} \right) \\ & + C_{exhy}(i,j,k) \times \left(\frac{-H_y^{n+\frac{1}{2}}(i,j,k+1) + 27H_y^{n+\frac{1}{2}}(i,j,k) - 27H_y^{n+\frac{1}{2}}(i,j,k-1) + H_y^{n+\frac{1}{2}}(i,j,k-2)}{24} \right) \\ & + C_{\psi exy}(i,j,k) \times \psi_{exy}^{n+\frac{1}{2}}(i,j,k) + C_{\psi exz}(i,j,k) \times \psi_{exz}^{n+\frac{1}{2}}(i,j,k), \end{aligned} \tag{3.11}$$

and,

$$\begin{aligned} \psi_{exy}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0ey}(m) \\ & * \left(\frac{-H_z^{n-m+\frac{1}{2}}(i,j+1,k) + 27H_z^{n-m+\frac{1}{2}}(i,j,k) - 27H_z^{n-m+\frac{1}{2}}(i,j-1,k) + H_z^{n-m+\frac{1}{2}}(i,j-2,k)}{24} \right), \end{aligned} \tag{3.12}$$

$$\begin{aligned} \psi_{exz}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0ez}(m) \\ & * \left(\frac{-H_y^{n-m+\frac{1}{2}}(i,j,k+1) + 27H_y^{n-m+\frac{1}{2}}(i,j,k) - 27H_y^{n-m+\frac{1}{2}}(i,j,k-1) + H_y^{n-m+\frac{1}{2}}(i,j,k-2)}{24} \right). \end{aligned} \tag{3.13}$$

The rest of the electric and magnetic field components can be upated with fourth order in a similar way. Equations 3.14 through 3.28 summarize. As the fourth order coefficients are the same as the second order coefficients, the coefficients for these equations are defined fully in [1].

$$\begin{aligned}E_y^{n+1}(i,j,k) = {} & C_{eye}(i,j,k)\times E_y^{n}(i,j,k)\\ & +C_{eyhx}(i,j,k)\times\left(\frac{-H_x^{n+\frac{1}{2}}(i,j,k+1)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j,k-1)+H_x^{n+\frac{1}{2}}(i,j,k-2)}{24}\right)\\ & +C_{eyhz}(i,j,k)\times\left(\frac{-H_z^{n+\frac{1}{2}}(i+1,j,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i-1,j,k)+H_z^{n+\frac{1}{2}}(i-2,j,k)}{24}\right)\\ & +C_{\psi eyz}(i,j,k)\times\psi_{eyz}^{n+\frac{1}{2}}(i,j,k)+C_{\psi eyx}(i,j,k)\times\psi_{eyx}^{n+\frac{1}{2}}(i,j,k),\end{aligned} \tag{3.14}$$

and,

$$\begin{aligned}\psi_{eyz}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0ez}(m)\\ & *\left(\frac{-H_x^{n-m+\frac{1}{2}}(i,j,k+1)+27H_x^{n-m+\frac{1}{2}}(i,j,k)-27H_x^{n-m+\frac{1}{2}}(i,j,k-1)+H_x^{n-m+\frac{1}{2}}(i,j,k-2)}{24}\right),\end{aligned} \tag{3.15}$$

$$\begin{aligned}\psi_{eyx}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0ex}(m)\\ & *\left(\frac{-H_z^{n-m+\frac{1}{2}}(i+1,j,k)+27H_z^{n-m+\frac{1}{2}}(i,j,k)-27H_z^{n-m+\frac{1}{2}}(i-1,j,k)+H_z-m^{n+\frac{1}{2}}(i-2,j,k)}{24}\right).\end{aligned} \tag{3.16}$$

$$\begin{aligned}E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k)\times E_z^{n}(i,j,k)\\ & +C_{ezhy}(i,j,k)\times\left(\frac{-H_y^{n+\frac{1}{2}}(i+1,j,k)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i-1,j,k)+H_y^{n+\frac{1}{2}}(i-2,j,k)}{24}\right)\\ & +C_{ezhx}(i,j,k)\times\left(\frac{-H_x^{n+\frac{1}{2}}(i,j+1,k)+27H_x^{n+\frac{1}{2}}(i,j,k)-27H_x^{n+\frac{1}{2}}(i,j-1,k)+H_x^{n+\frac{1}{2}}(i,j-2,k)}{24}\right)\\ & +C_{\psi ezx}(i,j,k)\times\psi_{ezx}^{n+\frac{1}{2}}(i,j,k)+C_{\psi ezy}(i,j,k)\times\psi_{ezy}^{n+\frac{1}{2}}(i,j,k),\end{aligned} \tag{3.17}$$

and,

$$\psi_{ezx}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0ex}(m) * \left(\frac{-H_y^{n-m+\frac{1}{2}}(i+1,j,k) + 27H_y^{n-m+\frac{1}{2}}(i,j,k) - 27H_y^{n-m+\frac{1}{2}}(i-1,j,k) + H_y^{n-m+\frac{1}{2}}(i-2,j,k)}{24} \right), \tag{3.18}$$

$$\psi_{ezy}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0ey}(m) * \left(\frac{-H_x^{n-m+\frac{1}{2}}(i,j+1,k) + 27H_x^{n-m+\frac{1}{2}}(i,j,k) - 27H_x^{n-m+\frac{1}{2}}(i,j-1,k) + H_x^{n-m+\frac{1}{2}}(i,j-2,k)}{24} \right). \tag{3.19}$$

$$\begin{aligned} H_x^{n+1}(i,j,k) = {} & C_{hxe}(i,j,k) \times H_x^{n}(i,j,k) \\ & + C_{hxey}(i,j,k) \times \left(\frac{-E_y^{n+\frac{1}{2}}(i,j,k+2) + 27E_y^{n+\frac{1}{2}}(i,j,k+1) - 27E_y^{n+\frac{1}{2}}(i,j,k) + E_y^{n+\frac{1}{2}}(i,j,k-1)}{24} \right) \\ & + C_{hxez}(i,j,k) \times \left(\frac{-E_z^{n+\frac{1}{2}}(i,j+2,k) + 27E_z^{n+\frac{1}{2}}(i,j+1,k) - 27E_z^{n+\frac{1}{2}}(i,j,k) + E_z^{n+\frac{1}{2}}(i,j-1,k)}{24} \right) \\ & + C_{\psi hxz}(i,j,k) \times \psi_{hxz}^{n+\frac{1}{2}}(i,j,k) + C_{\psi hxy}(i,j,k) \times \psi_{hxy}^{n+\frac{1}{2}}(i,j,k), \end{aligned} \tag{3.20}$$

and,

$$\psi_{hxz}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0hz}(m) * \left(\frac{-E_y^{n-m+\frac{1}{2}}(i,j,k+2) + 27E_y^{n-m+\frac{1}{2}}(i,j,k+1) - 27E_y^{n-m+\frac{1}{2}}(i,j,k) + E_y^{n-m+\frac{1}{2}}(i,j,k-1)}{24} \right), \tag{3.21}$$

$$\psi_{hxy}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0hy}(m) * \left(\frac{-E_z^{n-m+\frac{1}{2}}(i,j+2,k) + 27E_z^{n-m+\frac{1}{2}}(i,j+1,k) - 27E_z^{n-m+\frac{1}{2}}(i,j,k) + E_z^{n-m+\frac{1}{2}}(i,j-1,k)}{24} \right). \tag{3.22}$$

$$
\begin{aligned}
H_y^{n+1}(i,j,k) = {} & C_{hye}(i,j,k) \times H_y^n(i,j,k) \\
& + C_{hyez}(i,j,k) \times \left(\frac{-E_z^{n+\frac{1}{2}}(i+2,j,k) + 27E_z^{n+\frac{1}{2}}(i+1,j,k) - 27E_z^{n+\frac{1}{2}}(i,j,k) + E_z^{n+\frac{1}{2}}(i-1,j,k)}{24} \right) \\
& + C_{hyex}(i,j,k) \times \left(\frac{-E_x^{n+\frac{1}{2}}(i,j,k+2) + 27E_x^{n+\frac{1}{2}}(i,j,k+1) - 27E_x^{n+\frac{1}{2}}(i,j,k) + E_x^{n+\frac{1}{2}}(i,j,k-1)}{24} \right) \\
& + C_{\psi hyx}(i,j,k) \times \psi_{hyx}^{n+\frac{1}{2}}(i,j,k) + C_{\psi hyz}(i,j,k) \times \psi_{hyz}^{n+\frac{1}{2}}(i,j,k),
\end{aligned}
\tag{3.23}
$$

and,

$$
\begin{aligned}
\psi_{hyx}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0hx}(m) \\
& * \left(\frac{-E_z^{n-m+\frac{1}{2}}(i+2,j,k) + 27E_z^{n-m+\frac{1}{2}}(i+1,j,k) - 27E_z^{n-m+\frac{1}{2}}(i,j,k) + E_z^{n-m+\frac{1}{2}}(i-1,j,k)}{24} \right),
\end{aligned}
\tag{3.24}
$$

$$
\begin{aligned}
\psi_{hyz}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0hz}(m) \\
& * \left(\frac{-E_x^{n-m+\frac{1}{2}}(i,j,k+2) + 27E_x^{n-m+\frac{1}{2}}(i,j,k+1) - 27E_x^{n-m+\frac{1}{2}}(i,j,k) + E_x^{n-m+\frac{1}{2}}(i,j,k-1)}{24} \right).
\end{aligned}
\tag{3.25}
$$

$$
\begin{aligned}
H_z^{n+1}(i,j,k) = {} & C_{hze}(i,j,k) \times H_z^n(i,j,k) \\
& + C_{hzex}(i,j,k) \times \left(\frac{-E_x^{n+\frac{1}{2}}(i,j+2,k) + 27E_x^{n+\frac{1}{2}}(i,j+1,k) - 27E_x^{n+\frac{1}{2}}(i,j,k) + E_x^{n+\frac{1}{2}}(i,j-1,k)}{24} \right) \\
& + C_{hzey}(i,j,k) \times \left(\frac{-E_y^{n+\frac{1}{2}}(i+2,j,k) + 27E_y^{n+\frac{1}{2}}(i+1,j,k) - 27E_y^{n+\frac{1}{2}}(i,j,k) + E_y^{n+\frac{1}{2}}(i-1,j,k)}{24} \right) \\
& + C_{\psi hzy}(i,j,k) \times \psi_{hzy}^{n+\frac{1}{2}}(i,j,k) + C_{\psi hzx}(i,j,k) \times \psi_{hzx}^{n+\frac{1}{2}}(i,j,k),
\end{aligned}
\tag{3.26}
$$

and,

$$
\begin{aligned}
\psi_{hzy}^{n+\frac{1}{2}}(i,j,k) = {} & \sum_{m=0}^{m=n-1} Z_{0hy}(m) \\
& * \left(\frac{-E_x^{n-m+\frac{1}{2}}(i,j+2,k) + 27E_x^{n-m+\frac{1}{2}}(i,j+1,k) - 27E_x^{n-m+\frac{1}{2}}(i,j,k) + E_x^{n-m+\frac{1}{2}}(i,j-1,k)}{24} \right),
\end{aligned}
\tag{3.27}
$$

$$\psi_{hzx}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0hx}(m) * \left(\frac{-E_y^{n-m+\frac{1}{2}}(i+2,j,k) + 27E_y^{n-m+\frac{1}{2}}(i+1,j,k) - 27E_y^{n-m+\frac{1}{2}}(i,j,k) + E_y^{n-m+\frac{1}{2}}(i-1,j,k)}{24} \right). \tag{3.28}$$

Equations 3.11 through 3.28 explain how to update the fourth order CPML, but they do not tell the full story. The CPML presented in [1] has three optimization parameters that can be tuned to ensure optimum boundary absorption. These parameters are sigma factor (σ_f), kappa max (κ_{max}), and alpha max (α_{max}), and they are re-optimized for the fourth order FDTD code in section 4.4.

3.2 Fourth Order CPML MATLAB Implementation

Implementing fourth order accurate CPML into the Matlab code had some difficulties largely due to the need for two layers of PEC boundary terminating the domain. Two layers of PEC are needed to terminate the domain of a fourth order simulation because the fourth order mask is bigger than the second order mask. Two field components beside the component being updated are needed. At the edges, say one cell from the domain boundary, it is not possible to get two components to update the field component. The first component that can be updated with fourth order accuracy is two cells away from the boundary edge, hence the need for two layers of PEC. The same technique is used to terminate the outer limit of the PML region, of a higher order FDTD formulation, in a work done by Wu and Kuo [27].

A fourth order simulation terminated by PEC layers without any CPML is relatively simple to implement. As long as the updating equations are ensured to not update the two layers at the boundary, the simulation will run. When CPML is involved, the CPML layers must also be shifted so as to have the appropriate matching characteristics. Figure 3.1 shows a diagram of the FDTD domain with CPML and two layers of PEC boundary termination.

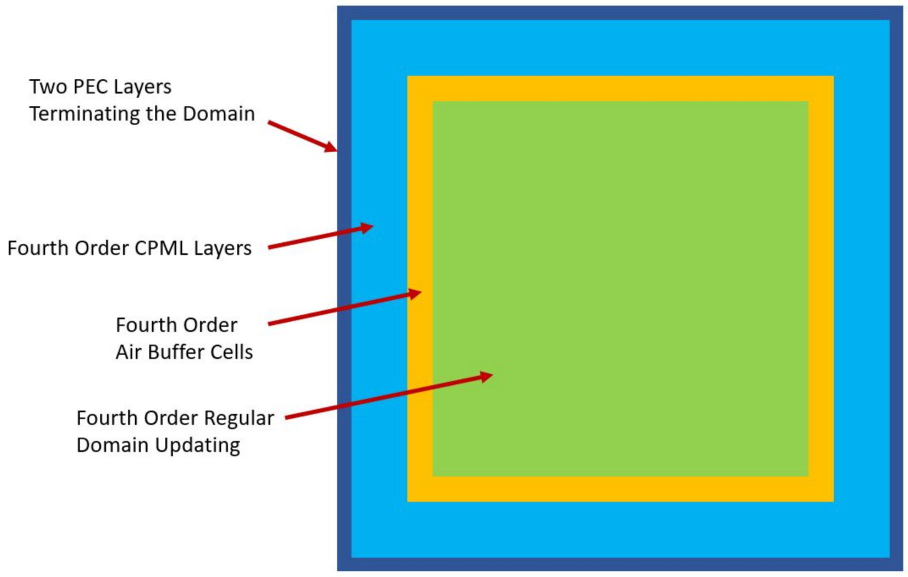


Figure 3.1 A diagram of the FDTD computational domain with CPML layers.

The most elegant way to shift the CPML layers to account for the two layers of PEC is to shift the field components the CPML coefficients and ψ value applies to. Listing Listing 3.1 shows this shift. As an example, line 5 indexes the H_z field component rather redundantly as:
$(i+2+1,:,:), (i+1+1,:,:), (i+1,:,:), (i-1+1,:,:)$ the extra "+1" in the first position of the array is the shift accommodating for the second layer of PEC on the boundary of the domain. Since this is the lower X CPML layers of the domain, the "+1" ensures the fourth order mask does not try to index H_z with a "0" or negative index. At the upper X-direction CPML layers (line 19 in the code for example), there is a "-1" in the H_z indexing creating a second layer of PEC and ensuring the fourth order mask does not try to index H_z with a value larger than the array size for H_z. Similar $+1$ or -1 shifts are applied to the upper and lower Y direction and Z direction CPML layers.

It should also be noted that the 1/24 in equation 3.12 has been tied into the definition of CPML_a_ex_xn. The first three terms of equation 3.11 are handled by the regular "update electric fields" code and wrapped into simply E_x in code listing Listing 3.1.

Listing 3.1: update electric field CPML ABC

```
3.1 % apply CPML to electric field components
3.2 if is_cpml_xn
3.3     for i = 1:n_cpml_xn
3.4         Psi_eyx_xn(i,:,:) = cpml_b_ex_xn(i) * Psi_eyx_xn(i,:,:) ...
3.5             + cpml_a_ex_xn(i)*(-Hz(i+2+1,:,:)+27*Hz(i+1+1,:,:)-27*Hz(i+1,:,:)+Hz(i
                    -1+1,:,:));
3.6         Psi_ezx_xn(i,:,:) = cpml_b_ex_xn(i) * Psi_ezx_xn(i,:,:) ...
```

```
            + cpml_a_ex_xn(i)*(-Hy(i+2+1,:,:)+27*Hy(i+1+1,:,:)-27*Hy(i+1,:,:)+Hy(i
                -1+1,:,:));
    end
    Ey(2+1:n_cpml_xn+1+1,:,:) = Ey(2+1:n_cpml_xn+1+1,:,:) ...
            + CPsi_eyx_xn .* Psi_eyx_xn;
    Ez(2+1:n_cpml_xn+1+1,:,:) = Ez(2+1:n_cpml_xn+1+1,:,:) ...
            + CPsi_ezx_xn .* Psi_ezx_xn;
end

if is_cpml_xp
    n_st = nx - n_cpml_xp;
    for i = 1:n_cpml_xp
        Psi_eyx_xp(i,:,:) = cpml_b_ex_xp(i) * Psi_eyx_xp(i,:,:) ...
            + cpml_a_ex_xp(i)*(-Hz(i+n_st+1-1,:,:)+27*Hz(i+n_st-1,:,:)-27*Hz(i+n_st
                -1-1,:,:)+Hz(i+n_st-2-1,:,:));
        Psi_ezx_xp(i,:,:) = cpml_b_ex_xp(i) * Psi_ezx_xp(i,:,:) ...
            + cpml_a_ex_xp(i)*(-Hy(i+n_st+1-1,:,:)+27*Hy(i+n_st-1,:,:)-27*Hy(i+n_st
                -1-1,:,:)+Hy(i+n_st-2-1,:,:));
    end

    Ey(n_st+1-1:nx-1,:,:) = Ey(n_st+1-1:nx-1,:,:) ...
        + CPsi_eyx_xp .* Psi_eyx_xp;
    Ez(n_st+1-1:nx-1,:,:) = Ez(n_st+1-1:nx-1,:,:) ...
        + CPsi_ezx_xp .* Psi_ezx_xp;

end

if is_cpml_yn
    for i = 1:n_cpml_yn
        Psi_ezy_yn(:,i,:) = cpml_b_ey_yn(i) * Psi_ezy_yn(:,i,:) ...
            + cpml_a_ey_yn(i)*(-Hx(:,i+2+1,:)+27*Hx(:,i+1+1,:)-27*Hx(:,i+1,:)+Hx(:,i
                -1+1,:));
        Psi_exy_yn(:,i,:) = cpml_b_ey_yn(i) * Psi_exy_yn(:,i,:) ...
            + cpml_a_ey_yn(i)*(-Hz(:,i+2+1,:)+27*Hz(:,i+1+1,:)-27*Hz(:,i+1,:)+Hz(:,i
                -1+1,:));
    end
    Ez(:,2+1:n_cpml_yn+1+1,:) = Ez(:,2+1:n_cpml_yn+1+1,:) ...
        + CPsi_ezy_yn .* Psi_ezy_yn;
    Ex(:,2+1:n_cpml_yn+1+1,:) = Ex(:,2+1:n_cpml_yn+1+1,:) ...
        + CPsi_exy_yn .* Psi_exy_yn;
end

if is_cpml_yp
    n_st = ny - n_cpml_yp;
    for i = 1:n_cpml_yp
        Psi_ezy_yp(:,i,:) = cpml_b_ey_yp(i) * Psi_ezy_yp(:,i,:) ...
            + cpml_a_ey_yp(i)*(-Hx(:,i+n_st+1-1,:)+27*Hx(:,i+n_st-1,:)-27*Hx(:,i+
                n_st-1-1,:)+Hx(:,i+n_st-2-1,:));
        Psi_exy_yp(:,i,:) = cpml_b_ey_yp(i) * Psi_exy_yp(:,i,:) ...
            + cpml_a_ey_yp(i)*(-Hz(:,i+n_st+1-1,:)+27*Hz(:,i+n_st-1,:)-27*Hz(:,i+
                n_st-1-1,:)+Hz(:,i+n_st-2-1,:));
    end

    Ez(:,n_st+1-1:ny-1,:) = Ez(:,n_st+1-1:ny-1,:) ...
        + CPsi_ezy_yp .* Psi_ezy_yp;
    Ex(:,n_st+1-1:ny-1,:) = Ex(:,n_st+1-1:ny-1,:) ...
        + CPsi_exy_yp .* Psi_exy_yp;
end

```

```
if is_cpml_zn
    for i = 1:n_cpml_zn
        Psi_exz_zn(:,:,i) = cpml_b_ez_zn(i) * Psi_exz_zn(:,:,i) ...
            + cpml_a_ez_zn(i)*(-Hy(:,:,i+2+1)+27*Hy(:,:,i+1+1)-27*Hy(:,:,i+1)+Hy
                (:,:,i-1+1));
        Psi_eyz_zn(:,:,i) = cpml_b_ez_zn(i) * Psi_eyz_zn(:,:,i) ...
            + cpml_a_ez_zn(i)*(-Hx(:,:,i+2+1)+27*Hx(:,:,i+1+1)-27*Hx(:,:,i+1)+Hx
                (:,:,i-1+1));
    end
    Ex(:,:,2+1:n_cpml_zn+1+1) = Ex(:,:,2+1:n_cpml_zn+1+1) ...
        + CPsi_exz_zn .* Psi_exz_zn;
    Ey(:,:,2+1:n_cpml_zn+1+1) = Ey(:,:,2+1:n_cpml_zn+1+1) ...
        + CPsi_eyz_zn .* Psi_eyz_zn;
end

if is_cpml_zp
    n_st = nz - n_cpml_zp;
    for i = 1:n_cpml_zp
        Psi_exz_zp(:,:,i) = cpml_b_ez_zp(i) * Psi_exz_zp(:,:,i) ...
            + cpml_a_ez_zp(i)*(-Hy(:,:,i+n_st+1-1)+27*Hy(:,:,i+n_st-1)-27*Hy(:,:,i+
                n_st-1-1)+Hy(:,:,i+n_st-2-1));
        Psi_eyz_zp(:,:,i) = cpml_b_ez_zp(i) * Psi_eyz_zp(:,:,i) ...
            + cpml_a_ez_zp(i)*(-Hx(:,:,i+n_st+1-1)+27*Hx(:,:,i+n_st-1)-27*Hx(:,:,i+
                n_st-1-1)+Hx(:,:,i+n_st-2-1));
    end

    Ex(:,:,n_st+1-1:nz-1) = Ex(:,:,n_st+1-1:nz-1) ...
        + CPsi_exz_zp .* Psi_exz_zp;
    Ey(:,:,n_st+1-1:nz-1) = Ey(:,:,n_st+1-1:nz-1) ...
        + CPsi_eyz_zp .* Psi_eyz_zp;
end
```

CHAPTER 4
VERIFICATION EXAMPLES

The goal of this chapter is to show multiple examples to test the different aspects of the fourth order formulation presented in chapters 2 and 3.

4.1 Gaussian Propagation

In this section, a Gaussian pulse is generated and allowed to propagate in free space. Simulations with second order updating equations and simulations with fourth order updating equations are compared. The goal of such a comparison is to show that the basic fourth order updating equations are producing correct results and are more accurate than the second order ones. In this section both 1D and 3D second and fourth order simulations are compared. Similar analysis has been done in [2].

The simulation setup for the 1D and 3D and second and fourth order simulations are almost identical. The domain size is different between the 1D and 3D simulations, but that is the only difference between the four simulation setups. The 1D domain is one meter long, discretized by 300 cells, and filled completely with air. The 3D domain is one cubic meter, discretized by 300 cells in each direction, and filled completely with air. The boundaries of the domain are terminated with PEC, but the number of time steps is such as to not allow the Gaussian pulse to propagate to the boundary. For the 3D simulations, the source of the Gaussian pulse is an infinite sheet of Jz source located in the YZ plane. For the 1D simulation the source of the Gaussian pulse is a single point of Jz source. The width of the pulse is set to allow the minimum wavelength (λ_{min}) in the frequency domain to only be discretized by 5 cells. For second order simulations, it is recommended that 20 cells be used to discretize the smallest wavelength of interest to avoid numerical errors [1]. The total simulation time is ≈1.18 ns which is achieved through 213 time steps in 1D and 370 time steps in the 3D simulation. The Δt of the simulations is the maximum Δt based on the stability criterion that is dependent on the cell size and then multiplied by a Courant factor of 0.5.

4.1.1 Courant Factor

The maximum stable time step of the fourth order simulation is $\frac{6}{7}$ the maximum stable time step of a second order simulation for the same domain as detailed in section 2.2.3.3. As the Courant factor is defined as a coefficient on the maximum stable time step, second order and fourth order simulations with the same courant factor on the same domain will still have a different Δt value. Specifically, for any given Courant factor with a given domain, the Δt of the fourth order simulation will be $\frac{6}{7}$ times less than the Δt of the second order simulation. The goal of this section is to explore the effects of Courant factor on solution accuracy (namely dispersion) for the 3D second and forth order codes. These effects are closely studied in [24], and here a similar relationship for the fourth order scheme is presented.

The dispersion error in this section is calculated from the 3D Gaussian propagation simulations presented in section 4.1.2. The Courant factor is varied over a number of simulations, and for each simulation, approximately 1.18 ns of time is simulated. In order to keep the total time of the simulation constant while varying the courant factor, the number of time steps is different in each simulation. At the final time step, the minimum of the z component of the electric field is recorded as a measure of the dispersion. If perfect Gaussian propagation occurred with no dispersion error, this value would be zero as the ideal Gaussian waveform is positive over the entire simulation time. However, section 4.1.2 shows that the simulated Gaussian propagation is not perfect and will in fact go below zero. The more it goes below zero the greater the dispersion error. Figure 4.1 shows an example of how the dispersion error is calculated for a second order simulation with a Courant factor of 0.5. Note that 0.5 is not the optimum Courant factor for a 3D second order FDTD simulation, so the dispersion error in Figure 4.1 is larger than the minimum dispersion error of a second order FDTD simulation. Figure 4.1 serves to explain how dispersion error is calculated in this section, it is not meant to provide results to comment on the accuracy of the second order code, so a non-optimum Courant factor is reasonable in this case.

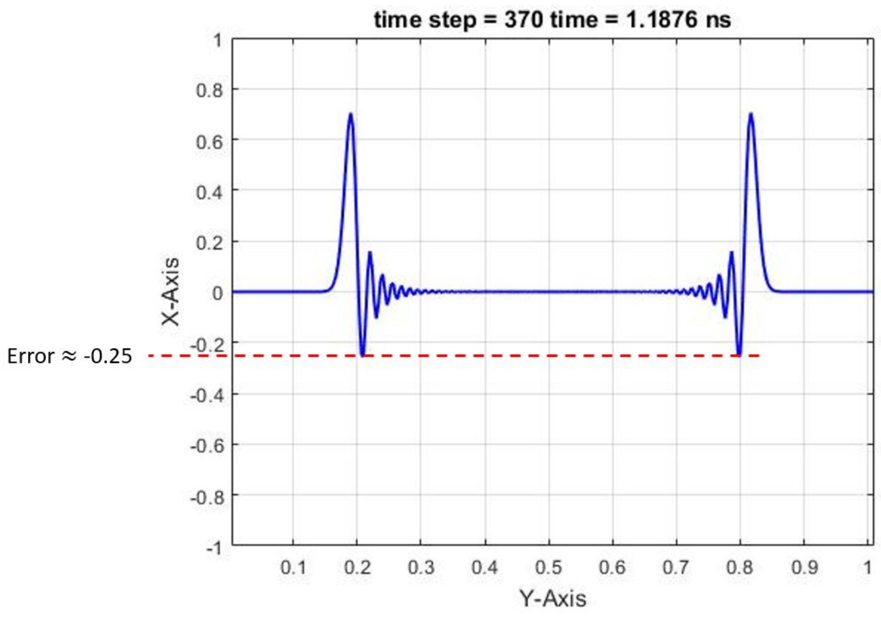
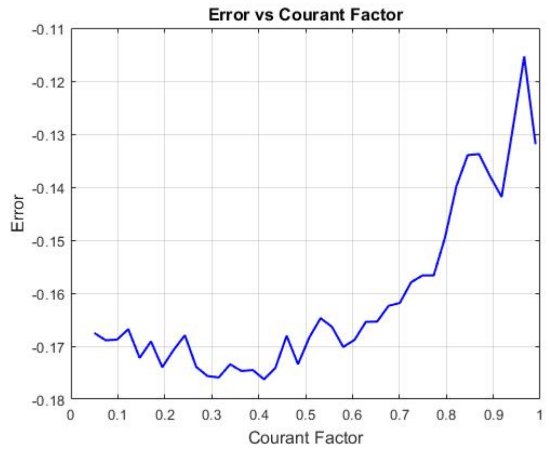
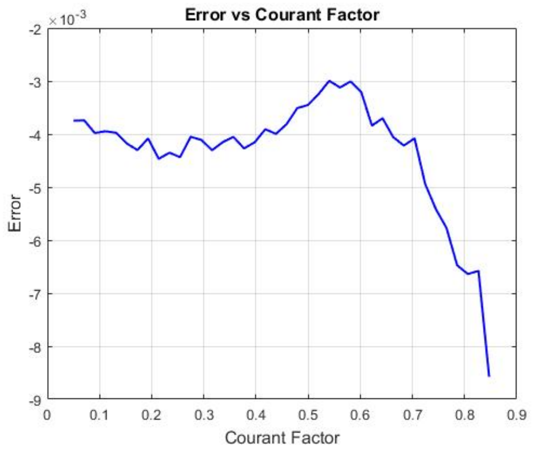


Figure 4.1 Example dispersion error calculation.

Figure 4.2 shows dispersion error vs Courant factor for both the second order and fourth order simulations.



(a)



(b)

Figure 4.2 Second and fourth order measured dispersion as a function of Courant factor.

Figure 4.2(a) are the second order results and Figure 4.2(b) are the fourth order results.

Firstly, note that the magnitude of the error in the fourth order simulation is roughly 30-40 times smaller than the error in the second order simulation. This is more evidence that the fourth order updating equations are working correctly and are more accurate than the second order updating equations.

Secondly, the different shapes of the dispersion error vs Courant factor curves are significant. The second order curve shows the minimum dispersion error is at a Courant factor of roughly 0.95. The minimum dispersion error for the fourth order simulation is at roughly 0.55. As most second order codes aim at using a Courant factor close to 0.95 [1], this information is important since the ultimate goal of using the fourth order code is to improve accuracy. The disadvantage of course is that using the smaller Courant factor of 0.55 will reduce the value of a time step and more time steps will be required to reach a desired simulation time.

4.1.2 3D Gaussian Propagation

In this section a Gaussian pulse is generated and allowed to propagate in free space. The dispersion error between the second order and fourth order simulations is compared. The Gaussian pulse is rather narrow, producing a frequency response with a maximum usable frequency with a wavelength that only covers five cells. For second order, generally 20 cells per wavelength is expected to obtain accurate results [1]. Since the cell size is effectively four times larger (at 5 cells per wavelength), it is expected we see some dispersion in the second order results. Since the fourth order formulation is expected to be more accurate, we expect to see less dispersion in the fourth order results than the second order results. All simulations in this section use the same cell size.

Figure 4.3 shows the simulation domain for the 3D free space Gaussian propagation simulation. Note that the optimum Courant factor is used for each simulation (0.95 for second order and 0.55 for fourth order as shown in Figure 4.2) but the total simulation time is held constant by using more time steps in the fourth order simulation.

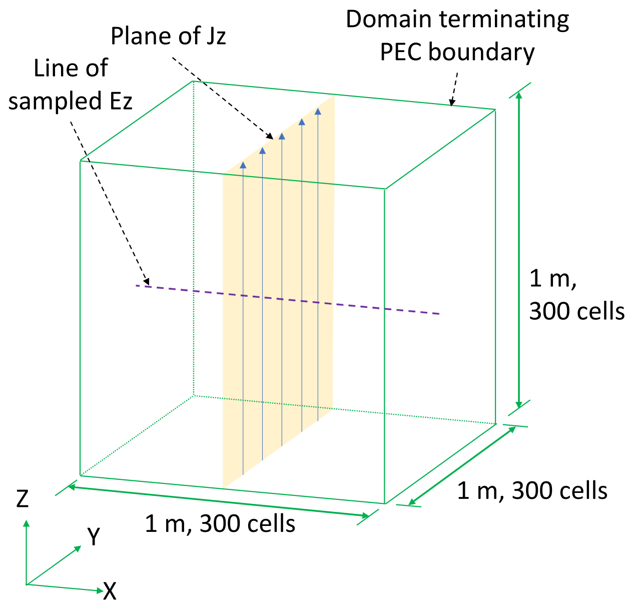
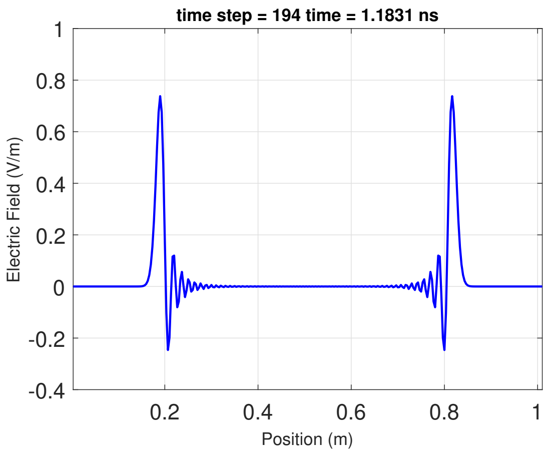
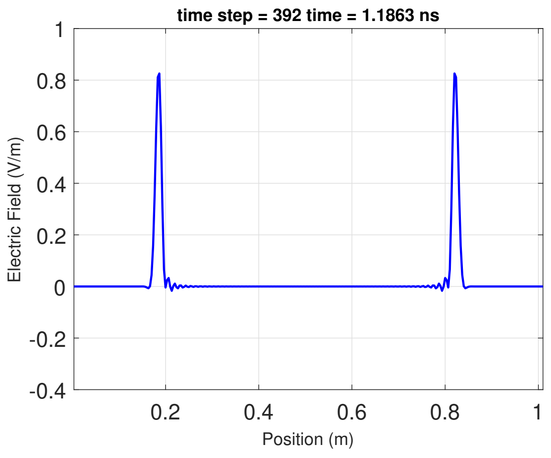


Figure 4.3 The simulation geometry used to simulated the 3D Gaussian propagation problem.

The E_z component of the electric field is sampled on a line parallel to the X-axis as shown in Figure 4.3. Figure 4.4 shows these sampled electric fields in the 3D free space Gaussian propagation simulation using the second order and the fourth order updating equations at 1.1876 ns.



(a)



(b)

Figure 4.4 Second and forth order results of the 3D Gaussian propogation simulation.

Figure 4.4(a) are the second order results for the E_z component of the electric field and Figure 4.4(b) are the fourth order results for the E_z component of the electric field.

As shown by Figure 4.4, the fourth order results have less dispersion than the second order results. This is evidence that the fourth order updating equations are correct and can maintain more accuracy at larger cell sizes than the second order updating equations. Section 5.1.3 elaborates on the specific computational advantages of using fourth order FDTD. Note this simulation does not allow the Gaussian propagation to reach the boundaries of the domain. These results only show evidence that the fourth order updating equations are correct and more accurate than the second order ones. It does not evaluate the fourth order boundaries performance. A study of fourth order domain terminating boundaries are explored in detail in sections 4.3, 4.4, and 4.5.

4.2 Second Order and Fourth Order Cavity Resonator

The cavity resonator problem is a simple problem to simulate. The domain consists of empty space bounded by a reflecting boundary, which in this case is a PEC. This simple domain makes it easy to validate the basic fourth order formulation is working in free space. The validation is done by computing the analytical solution and comparing the simulations of the cavity resonances with the second and fourth order formulations.

4.2.1 Analytical Governing Equation

The analytical solution for the resonant modes of a 3D rectangular cavity resonator is given in [28] as.

$$f = \frac{1}{2\pi\sqrt{\mu\epsilon}}\sqrt{\left(\frac{m\pi}{h}\right)^2 + \left(\frac{n\pi}{L}\right)^2 + \left(\frac{p\pi}{W}\right)^2}, \tag{4.1}$$

where h is the height, L is the length and W is the width of the cavity. m and n are integers equal to 0, 1, 2, 3... and p is an integer equal to 1, 2, 3....

4.2.2 Simulation Set Up

For the simulations done in the following sections, the cavity is divided into 14x14x14 cells the x, y, and z directions and filled with air. The cell size is 1/300 of a meter making the total size of the cavity 4.67 cm by 4.67 cm by 4.67 cm. The cavity walls are made of perfect electric conductors. Figure 4.5 shows this domain.

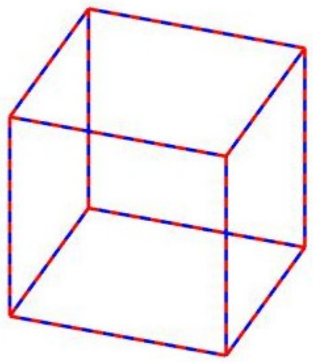


Figure 4.5 The problem space showing the cavity resonator simulation domain.

Both the second and fourth order simulations are run for 100,000 time steps with a Courant factor of 0.5. Many time steps are used such that the generated signal from the source can bounce back and forth from the walls many times. This bouncing wave is sampled in the time domain, and then a frequency domain response can be obtained.

The source is one z-directed current density component located at x = 4, y = 4, z = 4 in the domain. At time step one, the source is set equal to 10, and at all other time steps the source is set equal to zero. The sample point is an E_z field component located in the middle of the first octant of the 3D cube that is the domain. At every time step, a sample value is taken from this point and stored into an array.

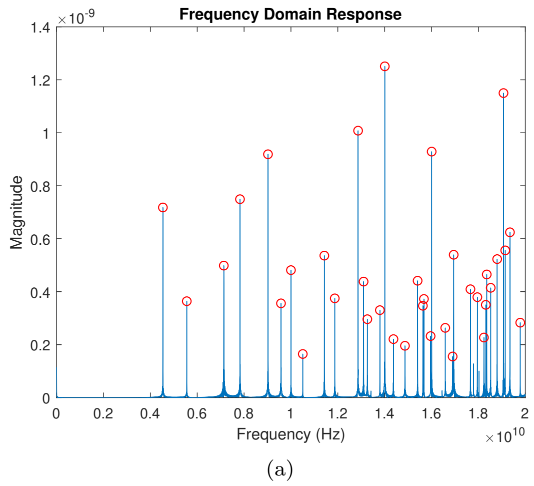
After the simulation is completed, the array containing all the electric field samples over all time steps is converted to the frequency domain with a Fourier transform. Figure 4.6 shows the results of the Fourier transforms for the second and fourth order simulations. After the Fourier transform, a peak finding algorithm (see Listing 4.1 for details) finds the peaks in the frequency spectrum. The detected peaks are shown in Figure 4.6 as red circles. Once these peaks are detected, another algorithm matches them to the analytical mode with the closest frequency to the peak. The expected mode frequencies for the first 1000 modes (m, n = 0, 1, 2,...,10 and p = 1, 2, 3,...,10) are calculated using equation 4.1 and then compared to each peak. Due to the fact that the modes and data peaks become closer in frequency, and thus have a higher chance of matching error as frequency increases, only the first ten peaks are analyzed. Table 4.1 shows the details of the obtained resonances and the errors due to second and fourth order simulations.

Listing 4.1: Calculate Cavity Resonances Code

```
figure(99)
plot(EzSample1)
fArray = 1:5000000:10000*2000000;
PlotFrequenciesOfEzSample1 = zeros(length(fArray),1);
frequenciesOfEzSample1 = timeToFrequency(EzSample1,dt,fArray,0);
for p = 1:length(fArray)
    PlotFrequenciesOfEzSample1(p) = p*dt;
end

counter = 1;
peaks2 = [];
locs2 = [];
for i = 3:length(frequenciesOfEzSample1)-2
    cond1 = abs(frequenciesOfEzSample1(i)) > abs(frequenciesOfEzSample1(i+1));
    cond2 = abs(frequenciesOfEzSample1(i)) > abs(frequenciesOfEzSample1(i-1));
    cond3 = abs(frequenciesOfEzSample1(i)) > 1.5*10^-(10);
    %cond4 = abs(frequenciesOfEzSample1(i+1)) > abs(frequenciesOfEzSample1(i+2));
    %cond5 = abs(frequenciesOfEzSample1(i-1)) > abs(frequenciesOfEzSample1(i-2));
    if cond1 && cond2 && cond3
        locs2(counter) = fArray(i);
        peaks2(counter) = abs(frequenciesOfEzSample1(i));
        counter = counter + 1;
    end
end

figure(100)
plot(fArray,abs(frequenciesOfEzSample1),locs2,peaks2,'rO')
title('Frequency Domain Response')
xlabel('Frequency (Hz)')
ylabel('Magnitude')

%Figure out resonant frequencies from textbook equation, then match to
%peaks data.
temp = 100000000000000;
peak_matches = zeros(counter,4);
temp2 = zeros(3,1);
for i = 1:counter-1
    for m = 0:10
        for n = 0:10
            for p = 0:10
                freq1 = (1/(2*pi*sqrt(mu_0*eps_0)))*sqrt((m*pi/(nx*dx))^2+(n*pi/(ny*dy))^2+(p*pi/(nz*dz))^2);
                if abs(locs2(i)-freq1) < temp
```

```
                    temp = abs(locs2(i)-freq1);
                    temp2(1) = m;
                    temp2(2) = n;
                    temp2(3) = p;
                end
            end
        end
    end
    peak_matches(i,1) = locs2(i);
    peak_matches(i,2) = temp2(1);
    peak_matches(i,3) = temp2(2);
    peak_matches(i,4) = temp2(3);
    temp = 100000000000000;
end

test = 1;
peak_f = peak_matches(test,1);
m = peak_matches(test,2);
n = peak_matches(test,3);
p = peak_matches(test,4);

freq_calc = (1/(2*pi*sqrt(mu_0*eps_0)))*sqrt((m*pi/(nx*dx))^2+(n*pi/(ny*dy))^2+(p*pi/(nz*dz))^2);

outputData = zeros(counter,7);
for i = 1:counter
    %temp_str = ["(", num2str(peak_matches(i,2)), ",", num2str(peak_matches(i,3)), ",", num2str(peak_matches(i,4)), ")"];
    m = peak_matches(i,2);
    n = peak_matches(i,3);
    p = peak_matches(i,4);

    freq_calc = (1/(2*pi*sqrt(mu_0*eps_0)))*sqrt((m*pi/(nx*dx))^2+(n*pi/(ny*dy))^2+(p*pi/(nz*dz))^2);
    outputData(i,1) = peak_matches(i,2);
    outputData(i,2) = peak_matches(i,3);
    outputData(i,3) = peak_matches(i,4);
    outputData(i,4) = freq_calc;
    outputData(i,5) = peak_matches(i,1);
    outputData(i,6) = 100*abs(freq_calc-peak_matches(i,1))/freq_calc;
    outputData(i,7) = (c/freq_calc)/dx;
end

excelFile = 'C:\Users\adesp\Documents\SchoolWork\Thesis\Deliverable Code and Results\Chapter 4 Verification Examples\Section 4.2 Cavity Resonator\S22_PEC_Cavity_Resonator_Results\S22_test.csv';
csvwrite(excelFile,outputData)
```

4.2.3 Simulation Results



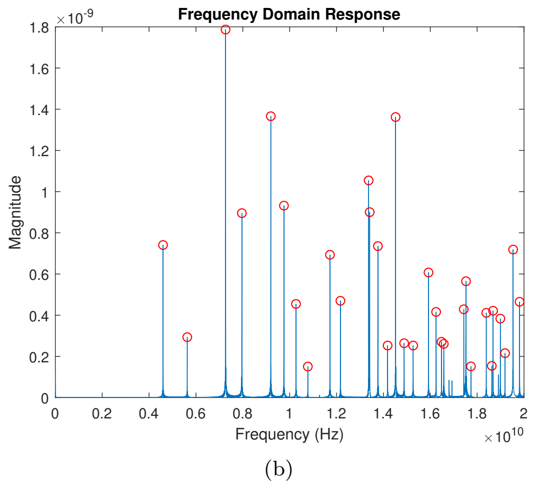


Figure 4.6 Cavity resonator simulation results for second and fourth order simulations.

Figure 4.6(a) is the second order simulation and Figure 4.6(b) is the fourth order simulation.

It is difficult to analyze the differences between the second order and fourth order simulations by only looking at Figure 4.6. The key take away from Figure 4.6 is that discrete peaks are clearly present and the peak finding algorithm is able to only detect the true peaks. The magnitude of the peaks differ between the second and fourth order simulations, but upon close examination, it can be deduced the peak frequencies are roughly the same on both simulations.

Table 4.1 Frequency Peaks vs. Calculated Mode Frequencies

Mode	Analytical f	Cells Per λ	2^{nd} Ord. f	2^{nd} Ord. % Error	4^{th} Ord. f	4^{th} Ord. % Error
(0,1,1)	4.5425 GHz	19.799	4.535 GHz	0.16602	4.595 GHz	1.1548
(1,1,1)	5.5635 GHz	16.166	5.555 GHz	0.15197	5.63 GHz	1.1961
(0,1,2)	7.1824 GHz	12.522	7.135 GHz	0.6598	7.265 GHz	1.1502
(1,1,2)	7.8679 GHz	11.431	7.825 GHz	0.54542	7.96 GHz	1.1704
(0,2,2)	9.0851 GHz	9.8995	9.02 GHz	0.71638	9.195 GHz	1.2099
(0,0,3)	9.6362 GHz	9.3333	9.575 GHz	0.63496	9.755 GHz	1.233
(0,1,3)	10.157 GHz	8.8544	10 GHz	1.5499	10.27 GHz	1.1082
(1,1,3)	10.653 GHz	8.4423	10.505 GHz	1.3912	10.775 GHz	1.1433
(0,2,3)	11.581 GHz	7.7658	11.425 GHz	1.3492	11.72 GHz	1.198
(1,2,3)	12.018 GHz	7.4833	11.87 GHz	1.2351	12.165 GHz	1.2195

As shown in table Table 4.1 both the second and fourth order simulations accurately simulate the resonant frequencies of the cavity. The "cells Per λ" column in the table tells us how many cells are in one wavelength of the analytically calculated frequency. As explained in section 4.1, the number of cells per wavelength is a measurment of how fine the computational grid is. More cells per wavelength will make the simulation more accurate. For second order simulations, about 20 cells per wavelength is desired [1]. Since the fourth order simulation is theoretically more accurate than the second order simulation, it should require less cells per wavelength and maintain a high accuracy. Figure 4.7 shows how the error in the predicted resonant frequencies changes as frequency increases and cells per wavelength decreases for both simulations.

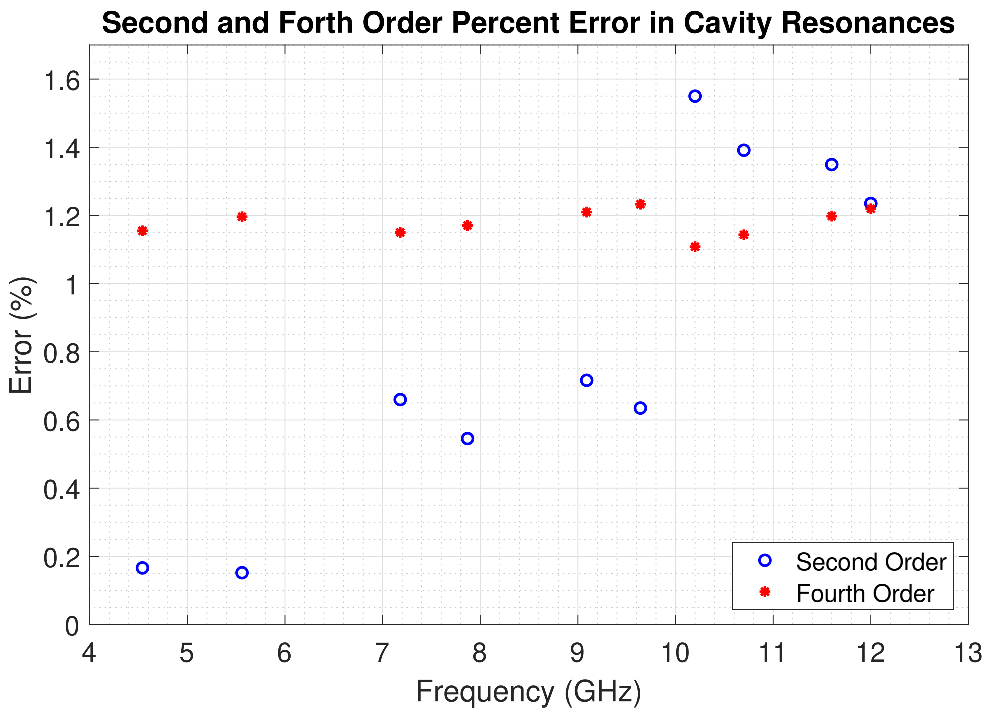


Figure 4.7 A plot showing the the percent error in the second and fourth order cavity resonators.

Figure 4.7 does not clearly show that the fourth order simulation overall has less error than the second order simulation. However, the error in the second order simulation generally increases as frequency increases and the cells per wavelength decrease from 19.8 to 7.5. Conversely, the error in the fourth order simulation stays roughly constant. This shows that the second order simulation has noticeable changes in error when the cells per wavelength ratio is reduced to 7.5, while the fourth order simulation is still accurate even at 7.5 cells per wavelength.

4.3 Basic Higher Order CPML Verification

The goal of this section is to compare the fourth order CPML formulation presented in chapter 3 to the second order CPML formulation. As mentioned in section 3.1, the three optimization parameters sigma factor (σ_f), kappa max (κ_{max}), and alpha max (α_{max}) likely have different optimized values for second and fourth orders. In this section, the optimum values for the second order are used with the fourth order CPML to understand if they need to be re-optimized. The second order optimum values are taken to be $\sigma_f = 1.5$, $\kappa_{max} = 7$, and $\alpha_{max} = 0.05$ from [1] and [25].

For these initial simulations using the fourth order CPML, a Gaussian pulse in free space is generated and allowed to interact with the CPML boundary. The Gaussian pulse has a maximum frequency of 5 cells per wavelength and is generated from a plate source. The total domain is 200 by 200 by 200 cells, and an E_z field component sample is taken every time step at approximately nx = 180, ny = 100, nz = 100 (so near the xp CPML boundary). The simulation was run for 1400 time steps for both the second and fourth order simulations. Figure 4.8 shows the geometry of this simulation while Figure 4.9 shows the results from both the second order and fourth order simulations and compares them.

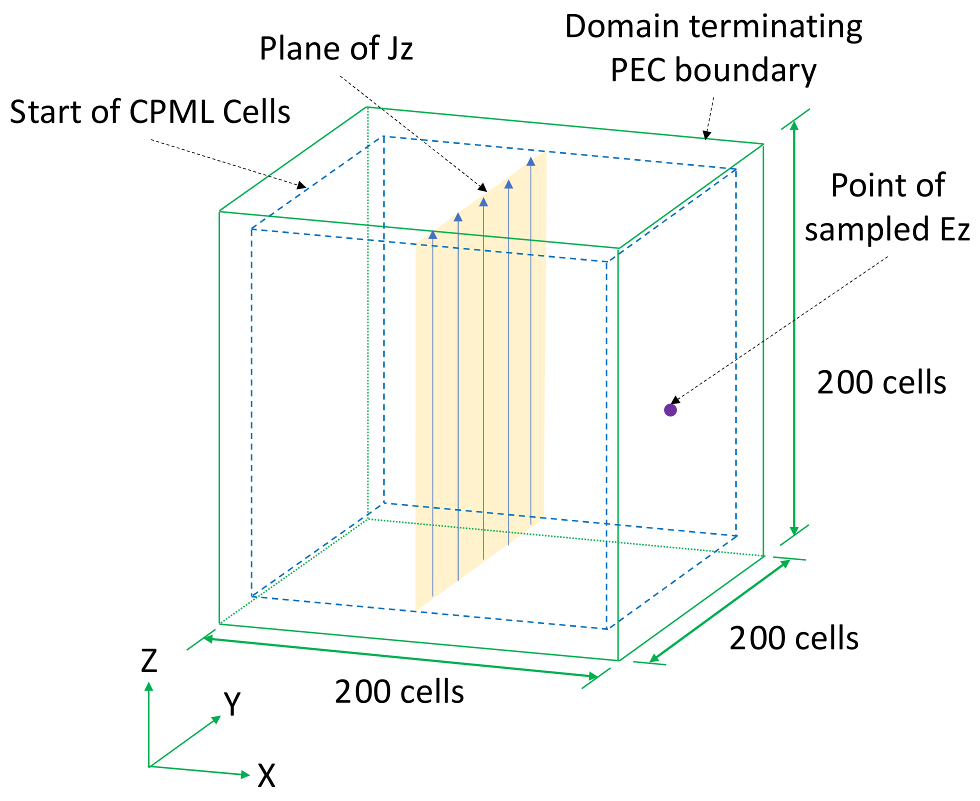
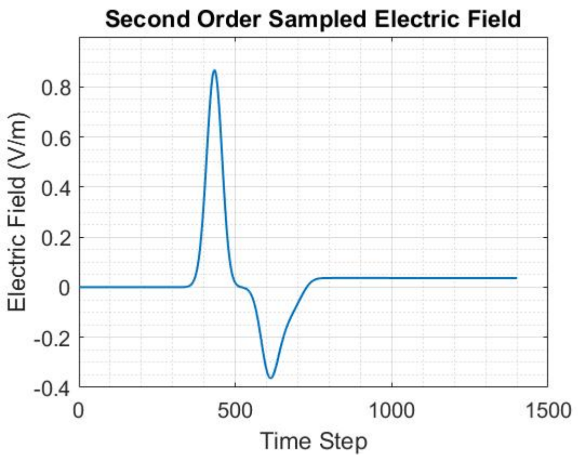
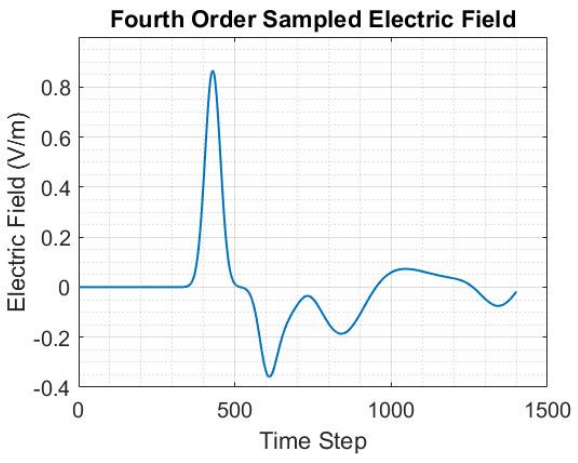


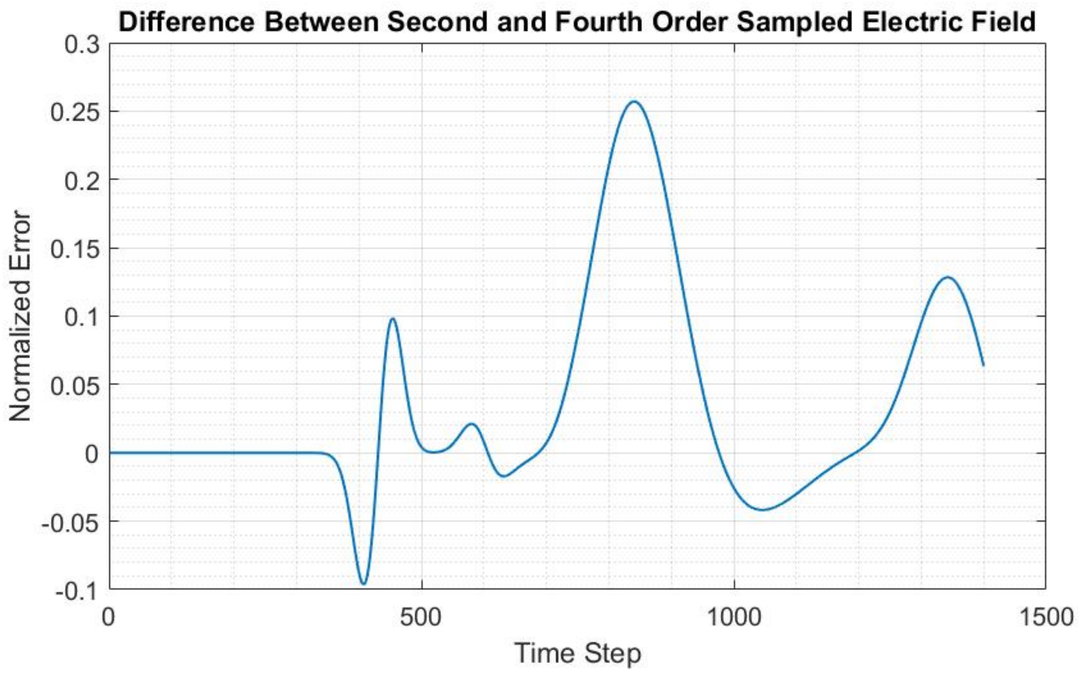
Figure 4.8 The geometry used to initially test the fourth order accurate CPML formulation.



(a)



(b)



(c)

Figure 4.9 a) second order sampled electric field, b) fourth order sample electric field, c) second order minus fourth order normalized to max of second order.

The Gaussian pulse goes into the CPML boundary at approximately time step 750. Figure 4.14(a) shows that the second order CPML boundary absorbed the Gaussian pulse well. However, it should be noted that the negative spike in the electric field around time step 850 in the fourth order simulation is the reflection of Gaussian pulse off the CPML boundary. Since the goal of CPML is to absorb all incoming waves, it is clear that the fourth order CPML parameters are not yet optimized. This is further highlighted in Figure 4.14(c) by noticing the large spikes at time step 850 and 1350.

To summarize, Figure 4.14 does not show simulation divergence for the fourth order CPML simulation meaning the formulation in chapter 3 is correct. However, the fourth order CPML does not absorb well using the second order optimizated values of $\sigma_f = 1.5$, $\kappa_{max} = 7$, and $\alpha_{max} = 0.05$. The next sections find the optimal values of these parameters for fourth order CPML.

4.4 Fourth Order CPML Parameter Sweeps

The goal of this section is to optimize the CPML optimization parameters for the fourth order CPML code implemented in section 4.3. The key parameters of the CPML that need to be optimized are sigma factor (σ_f), kappa max (κ_{max}), and alpha max (α_{max}). Using the values of these parameters that are optimized for the second order CPML as a starting place, a simple guess and check parameter optimization analysis can be performed. This analysis is done by varying the optimization parameters and measuring the CPML performance for each combination. This method will be referred to as a parameter sweep.

CPML boundaries are supposed to mimic an infinitly large computational domain. Therefore, the best way to measure the performance of the CPML is to simulate the same geometry in a small domain terminated with CPML boundaries and compare it to a large domain terminated with PEC boundaries. The large domain should be big enough and simulated for a short enough time such that no field is allowed to reflect off the boundary and interact with the sample point(s) before the simulation ends.

4.4.1 Simulation Set Up

These simulations use the higher order CPML implementation described in section 4.3 and utilize a simple thin wire dipole geometry (see Figure 4.10) as the source of field. Each half of the dipole consists of a thin wire with a radius of 0.05 mm, a length of 10 cells, and an orientation in the Z direction. The dipole is excited by a voltage source 2 cells long and set to produce a Gaussian voltage waveform with max frequency of 5 cells per λ. The simulation is run for 370 time steps with a Δt Courant factor of 0.5 for both the small and the large domains. The cell size is not of high importance for this analysis, but it should be stated the cell size for all simulations is 1/300 by 1/300 by 1/300 meters. The large domain is discretized with 300 by 300 by 322 cells and is terminated by PEC boundaries. The number of time steps, Courant factor, and size of the large domain are such that no fields are able to reach the boundary and come back to the sample point located 3 cells from the dipole in the X and Y directions. The small domain uses the same simulation set up as the large domain except that it is 14 by 14 by 36 cells large and terminated by 8 layers of CPML at the boundaries.

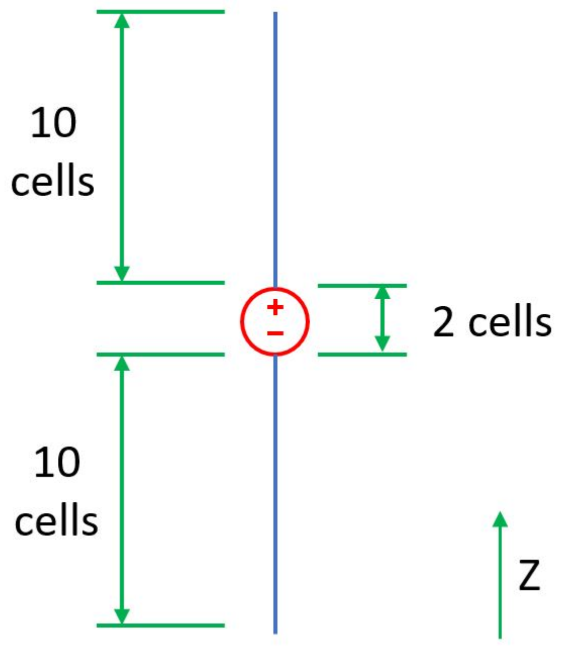


Figure 4.10 The thin wire dipole geometry used to collect sweep data.

4.4.2 Parameter Sweep Implimentation

To measure the effectivity of the CPML, a simulation with a large domain (large meaning no electromagnetic field reached the edges during sampling) and a simulation with a small domain (where the electromagnetic fields interacted with the CPML) are directly compared. This is done by having a point in space from which the E_z component of the electric field is measured. A measurement is taken every time step from the same location in space in the large domain and the small domain. These measurement arrays are then compared. For every sample in the arrays, the difference is calculated and then the absolute value of each difference is taken. Then the absolute values of the differences from every time step are added together. This sum will be referred to as the total accumulated difference between the small domain and the large domain. The log of this total accumulated difference is plotted as a function of σ_f and κ_{max} in Figure 4.11 and Figure 4.12.

There are a few things to keep in mind when thinking about the total accumulated difference. First, the total accumulated difference is not a measure of relative or percent difference as it is not normalized by the value of the electric field of the large domain. This is acceptable because the value of the difference is not important, the goal is to find the σ_f and κ_{max} that results in the least amount of difference between the large and small domains. Secondly, the total accumulated error will change with the number of time steps.

However, the same number of time steps (370) is used for every σ_f and κ_{max} value, so there is no issue.

The CPML's σ_f and κ_{max} values are the two parameters that appear to effect the performance of the CPML the most [25]. However, there are two other parameters that can have an effect: the order of the CPML and the α_{max} of the CPML. The order of the CPML is taken to be an integer number, generally 2 or 3 [1] and thus no sweep is needed. α_{max}, however, is not an integer and a sweep is needed for it. The results of sweeping over a range of α_{max} values while still sweeping over σ_f and κ_{max} is shown in Figure 4.13. Note these results are generated using the same process used by Gedney [25].

The parameter sweep code is run from a new file titled "parameter_sweep.m". The code for this file is shown in Listing 4.2. Note how this matlab code calls the fdtd_solve code that is generally the root starting program for the FDTD codes presented in this paper. "fdtd_solve.m" calls the "define_problem_space_parameters.m" code that sets the values for optimization parameters of the CPML (see Listing 4.3 lines 91-95). A new file titled "process_data.m" is called at the end of the "fdtd_solve.m" file and shown in Listing 4.4. The process data file is what calculates the total accumulated difference for the given CPML parameter values.

Listing 4.2: Parameter Sweep - root starting program

```
clear all; close all; clc;

sweepSize = 20;

sigma_factor_array = zeros(sweepSize,sweepSize);
kappa_array = zeros(sweepSize,sweepSize);
error_array = zeros(sweepSize,sweepSize);
simulation_Counter = 0;
for vu = 1:1
    for tu = 1:sweepSize
        for yu = 1:sweepSize
            disp(['Simulations Complete = ', num2str(simulation_Counter), ' out of '
                , num2str(sweepSize*sweepSize)]);
            simulation_Counter = simulation_Counter +1;
            fdtd_solve;
            sigma_factor_array(tu,yu) = boundary.cpml_sigma_factor;
            kappa_array(tu,yu) = boundary.cpml_kappa_max;
            error_array(tu,yu) = sumDiffData;

        end
    end
    figure(19)
    [dataPlot,h] = contour(sigma_factor_array,kappa_array,log(error_array),15);
    clabel(dataPlot,h)
    xlabel('Sigma Factor')
    ylabel('Kappa Max')
    title('Parameter Sweep: Alpha Max = 0.05')
    %title(['Parameter Sweep: Alpha Max = 0.05 CPML Cells = ', num2str(7+vu)])

    drawnow;
    F2(vu) = getframe(gcf);

end


% writerObj = VideoWriter('C:\Users\adesp\Documents\SchoolWork\Thesis\Deliverable
    Code and Results\Chapter 4 Verification Examples\Section 4.4
    S24_CPML_Parameter_Sweep(4D)\S24_CPML_Parameter_Sweep4D(dipole)_Results\4D_CPML_
    sweep.avi');
% writerObj.FrameRate = 15;
% % set the seconds per image
% % open the video writer
% open(writerObj);
% % write the frames to the video
% for pu=1:length(F2)
%     % convert the image to a frame
%     frame = F2(pu) ;
%     writeVideo(writerObj, frame);
% end
% % close the writer object
% close(writerObj);
```

Listing 4.3: Define Problem Space Parameters

```
disp('defining the problem space parameters');

cellSizeScale = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%parameters for creating 2D array in XZ plane.
array_dx = 20e-3;
array_dz = 30e-3;
n_elements_x = 1;
n_elements_z = 1;

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

% maximum number of time steps to run FDTD simulation
number_of_time_steps = 370; %4000

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.5;

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = 20;

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = 1/300;
dy = 1/300;
dz = 1/300;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 0;
tonyAirBufferCells = 7;%150 %4
tonySampleDistance = 3;%120 %3
tonyCPMLCells = 8;

% boundary.type_xn = 'pec';
% boundary.air_buffer_number_of_cells_xn = tonyAirBufferCells;
%
%
% boundary.type_xp = 'pec';
% boundary.air_buffer_number_of_cells_xp = tonyAirBufferCells;
%
%
% boundary.type_yn = 'pec';
% boundary.air_buffer_number_of_cells_yn = tonyAirBufferCells;
%
%
% boundary.type_yp = 'pec';
% boundary.air_buffer_number_of_cells_yp = tonyAirBufferCells;
%
%
% boundary.type_zn = 'pec';
% boundary.air_buffer_number_of_cells_zn = tonyAirBufferCells;
%
%
% boundary.type_zp = 'pec';
% boundary.air_buffer_number_of_cells_zp = tonyAirBufferCells;


boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = tonyAirBufferCells;
boundary.cpml_number_of_cells_xn = tonyCPMLCells;%8

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = tonyAirBufferCells;
boundary.cpml_number_of_cells_xp = tonyCPMLCells;%8

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = tonyAirBufferCells;
boundary.cpml_number_of_cells_yn = tonyCPMLCells;%8

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = tonyAirBufferCells;
boundary.cpml_number_of_cells_yp = tonyCPMLCells;%8

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = tonyAirBufferCells;
boundary.cpml_number_of_cells_zn = tonyCPMLCells;%8

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = tonyAirBufferCells;
boundary.cpml_number_of_cells_zp = tonyCPMLCells;%8

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0 + 0.1*yu; %1.3 %0.65
boundary.cpml_kappa_max = 0.5 + 0.05*tu; %7 %1
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05; %0.05

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

Listing 4.4: Parameter Sweep - process data

```
filename1 = 'C:\Users\adesp\Documents\SchoolWork\Thesis\Deliverable Code and Results
    \Chapter 4 Verification Examples\Section 4.4 S24_CPML_Parameter_Sweep(4D)\
    S24_CPML_Parameter_Sweep(dipole)big_Results\Big_Domain.csv';
largeDomainData = csvread(filename1);

differenceData = zeros(370);

differenceData = largeDomainData-EzSample1;

sumDiffData = sum(abs(differenceData));

```

```
% plot(differenceData)
% xlabel('Time Step')
% ylabel('Difference in Electric Field Strength')
% title(["Parameter Sweep, Total Difference: ", num2str(sumDiffData)],'fontsize',12)
% drawnow;
```

4.4.3 Two Parameter Sweep

The minimum total accumulated error needs to be determined by sweeping over three variables and plotting the results in an effective way. This technique is also performed in [25]. It is hard to plot a function that depends on three variables (effectively a 4D plot), so to begin the total accumulated error is only plotted as a function two of the three variables, κ_{max} and σ_f. α_{max} is set to 0.05, the optimum value for second order CPML [1]. Once these results are understood and the minimum found, α_{max} can be effectively swept as shown in section 4.4.4. Figure 4.11 shows the results of such a sweep where κ_{max} $= 0.5n$ and $\sigma_f = 0.4 + 0.2m$ where $n = 1, 2, 3...10$ and $m = 1, 2, 3...10$.

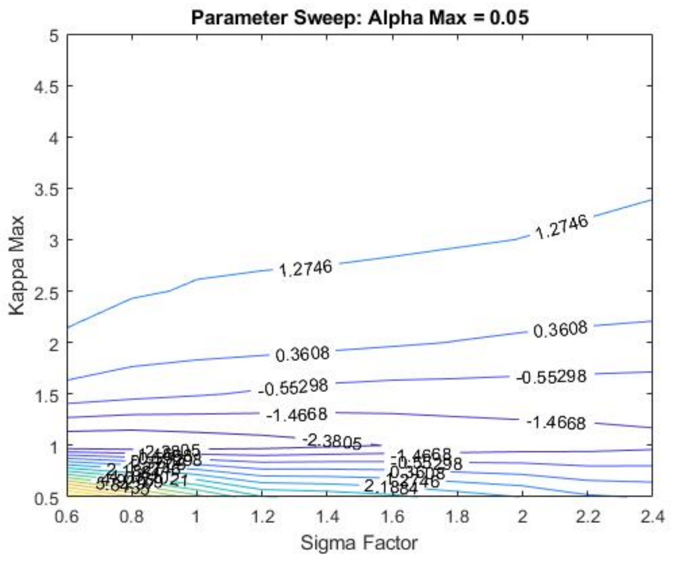


Figure 4.11 The initial parameter sweep over κ_{max} and σ_f. The total accumulated difference for 10 values of each variable is plotted.

After seeing the results of the first sweep, a second sweep was performed with a more refined range for κ_{max} and σ_f. The results of this refined sweep are shown in Figure 4.12 where $\kappa_{max} = 0.5 + 0.05n$ and σ_f $= 0.1m$ where $n = 1, 2, 3...20$ and $m = 1, 2, 3...20$.

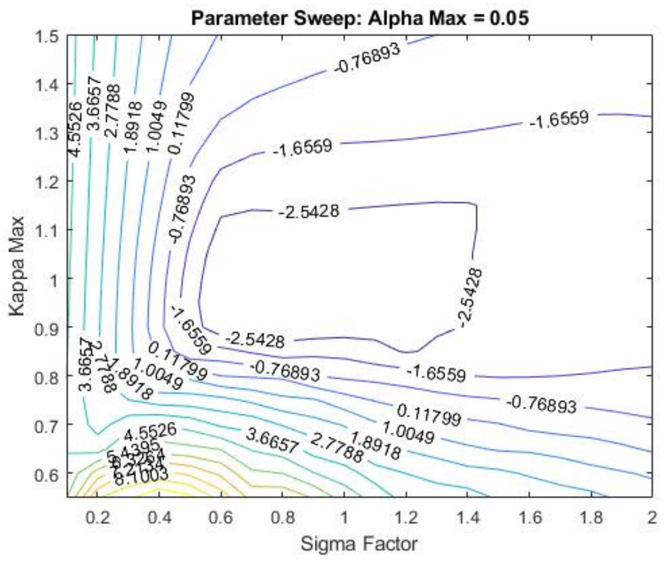
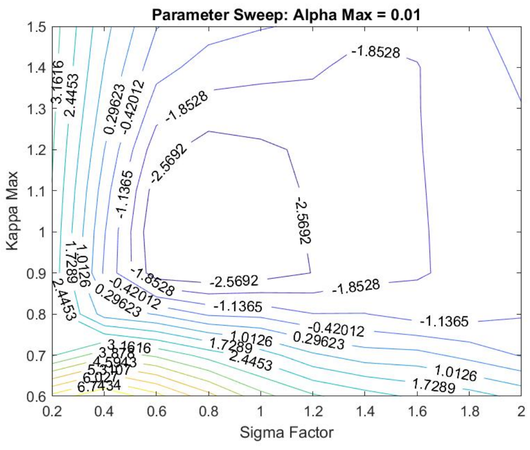


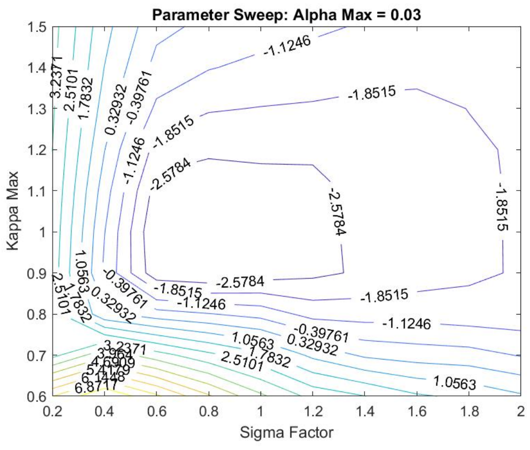
Figure 4.12 The refined parameter sweep over κ_{max} and σ_f. The total accumulated difference for 20 values of each variable is plotted.

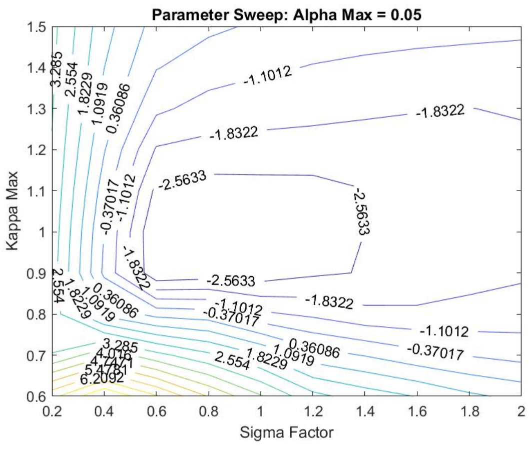
From Figure 4.12 it is clear CPML with a σ_f between 0.6 and 1.4 paired with a κ_{max} between 0.9 and 1.1 will perform the best for an α_{max} of 0.05.

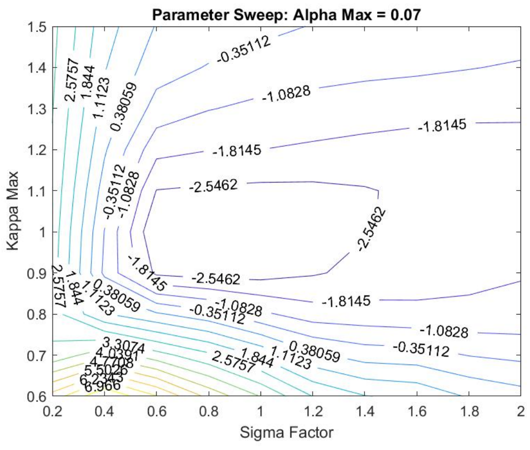
4.4.4 Three Parameter Sweep

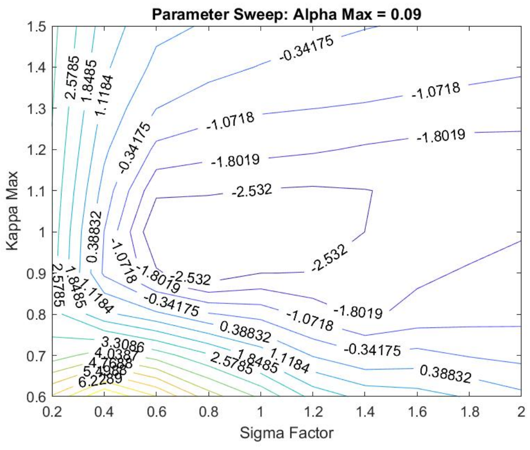
The goal of this section is to perform a sweep over all three variables that can effect the performance of the fourth order CPML. To do this, the same range of σ_f and κ_{max} as swept in Figure 4.12 is used, and then the total accumalted difference is calculated for 10 different α_{max} values. The optimum value of α_{max} in second order CPML is 0.05 according to [1], so an appropriate range of the ten α_{max} sweeping values would be from 0.01 to 0.1. The results of six of these α_{max} values are shown in Figure 4.13 where κ_{max} $= 0.5 + 0.1n$, σ_f $= 0.2m$, and $\alpha_{max} = 0.01p$ where $n = 1, 2, 3...10$, $m = 1, 2, 3...10$, and $p = 1, 2, 3...10$.


(a)


(b)


(c)


(d)


(e)

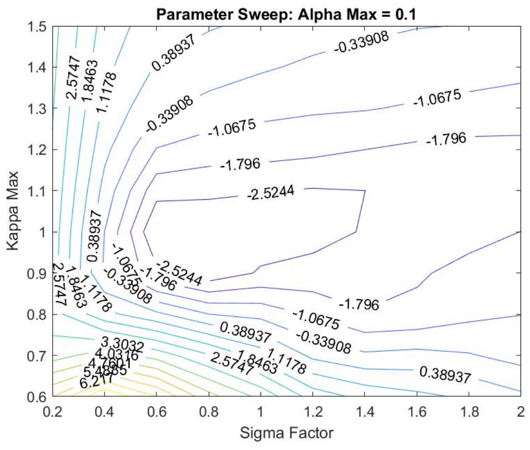
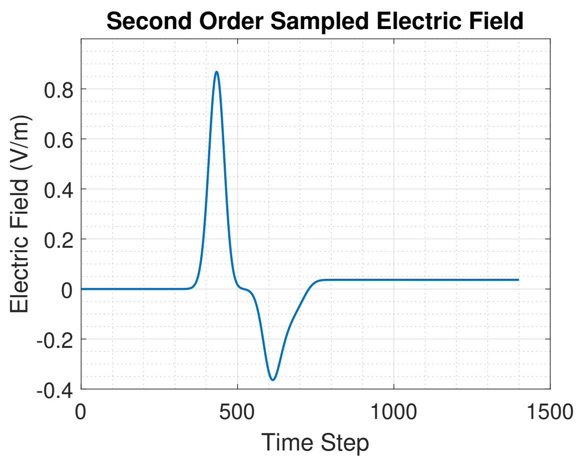

(f)

Figure 4.13 The results of the three CPML parameter sweep.

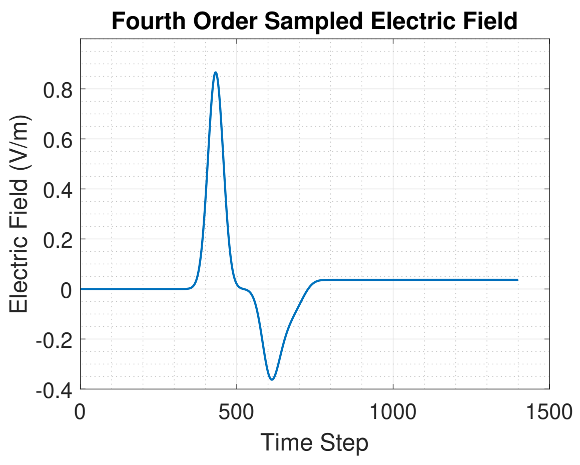
As shown in Figure 4.13, there is not much variation in the σ_f and κ_{max} sweep results as α_{max} changes values. The lowest value of the total accumulated difference of -25.8 dB happens when $\alpha_{max} = 0.03$. At the optimum α_{max} for the second order CPML (0.05) the lowest total accumulated difference is -25.6 dB. Since these total accumulated difference values are both very small and are very similar, an α_{max} of 0.05 will be considered optimum. The optimum σ_f and κ_{max} values will be considered to both be equal to 1. These three optimum values are used in section 4.5 to compare the effectiveness of the optimized fourth order CPML to the optimal second order CPML.

4.5 Second Order CPML Compared to Fourth Order CPML

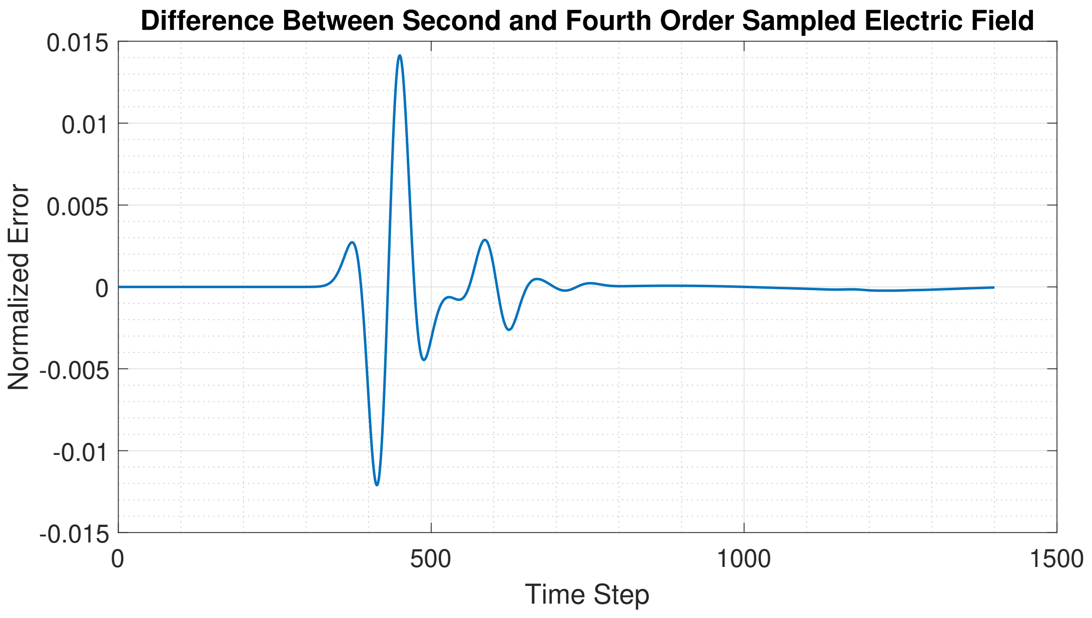
The goal of this section is to re-create the comparison done in section 4.3 but using the best CPML parameters from section 4.4.4. The same geometry and Gaussian pulse were used as in section Figure 4.14. Figure 4.14 shows the results of this comparison.



(a)



(b)

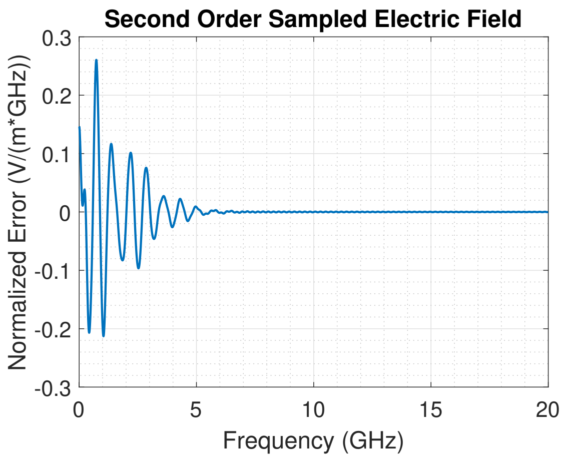


(c)

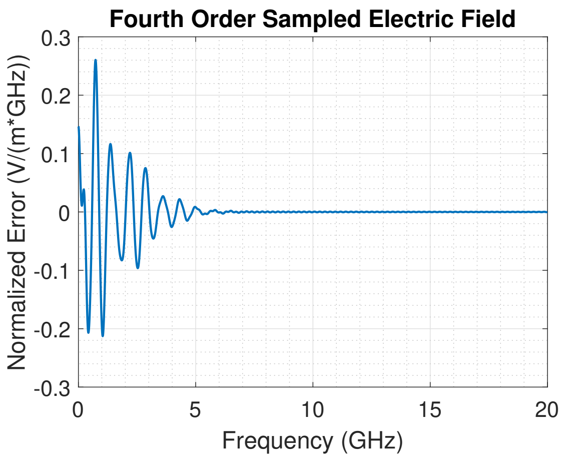
Figure 4.14 a) second order sampled electric field, b) fourth order sample electric field, c) second order minus fourth order normalized to max of second order.

As shown in Figure 4.14(a) and Figure 4.14(b), the reflections are almost zero in both second order and fourth order simulations since the sampled electric field flattens out at about 800 time steps. Figure 4.14(c) shows the relative difference (normalized to the maximum of the second order sampled data) between the fourth and the second order simulations. Since this relative difference is very small in magnitude (0.015 or 1.5%) even at it's maximum, it is reasonable to conclude that the optimized fourth order CPML is working as well as the second order CPML. The fact that the second and fourth order simulations are not exactly the same is expected. The fourth order simulation is more accurate than the second order simulation and a waveform with only five cells per λ_{min} is used to excite the system, meaning the second order simulation will be prone to errors. The larger differences shown from roughly time step 350 to 800 are most pronounced because those are the time steps the Gaussian pulse passes through the sample point. This is expected as the actual resolution of the pulse should be different between the second and fourth order simulations due to the fourth order simulation being more accurate. After the pulse passes through the sample point, both simulations should sample zero electric field at the sample point, indicating no field was reflected from the boundary. After time step 800, both Figure 4.14(a) and Figure 4.14(b) flatten out and Figure 4.14(c) is very close to zero as expected. Overall, Figure 4.14 shows strong evidence that the optimized fourth order CPML is absorbing incident waves as well as the second order CPML.

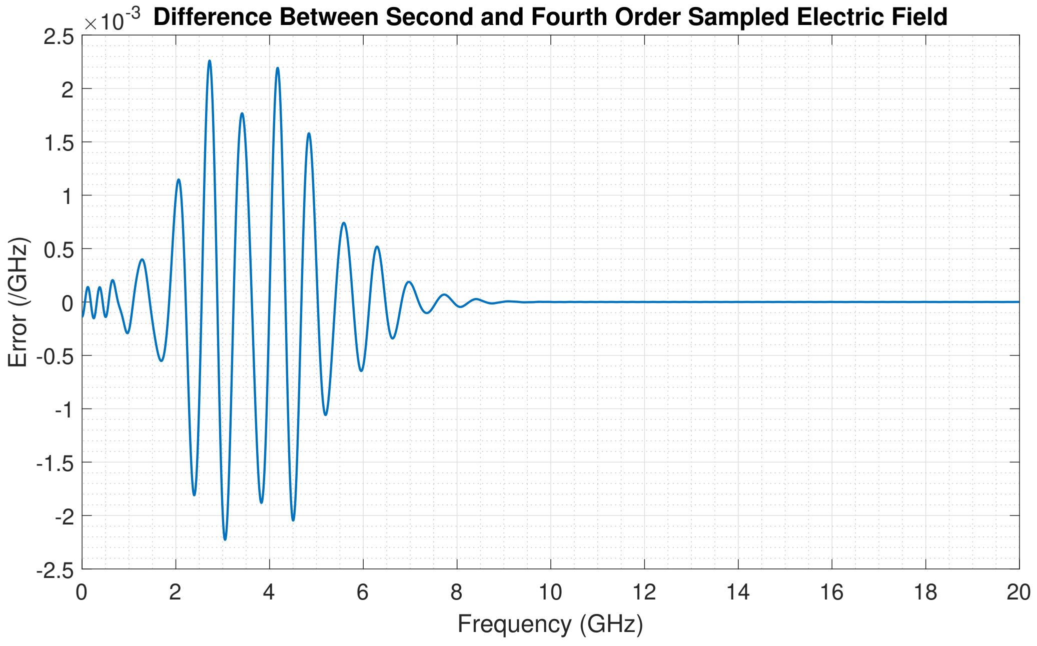
Figure 4.15 shows the results of Figure 4.14 converted into the frequency domain via a Fourier transform. Note that from Figure 4.15(a) and Figure 4.15(b) it appears the Gaussian pulses shown inFigure 4.14(a) and Figure 4.14(b) contain frequencies from roughly 0-6 GHz. The magnitude of these frequencies are at maximum about $0.26\frac{V}{m*GHz}$. The normalized error has a magnitude of roughly $2.2*10^{-3}\frac{V}{m*GHz}$, which is about 100 times less than the approximate magnitude of the second and fourth order frequency magnitudes. This also shows that fourth order CPML is performing very similarly to the second order CPML.



(a)



(b)



(c)

Figure 4.15 a) second order sampled electric field, b) fourth order sample electric field, c) second order minus fourth order error.

4.6 Thin Wire Dipole Antenna Simulation

The simulation of a thin wire dipole antenna is of interest as a verification example because it does not use PEC materials, and the expected results are well documented [1]. More information about dipole antennas can also be found in [28].

4.6.1 Thin Wire Dipole Simulation Setup

Figure 4.16 shows the problem space for the thin wire dipole simulation. The red dashed line shows the boundary of the CPML while the blue box shows the outer boundary of the domain.

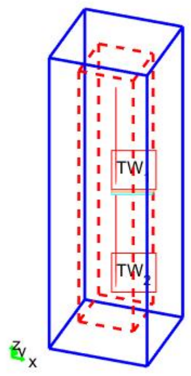


Figure 4.16 The problem space showing the thin wire dipole geometry.

More specifically, the complete dipole shown in Figure 4.16 is 20mm long and has a radius of 0.05 mm. At the center of the dipole, there is a voltage source half a millimeter long with an impedance of 50Ω. This voltage source generates a Gaussian voltage pulse with a maximum frequency that corresponds to a wavelength of 20 cells. The cell size of the simulation is 0.25mm. There are 10 cells of air buffer between the dipole antenna and the CPML boundary. The CPML boundary consists of 8 layers of cells.

Based on the analytical solutions for a dipole antenna, the maximum radiation (and minimum S_{11} reflection coefficient) should occur when the excitation frequency has a wavelength twice as long as the length of the dipole. In other words, maximum radiation will occur when the frequency is such that the antenna acts as a half wavelength dipole antenna.

In equation form:

$$L_{dipole} = \lambda/2 \tag{4.2}$$

For this specific dipole:

$$20 \times 10^{-3} m = \lambda/2 \tag{4.3}$$

$$40 \times 10^{-3} m = \lambda \tag{4.4}$$

$$f = \frac{c}{\lambda} \tag{4.5}$$

$$f = \frac{3 \times 10^8 m/s}{40 \times 10^{-3} m} \tag{4.6}$$

$$f = 7.5\, GHz \tag{4.7}$$

Since we are most interested in the response of the dipole when it is acting as a half wavelength dipole, the far field patterns will be calculated for 7.5 GHz. However, based on the analysis shown in section 5.1.1, a dipole antenna 20 mm long with a radius of 0.05 mm has a minum S_{11} of 7.122 GHz.

Code Listing 4.5, Listing 4.6, and Listing 4.8 define the simulation setup for both the second and fourth order simulations. Specifically, code Listing 4.5 shows the dimensions of the dipole in terms of number of cells. Information about the rest of the FDTD codes can be found in [1], if needed. The exact problem space parameters are slightly different between the second and fourth order simulations and code listings for these can be found in section 4.6.2.

Listing 4.5: define geometry (Thin Wire Dipole)

```
disp('defining the problem geometry');

bricks  = [];
spheres = [];
thin_wires = [];

% define a thin wire
thin_wires(1).min_x = 0;
thin_wires(1).min_y = 0;
thin_wires(1).min_z = cellSizeScale*0.25e-3;
thin_wires(1).max_x = 0;
thin_wires(1).max_y = 0;
thin_wires(1).max_z = 10e-3;
thin_wires(1).radius = 0.05e-3;
thin_wires(1).direction = 'z';

% define a thin wire
thin_wires(2).min_x = 0;
thin_wires(2).min_y = 0;
thin_wires(2).min_z = -10e-3;
thin_wires(2).max_x = 0;
thin_wires(2).max_y = 0;
thin_wires(2).max_z = -cellSizeScale*0.25e-3;
thin_wires(2).radius = 0.05e-3;
thin_wires(2).direction = 'z';
```

Listing 4.6: define sources and lumped circuit elements (Thin Wire Dipole)

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
```

```
resistors = [];
inductors = [];
capacitors = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% voltage sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude    : volts
voltage_sources(1).min_x = 0;
voltage_sources(1).min_y = 0;
voltage_sources(1).min_z = -cellSizeScale*0.25e-3;
voltage_sources(1).max_x = 0;
voltage_sources(1).max_y = 0;
voltage_sources(1).max_z = cellSizeScale*0.25e-3;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'gaussian';
voltage_sources(1).waveform_index = 1;
```

Listing 4.7: define output parameters (Thin Wire Dipole)

```
disp('defining output parameters');

sampled_electric_fields = [];
sampled_magnetic_fields = [];
sampled_voltages = [];
sampled_currents = [];
ports = [];
farfield.frequencies = [];

% figure refresh rate
plotting_step = 10;

% mode of operation
run_simulation = true;
show_material_mesh = true;
show_problem_space = true;

% far field calculation parameters
farfield.frequencies(1) = 7.0e9;
farfield.number_of_cells_from_outer_boundary = 13;

% frequency domain parameters
frequency_domain.start = 20e6;
frequency_domain.end   = 20e9;
frequency_domain.step  = 20e6;

% define sampled voltages
sampled_voltages(1).min_x = 0;
sampled_voltages(1).min_y = 0;
sampled_voltages(1).min_z = -cellSizeScale*0.25e-3;
sampled_voltages(1).max_x = 0;
sampled_voltages(1).max_y = 0;
sampled_voltages(1).max_z = cellSizeScale*0.25e-3;
sampled_voltages(1).direction = 'zp';
sampled_voltages(1).display_plot = false;

% define sampled currents
sampled_currents(1).min_x = 0;
sampled_currents(1).min_y = 0;
sampled_currents(1).min_z = 0;
sampled_currents(1).max_x = 0;
sampled_currents(1).max_y = 0;
sampled_currents(1).max_z = 0;
sampled_currents(1).direction = 'zp';
sampled_currents(1).display_plot = false;

% define ports
ports(1).sampled_voltage_index = 1;
ports(1).sampled_current_index = 1;
ports(1).impedance = 50;
ports(1).is_source_port = true;
```

4.6.2 Improved thin wire formulation results

Code Listing 4.8 shows the second order problem space parameters while Figure 4.17 shows the second order results for this simulation.

Listing 4.8: define problem space parameters (Second Order Thin Wire Dipole)

```
disp('defining the problem space parameters');

cellSizeScale = 1;

% maximum number of time steps to run FDTD simulation
number_of_time_steps = floor(4000/cellSizeScale); %4000

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9;

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = floor(20/cellSizeScale);

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = cellSizeScale*0.25e-3;
dy = cellSizeScale*0.25e-3;
dz = cellSizeScale*0.25e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 0;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1.3;
boundary.cpml_kappa_max = 7;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 2.2;
material_types(4).mu_r     = 1;
material_types(4).sigma_e  = 0;
material_types(4).sigma_m  = 0;
material_types(4).color    = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

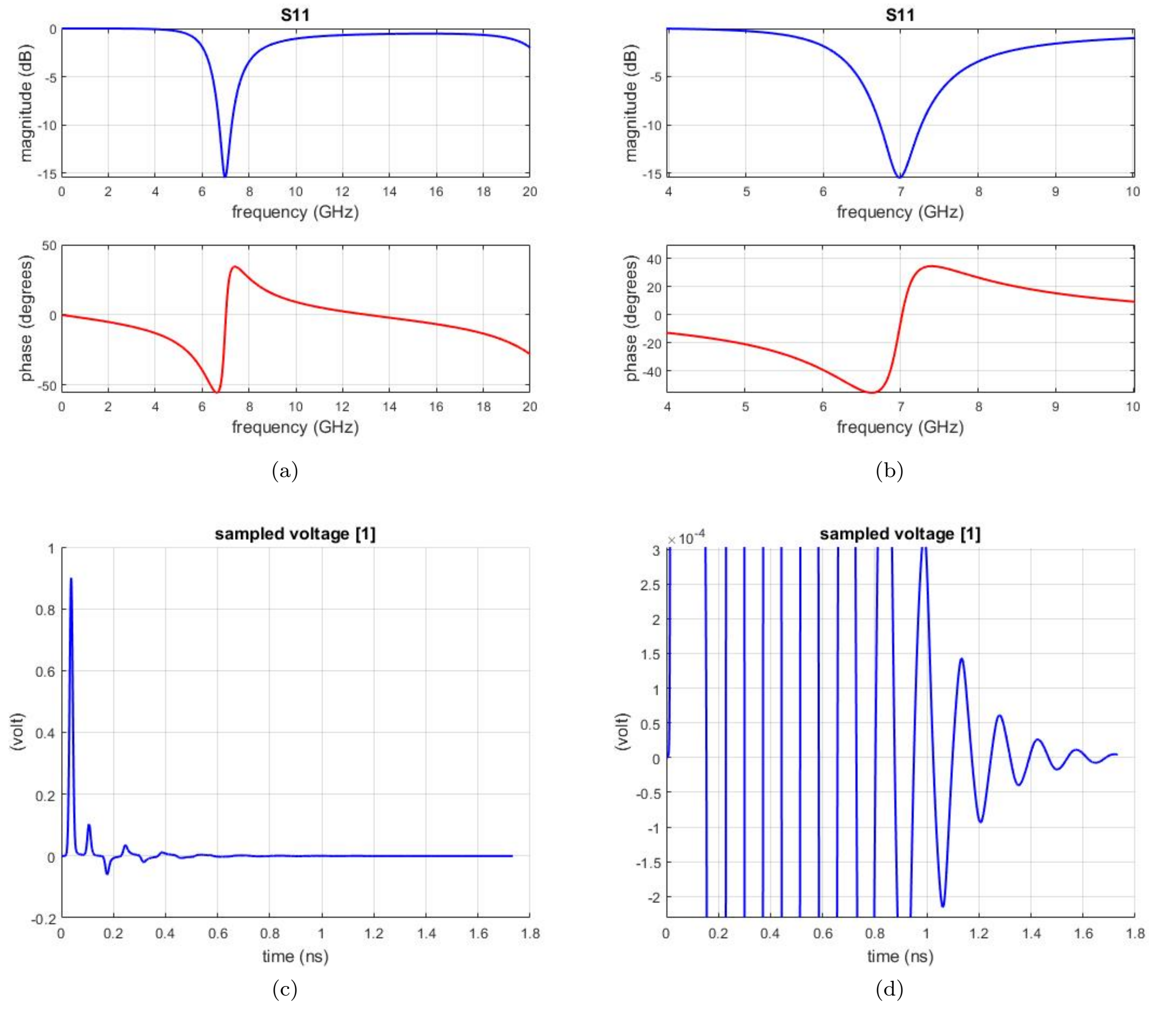


Figure 4.17 Second order thin wire dipole improved formulation results.

Code Listing 4.9 shows the fourth order problem space parameters while Figure 4.18 shows the fourth order results for this simulation. This simulation uses the fourth order thin wire formulation presented in section 2.2.4.

Listing 4.9: define problem space parameters (Fourth Order Thin Wire Dipole)

```
disp('defining the problem space parameters');

cellSizeScale = 1;

% maximum number of time steps to run FDTD simulation
number_of_time_steps = floor(4000/cellSizeScale); %4000

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7);

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = floor(20/cellSizeScale);

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = cellSizeScale*0.25e-3;
dy = cellSizeScale*0.25e-3;
dz = cellSizeScale*0.25e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 0;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = floor(10/cellSizeScale);
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0.6;
boundary.cpml_kappa_max = 1;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

```

```
% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

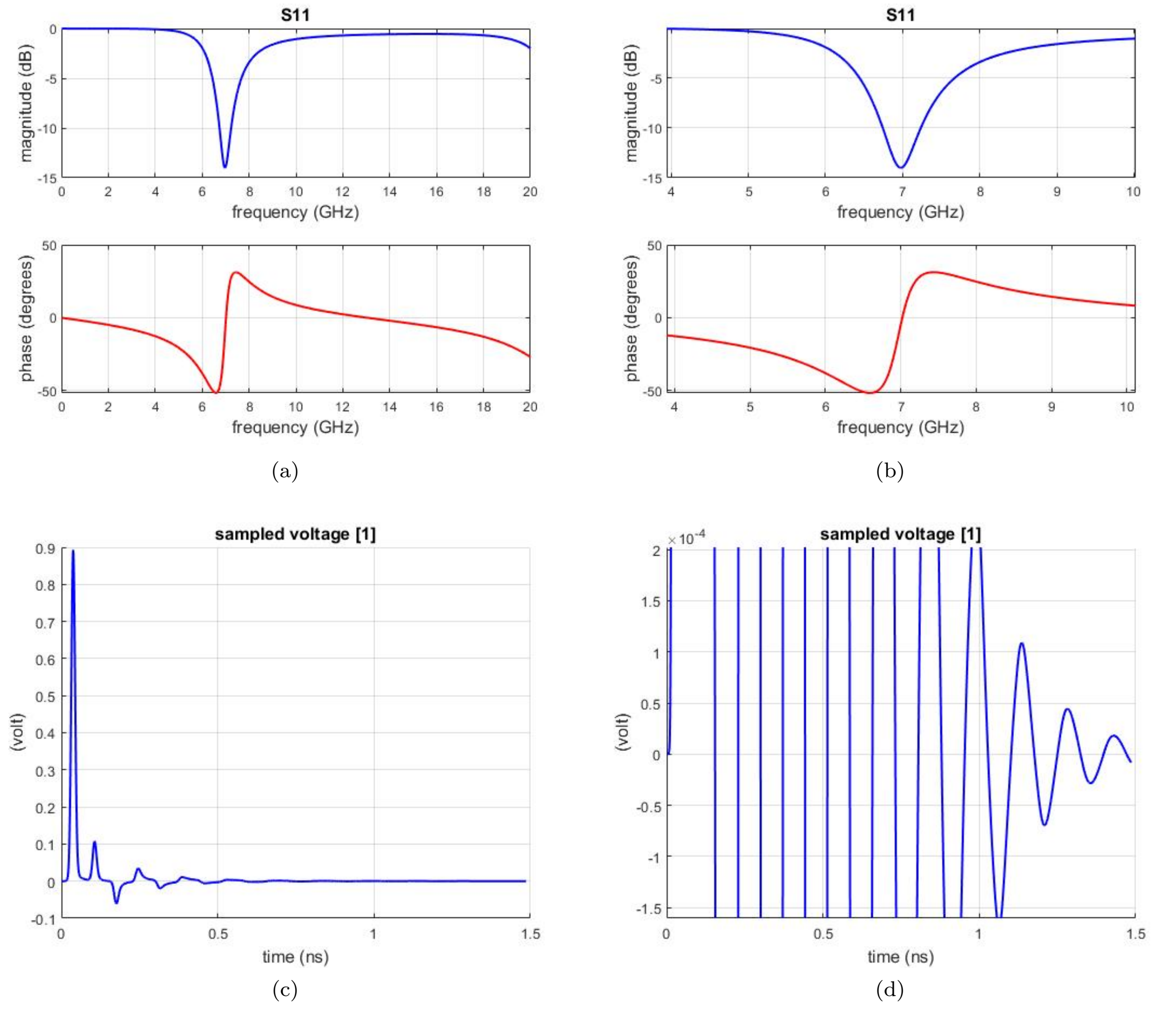
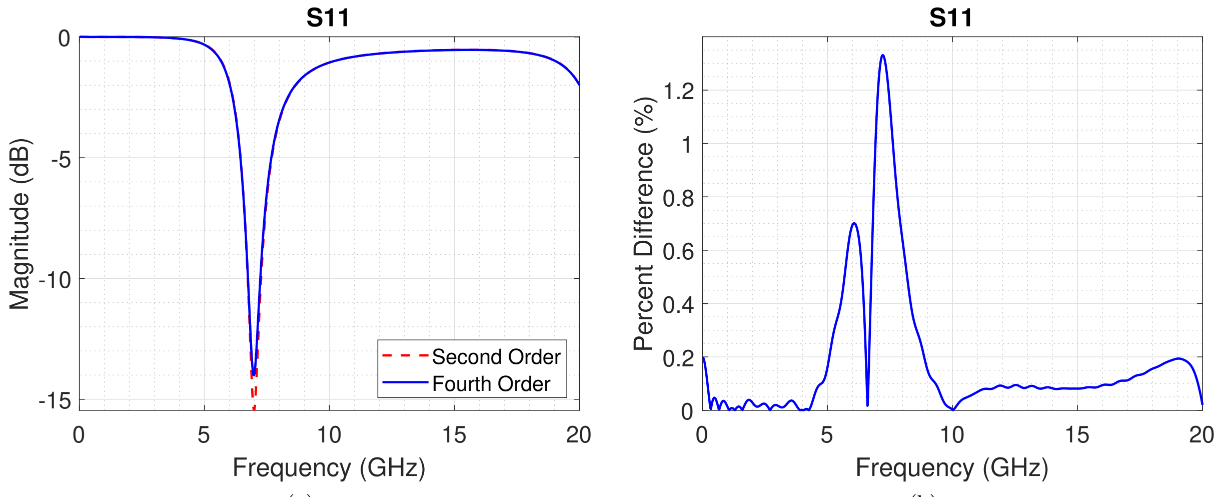


Figure 4.18 Fourth order thin wire dipole improved formulation results.

The results shown in Figure 4.17 and Figure 4.18 match almost exactly. Looking at the S_{11} plots for the second (Figure 4.17(a)) and fourth (Figure 4.18(a)) order simulations, the curves as a function of frequency are very smooth. This is important because in later sections the fourth order simulations of PEC objects have sharp peaks/valleys in the S_{11} curve that are incorrect. Additionally, the minimum S_{11} reflection coefficient in the second and fourth order simulations are both at 7 GHz, which is very close to the analytically predicted frequency of 7.122 GHz (1.71% error). Finally, the sampled voltage waveform dies out in the same way for the second and fourth order simulations. These observations all provide evidence that the fourth order simulation is working correctly.

Additionally, Figure 4.19 directly compares the S_{11} parameter simulated by the second and fourth order simulations of the thin wire dipole. Note that the percent difference in Figure 4.19(b) is calculated by taking the absolute value of the difference between the second and fourth order simulations (in linear scale, not dB scale), dividing by the maximum of the absolute value of the second order results, and then multiplying by 100 to arrive at a percent difference value. In equation form the percent difference is calculated by:

$$\frac{\left|10^{2^{nd}_{dB}/10} - 10^{4^{th}_{dB}/10}\right|}{max(10^{2^{nd}_{dB}/10})} \times 100. \tag{4.8}$$



(a) (b)

Figure 4.19 Direct comparison between the second and fourth order simulations of a thin wire dipole.

From Figure 4.19 it is clear the second order and fourth order simulation results for this thin wire dipole simulation are very similar as the maximum percent difference is 1.35%.

4.7 Problem with PEC Objects in Fourth Order Simulations

The goal of this section is to test how the fourth order simulation handles PEC objects. It has been shown by [23] and others that the larger mask of higher order updating equations have trouble simulating perfect electric conductor (PEC) objects. This section explores that issue further in the context of the specific fourth order updating equations presented in section 2.2.1. PEC bricks and PEC plates are two key PEC geometries that need to be explored. A PEC brick is generally defined by a region of PEC with finite dimensions (multiple cells) in all directions. A PEC plate is a two dimensional object only containing a plane of PEC field components. These two geometries interact with the fourth order updating equation's mask in different ways which is why they both need to be explored.

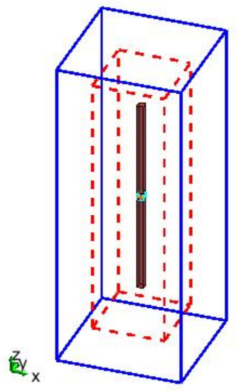
PEC bricks will be evaluated by using fourth order updating and simulating a dipole antenna consisting of two bricks (section 4.7.1). The PEC plate geometry is evaluated by simulating a simple PCB printed low-pass filter (section 4.7.2). As shown, the results of both these fourth order simulations are stable and close to the second order simulations for most of the frequency range. However, there are clear errors in the fourth order simulations in both geometries at lower frequencies.

4.7.1 Dipole Results

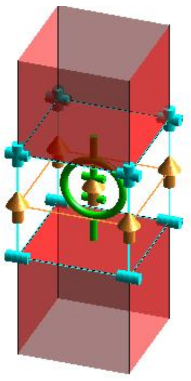
This section shows the results of a PEC dipole simulated with fourth order updating equations. Section 4.7.1.1 explains the problem setup and lists some key codes. Section 4.7.1.2 shows the results of the code for the second and fourth order simulations. The second order simulations serve as a reference as they match the peer reviewed results found in [1].

4.7.1.1 Simulation Setup

Figure 4.20 shows the geometry used in both the second order and fourth order simulations. The blue solid box in Figure 4.20(a) shows the outer boundary of the domain while the inner dashed red box shoes the inner boundary of the CPML layers. The voltage source is located between the PEC sides of the dipole (shown as red rectangles). The entire dipole has a length of 30 mm (15 mm for each side). Figure 4.20(b) shows a zoomed-in view of the voltage source and the start of both PEC sides of the dipole.



(a)



(b)

Figure 4.20 Dipole Geometry and source.

Code listings Listing 4.10, Listing 4.11, and Listing 4.12 define the simulation setup for both the second and fourth order simulations. Specifically, code Listing 4.10 shows the dimensions of the dipole in terms of number of cells. Information about the rest of the FDTD codes can be found in [1] if needed. The exact problem space parameters are slightly different between the second and fourth order simulations and code listings for these can be found in section 4.7.1.2.

Listing 4.10: define geometry (Brick Dipole)

```
disp('defining the problem geometry');

bricks  = [];
spheres = [];

% define substrate
bricks(1).min_x = -dx;
bricks(1).min_y = -dy;
bricks(1).min_z = dz;
bricks(1).max_x = dx;
bricks(1).max_y = dy;
bricks(1).max_z = 30*dz;
bricks(1).material_type = 2;

% define substrate
bricks(2).min_x = -dx;
bricks(2).min_y = -dy;
bricks(2).min_z = -30*dz;
bricks(2).max_x = dx;
bricks(2).max_y = dy;
bricks(2).max_z = -dz;
bricks(2).material_type = 2;
```

Listing 4.11: define sources and lumped elements (Brick Dipole)

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% voltage sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude   : volts
voltage_sources(1).min_x = -dx;
voltage_sources(1).min_y = -dy;
voltage_sources(1).min_z = -dz;
voltage_sources(1).max_x = dx;
voltage_sources(1).max_y = dy;
voltage_sources(1).max_z = dz;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'gaussian';
voltage_sources(1).waveform_index = 1;
```

Listing 4.12: define output parameters (Brick Dipole)

```
disp('defining output parameters');

sampled_electric_fields = [];
sampled_magnetic_fields = [];
sampled_voltages = [];
sampled_currents = [];
ports = [];
farfield.frequencies = [];

% figure refresh rate
plotting_step = 10;

% mode of operation
run_simulation = true;
show_material_mesh = true;
show_problem_space = true;

% far field calculation parameters
farfield.frequencies(1) = 4.5e9;
farfield.number_of_cells_from_outer_boundary = 13;

% frequency domain parameters
frequency_domain.start = 20e6;
frequency_domain.end   = 10e9;
frequency_domain.step  = 20e6;

% define sampled voltages
sampled_voltages(1).min_x = -dx;
sampled_voltages(1).min_y = -dy;
sampled_voltages(1).min_z = -dz;
sampled_voltages(1).max_x = dx;
sampled_voltages(1).max_y = dy;
sampled_voltages(1).max_z = dz;
sampled_voltages(1).direction = 'zp';
sampled_voltages(1).display_plot = false;

% define sampled currents
sampled_currents(1).min_x = -dx;
sampled_currents(1).min_y = -dy;
sampled_currents(1).min_z = 0;
sampled_currents(1).max_x = dx;
sampled_currents(1).max_y = dy;
sampled_currents(1).max_z = 0;
sampled_currents(1).direction = 'zp';
sampled_currents(1).display_plot = false;

% define ports
ports(1).sampled_voltage_index = 1;
ports(1).sampled_current_index = 1;
ports(1).impedance = 50;
ports(1).is_source_port = true;

% display problem space parameters
problem_space_display.labels = false;
problem_space_display.axis_at_origin = false;
problem_space_display.axis_outside_domain = true;
```

```
4.57  problem_space_display.grid_xn = false;
4.58  problem_space_display.grid_xp = false;
4.59  problem_space_display.grid_yn = false;
4.60  problem_space_display.grid_yp = false;
4.61  problem_space_display.grid_zn = false;
4.62  problem_space_display.grid_zp = false;
4.63  problem_space_display.outer_boundaries = true;
4.64  problem_space_display.cpml_boundaries = true;
```

4.7.1.2 Code Results

The problem space parameter code for the second order simulation is listed in code Listing 4.13. A few key parameters to notice are the courant factor, the cell sizes, and the CPML boundary parameters (lines 49-53).

Listing 4.13: define problem space parameters (Second Order Brick Dipole)

```
4.1   disp('defining the problem space parameters');
4.2
4.3   % maximum number of time steps to run FDTD simulation
4.4   number_of_time_steps = 40000;
4.5
4.6   % A factor that determines duration of a time step
4.7   % wrt CFL limit
4.8   courant_factor = 0.9;
4.9
4.10  % A factor determining the accuracy limit of FDTD results
4.11  number_of_cells_per_wavelength = 20;
4.12
4.13  % Dimensions of a unit cell in x, y, and z directions (meters)
4.14  dx = 0.5e-3;
4.15  dy = 0.5e-3;
4.16  dz = 0.5e-3;
4.17
4.18  % ==<boundary conditions>========
4.19  % Here we define the boundary conditions parameters
4.20  % 'pec' : perfect electric conductor
4.21  % 'cpml' : conlvolutional PML
4.22  % if cpml_number_of_cells is less than zero
4.23  % CPML extends inside of the domain rather than outwards
4.24
4.25  boundary.type_xn = 'cpml';
4.26  boundary.air_buffer_number_of_cells_xn = 10;
4.27  boundary.cpml_number_of_cells_xn = 8;
4.28
4.29  boundary.type_xp = 'cpml';
4.30  boundary.air_buffer_number_of_cells_xp = 10;
4.31  boundary.cpml_number_of_cells_xp = 8;
4.32
4.33  boundary.type_yn = 'cpml';
4.34  boundary.air_buffer_number_of_cells_yn = 10;
4.35  boundary.cpml_number_of_cells_yn = 8;
4.36
4.37  boundary.type_yp = 'cpml';
4.38  boundary.air_buffer_number_of_cells_yp = 10;
```

```
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 10;
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = 10;
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1.3;
boundary.cpml_kappa_max = 7;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

Figure 4.21 shows the results of the second order simulation to provide a reference to compare the fourth order simulation results to. There are a few key features to notice that are different in the fourth order simulation. As seen by examining Figure 4.21(a) and Figure 4.21(b), the sampled voltage completely dies out after about 2 ns. This is the expected response of a simulation with a short Gaussian waveform source. Another key point is that Figure 4.21(c) has a smooth curve showing the S_{11} reflection coefficient as a function of frequency, as expected.

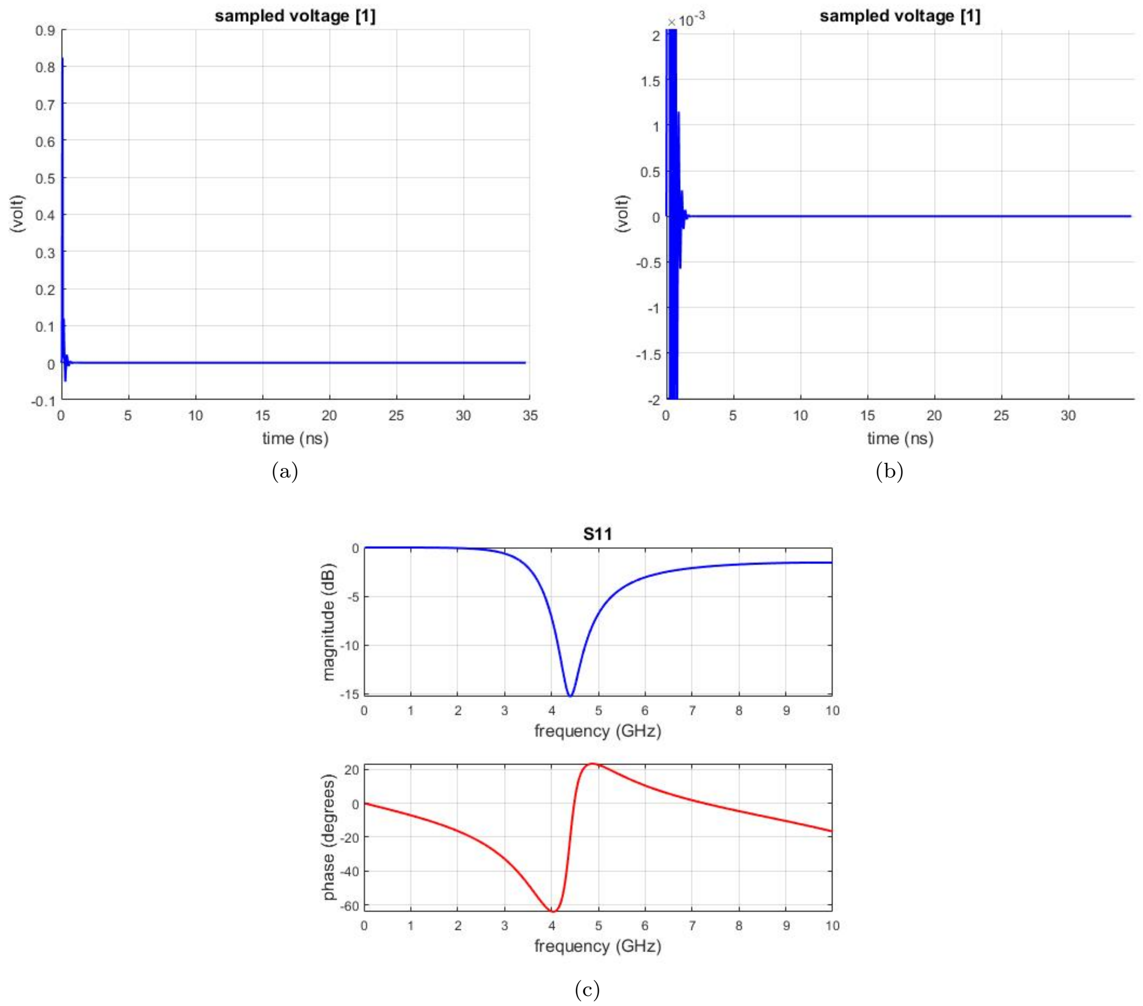


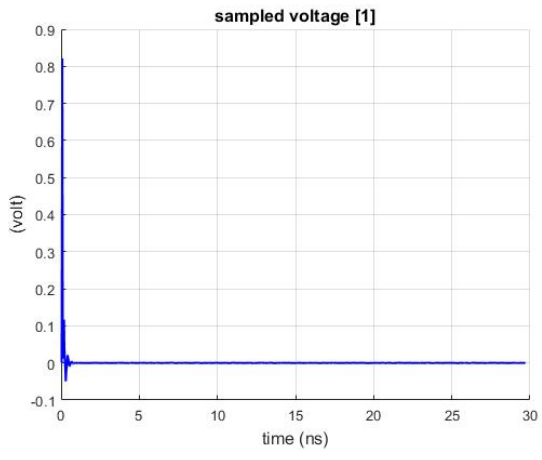
Figure 4.21 Reference second order PEC brick dipole with CPML boundaries.

The problem space parameter code for the fourth order simulation is listed in code Listing 4.14. A few key parameters to notice are the courant factor, the cell sizes, and the CPML boundary parameters. The courant factor has been adjusted to adhere to the stability criterion of a fourth order FDTD simulation (see section 2.2.3.3). The cells sizes are the same as the second order simulation to provide a good comparison. Finally, the CPML boundary parameters are different than the second order simulation. The κ_{max} and σ_{factor} match the optimum values for a fourth order simulation as found in section 4.4.

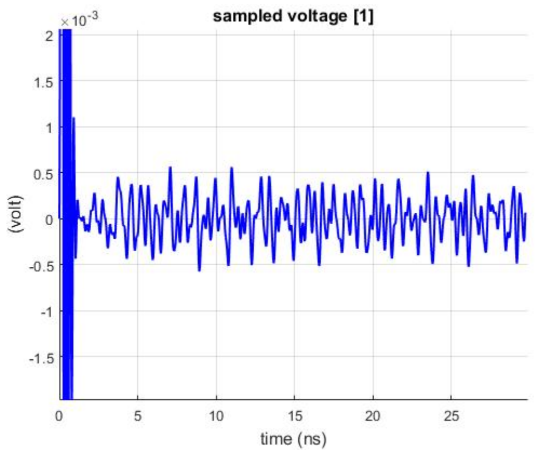
Listing 4.14: define problem space parameters (Fourth Order Brick Dipole)

```
disp('defining the problem space parameters');

% maximum number of time steps to run FDTD simulation
number_of_time_steps = 40000; %40000; %157269; %141540;

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7); %0.5; %0.12717;

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = 20;

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = 0.5e-3;
dy = 0.5e-3;
dz = 0.5e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 0;
tonyAirBufferSize = 10;
tonyCPMLCells = 8;

%CPML
boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = tonyAirBufferSize;
boundary.cpml_number_of_cells_xn = tonyCPMLCells;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = tonyAirBufferSize;
boundary.cpml_number_of_cells_xp = tonyCPMLCells;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = tonyAirBufferSize;
boundary.cpml_number_of_cells_yn = tonyCPMLCells;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = tonyAirBufferSize;
boundary.cpml_number_of_cells_yp = tonyCPMLCells;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = tonyAirBufferSize;
boundary.cpml_number_of_cells_zn = tonyCPMLCells;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = tonyAirBufferSize;
boundary.cpml_number_of_cells_zp = tonyCPMLCells;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1; %0.6
boundary.cpml_kappa_max = 1; %7
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>=============
% Here we define and initialize the arrays of material types
% eps_r    : relative permittivity
% mu_r     : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r    = 1;
material_types(1).mu_r     = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color    = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r    = 1;
material_types(2).mu_r     = 1;
material_types(2).sigma_e = 1e20;
material_types(2).sigma_m = 0;
material_types(2).color    = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r    = 1;
material_types(3).mu_r     = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color    = [0 1 0];

% substrate
material_types(4).eps_r    = 2.2;
material_types(4).mu_r     = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color    = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

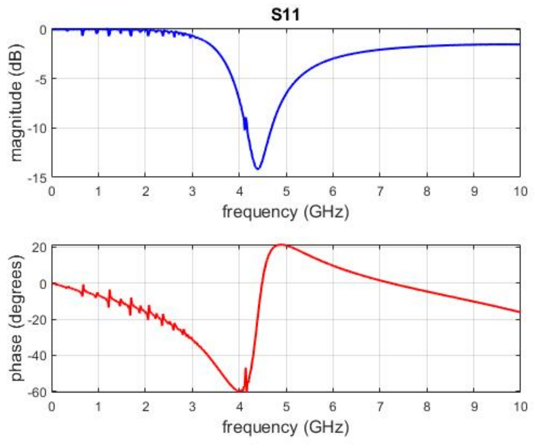
Figure 4.22 shows the results of the fourth order simulation. As shown by looking at Figure 4.22(a) and Figure 4.22(b), the sampled voltage does not zero out and in fact appears to oscillate indefinitely. This is not expected and is likely caused by the errors created by the large mask of the fourth order updating equations interacting with the PEC bricks of the dipole. Additionally, the S_{11} as a function of frequency curve is not smooth. It has many peaks and valleys at lower frequencies likely caused by the fact the sampled voltage did not die out. Further analysis into exactly what causes these low frequency errors is presented in section 4.8.



(a)



(b)



(c)

Figure 4.22 Fourth order PEC brick dipole results.

Additionally, Figure 4.23 directly compares the S_{11} parameter simulated by the second and fourth order simulations. Note that the percent difference in Figure 4.23(b) is calculated by taking the absolute value of the difference between the second and fourth order simulations (in linear scale, not dB scale), dividing by the maximum of the absolute value of the second order results, and then multiplying by 100 to arrive at a percent difference value. In equation form the percent difference is calculated by:

$$\frac{\left|10^{2^{nd}_{dB}/10} - 10^{4^{th}_{dB}/10}\right|}{max(10^{2^{nd}_{dB}/10})} \times 100. \tag{4.9}$$

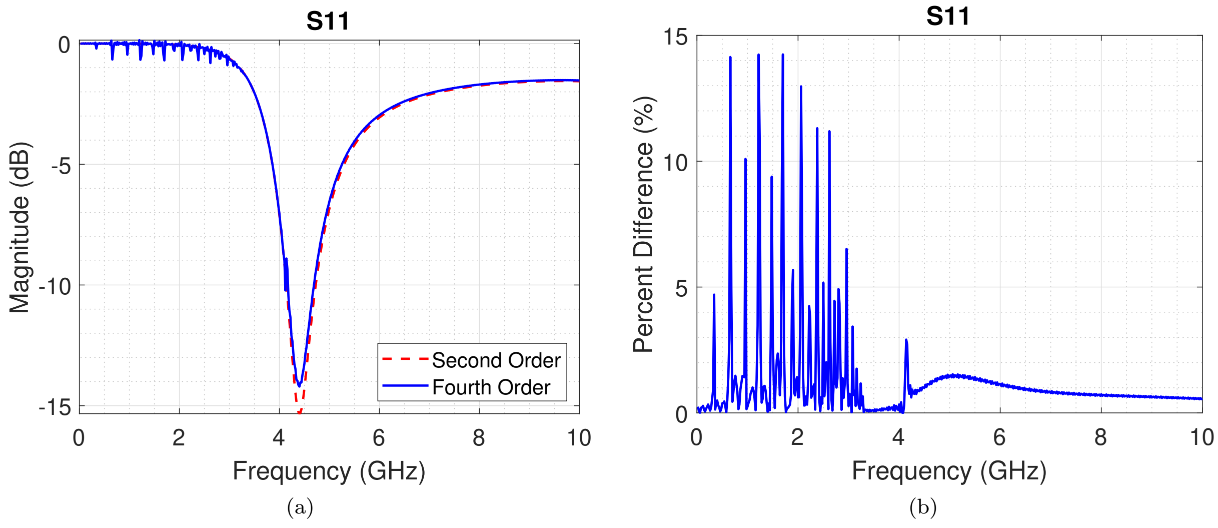


Figure 4.23 Direct comparison between the second and fourth order simulations of a PEC brick dipole.

From Figure 4.23 it is clear the second order and fourth order simulation results for this PEC brick dipole simulation are not similar over the entire 0-10 GHz frequency range. Note that large differences are evident in the low frequency range, roughly between 0 and 3.5 GHz. As frequency increases the percent difference between the second order and fourth order simulations decreases, with a maximum percent difference of 2% for frequencies between 4.5 and 10 GHz.

4.7.2 Microstrip Low-Pass Filter Results

This section shows the results of a microstrip low-pass filter simulated with PEC plates using fourth order updating equations. Section 4.7.2.1 explains the problem setup and lists some key codes. Section 4.7.2.2 shows the results of the code for the second and fourth order simulations. The second order simulations serve as a reference as they match the peer reviewed results found in [1].

4.7.2.1 Simulation Setup

Figure 4.24 shows the geometry used in both the second order and fourth order simulations. The solid blue box in Figure 4.24 shows the outer boundary of the domain while the inner dashed red box shows the inner boundary of the CPML layers. Figure 4.25 defines the exact dimension of the filter. In addition to Figure 4.25, it should be noted that the substrate of the filter is 3 cells thick. The opposite side of the substrate consists of a PEC plate ground plane that the voltage sources and the positive terminal of the ports connect to. Code Listing 4.15 shows all the details of the filter's geometry. It is hard to see in Figure 4.24, but the air buffer in the negative Z-direction is set to zero and thus the CPML layers start on the ground plane. This means the ground plane is connected directly to the perfectly absorbing CPML layers.

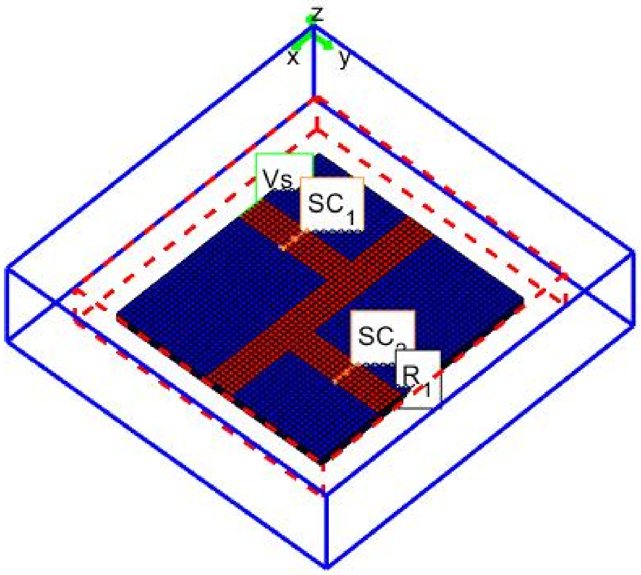


Figure 4.24 The geometry of the microstrip low-pass filter.

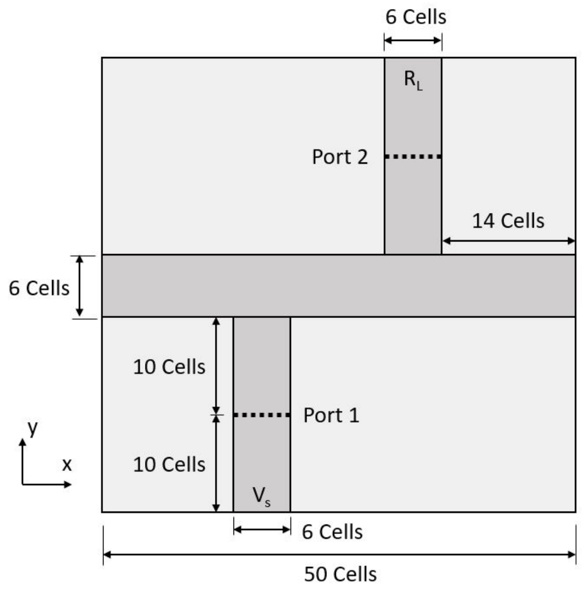


Figure 4.25 The dimensions of the microstrip low-pass filter.

Code Listing 4.15, Listing 4.16, and Listing 4.17 define the simulation setup for both the second and fourth order simulations. Specifically, code Listing 4.15 shows the dimensions of the filter in terms of number of cells. Information about the rest of the FDTD codes can be found in [1] if needed. The exact problem space parameters are slightly different between the second and fourth order simulations, and code listings for these can be found in section 4.7.2.2.

Listing 4.15: define geometry (Filter)

```
disp('defining the problem geometry');

bricks  = [];
spheres = [];

% define a substrate
bricks(1).min_x = 0;
bricks(1).min_y = 0;
bricks(1).min_z = 0;
bricks(1).max_x = 50*dx;
bricks(1).max_y = 46*dy;
bricks(1).max_z = 3*dz;
bricks(1).material_type = 4;

% define a PEC plate
bricks(2).min_x = 14*dx;
bricks(2).min_y = 0;
bricks(2).min_z = 3*dz;
bricks(2).max_x = 20*dx;
bricks(2).max_y = 20*dy;
bricks(2).max_z = 3*dz;
bricks(2).material_type = 2;

% define a PEC plate
bricks(3).min_x = 30*dx;
bricks(3).min_y = 26*dy;
bricks(3).min_z = 3*dz;
bricks(3).max_x = 36*dx;
bricks(3).max_y = 46*dy;
bricks(3).max_z = 3*dz;
bricks(3).material_type = 2;

% define a PEC plate
bricks(4).min_x = 0;
bricks(4).min_y = 20*dy;
bricks(4).min_z = 3*dz;
bricks(4).max_x = 50*dx;
bricks(4).max_y = 26*dy;
bricks(4).max_z = 3*dz;
bricks(4).material_type = 2;

% define a PEC plate as ground
bricks(5).min_x = 0;
bricks(5).min_y = 0;
bricks(5).min_z = 0;
bricks(5).max_x = 50*dx;
bricks(5).max_y = 46*dy;
bricks(5).max_z = 0;
bricks(5).material_type = 2;
```

Listing 4.16: define sources and lumped elements (Filter)

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% voltage sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude   : volts
voltage_sources(1).min_x = 14*dx;
voltage_sources(1).min_y = 0;
voltage_sources(1).min_z = 0;
voltage_sources(1).max_x = 20*dx;
voltage_sources(1).max_y = 0;
voltage_sources(1).max_z = 3*dz;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'gaussian';
voltage_sources(1).waveform_index = 1;

% resistors
% direction: 'x', 'y', or 'z'
% resistance : ohms
resistors(1).min_x = 30*dx;
resistors(1).min_y = 46*dy;
resistors(1).min_z = 0;
resistors(1).max_x = 36*dx;
resistors(1).max_y = 46*dy;
resistors(1).max_z = 3*dz;
resistors(1).direction = 'z';
resistors(1).resistance = 50;
```

Listing 4.17: define output parameters (Filter)

```
disp('defining output parameters');

sampled_electric_fields = [];
sampled_magnetic_fields = [];
sampled_voltages = [];
sampled_currents = [];
ports = [];

% figure refresh rate
```

```
plotting_step = 200;

% mode of operation
run_simulation = true;
show_material_mesh = true;
show_problem_space = true;

% frequency domain parameters
frequency_domain.start = 20e6;
frequency_domain.end   = 20e9;
frequency_domain.step  = 20e6;

% define sampled voltages
sampled_voltages(1).min_x = 14*dx;
sampled_voltages(1).min_y = 10*dy;
sampled_voltages(1).min_z = 0;
sampled_voltages(1).max_x = 20*dx;
sampled_voltages(1).max_y = 10*dy;
sampled_voltages(1).max_z = 3*dz;
sampled_voltages(1).direction = 'zp';
sampled_voltages(1).display_plot = false;

sampled_voltages(2).min_x = 30*dx;
sampled_voltages(2).min_y = 36*dy;
sampled_voltages(2).min_z = 0.0;
sampled_voltages(2).max_x = 36*dx;
sampled_voltages(2).max_y = 36*dy;
sampled_voltages(2).max_z = 3*dz;
sampled_voltages(2).direction = 'zp';
sampled_voltages(2).display_plot = false;

% define sampled currents
sampled_currents(1).min_x = 14*dx;
sampled_currents(1).min_y = 10*dy;
sampled_currents(1).min_z = 3*dz;
sampled_currents(1).max_x = 20*dx;
sampled_currents(1).max_y = 10*dy;
sampled_currents(1).max_z = 3*dz;
sampled_currents(1).direction = 'yp';
sampled_currents(1).display_plot = false;

sampled_currents(2).min_x = 30*dx;
sampled_currents(2).min_y = 36*dy;
sampled_currents(2).min_z = 3*dz;
sampled_currents(2).max_x = 36*dx;
sampled_currents(2).max_y = 36*dy;
sampled_currents(2).max_z = 3*dz;
sampled_currents(2).direction = 'yn';
sampled_currents(2).display_plot = false;

% define ports
ports(1).sampled_voltage_index = 1;
ports(1).sampled_current_index = 1;
ports(1).impedance = 50;
ports(1).is_source_port = true;

ports(2).sampled_voltage_index = 2;
ports(2).sampled_current_index = 2;
ports(2).impedance = 50;
```

```
4.69 ports(2).is_source_port = false;
```

4.7.2.2 Code Results

The problem space parameter code for the second order simulation is listed in code Listing 4.18. A few key parameters to notice are the courant factor, the cell sizes, and the CPML boundary parameters (lines 49-53).

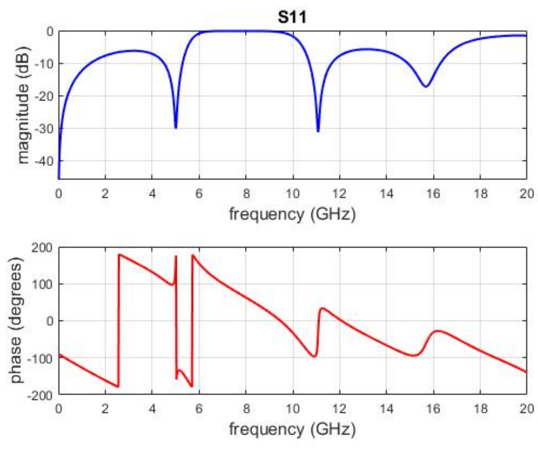
Listing 4.18: define problem space parameters (Second Order Filter)

```
4.1  disp('defining the problem space parameters');
4.2
4.3  % maximum number of time steps to run FDTD simulation
4.4  number_of_time_steps = 6000; %2000
4.5
4.6  % A factor that determines duration of a time step
4.7  % wrt CFL limit
4.8  courant_factor = 0.9;
4.9
4.10 % A factor determining the accuracy limit of FDTD results
4.11 number_of_cells_per_wavelength = 20;
4.12
4.13 % Dimensions of a unit cell in x, y, and z directions (meters)
4.14 dx = 0.4064e-3;
4.15 dy = 0.4233e-3;
4.16 dz = 0.265e-3; %to match S24
4.17
4.18 % ==<boundary conditions>========
4.19 % Here we define the boundary conditions parameters
4.20 % 'pec' : perfect electric conductor
4.21 % 'cpml' : conlvolutional PML
4.22 % if cpml_number_of_cells is less than zero
4.23 % CPML extends inside of the domain rather than outwards
4.24
4.25 boundary.type_xn = 'cpml';
4.26 boundary.air_buffer_number_of_cells_xn = 5;
4.27 boundary.cpml_number_of_cells_xn = 8;
4.28
4.29 boundary.type_xp = 'cpml';
4.30 boundary.air_buffer_number_of_cells_xp = 5;
4.31 boundary.cpml_number_of_cells_xp = 8;
4.32
4.33 boundary.type_yn = 'cpml';
4.34 boundary.air_buffer_number_of_cells_yn = 5;
4.35 boundary.cpml_number_of_cells_yn = 8;
4.36
4.37 boundary.type_yp = 'cpml';
4.38 boundary.air_buffer_number_of_cells_yp = 5;
4.39 boundary.cpml_number_of_cells_yp = 8;
4.40
4.41 boundary.type_zn = 'cpml';
4.42 boundary.air_buffer_number_of_cells_zn = 0;
4.43 boundary.cpml_number_of_cells_zn = 8;
4.44
4.45 boundary.type_zp = 'cpml';
```

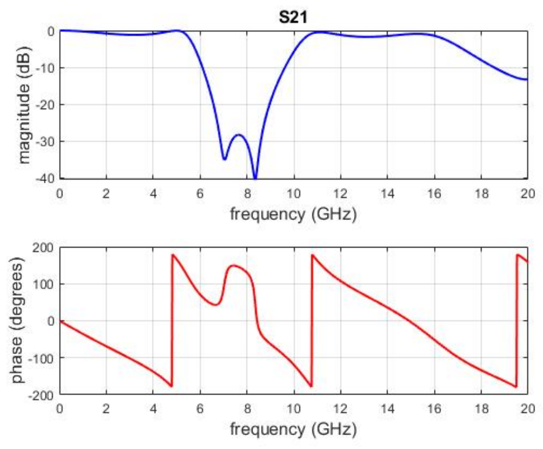
```
boundary.air_buffer_number_of_cells_zp = 10; %to match S24
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1.2;
boundary.cpml_kappa_max = 2.5;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ==<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% a dielectric
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% a dielectric
material_types(5).eps_r   = 3.2;
material_types(5).mu_r    = 1.4;
material_types(5).sigma_e = 0.5;
material_types(5).sigma_m = 0.3;
material_types(5).color   = [1 1 0];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

Figure 4.26 shows the results of the second order simulation to provide a reference to compare the fourth order simulation results to. There are a few key features to notice that are different in the fourth order simulation. As seen by examining Figure 4.26(c) and Figure 4.26(d), the sampled voltage completely dies out after about 2 ns. This is the expected response of a simulation with a short Gaussian waveform source. Another key point is that Figure 4.26(a) and Figure 4.26(b) have a smooth curve showing the S_{11} and S_{21} reflection coefficients as a function of frequency, as expected. As is also expected for a low-pass filter design, the S_{11} coefficient at low frequencies becomes very small and the S_{21} coefficient approaches 1.

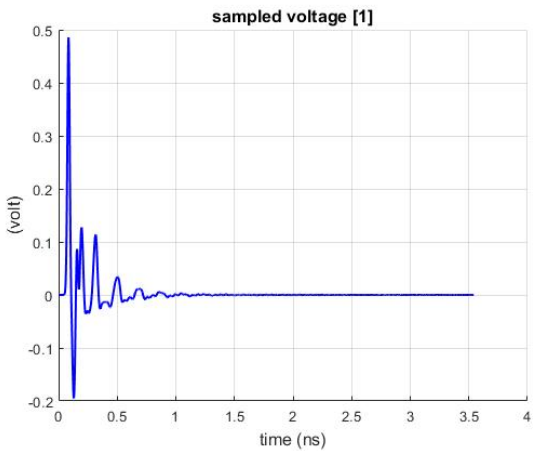
Figure 4.26 shows the results of the second order simulation to provide a reference for what the fourth order simulation results should look like.



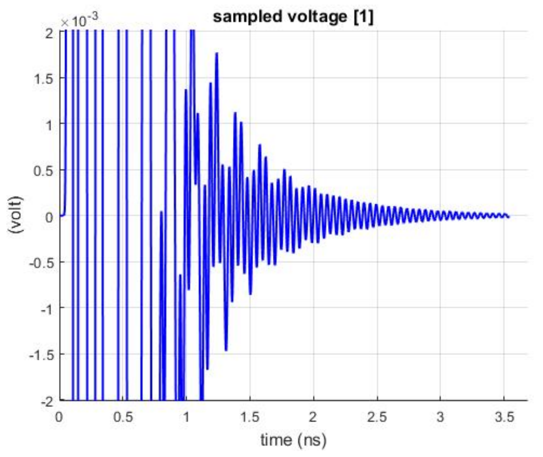
(a)



(b)



(c)



(d)

Figure 4.26 Second order filter with CPML boundaries for comparison.

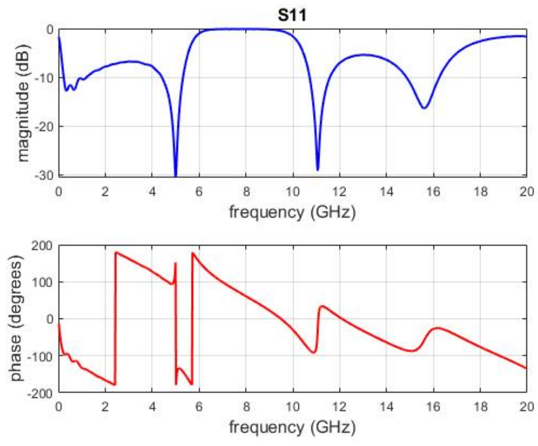
The problem space parameter code for the fourth order simulation is listed in code Listing 4.19. A few key parameters to notice are the courant_factor, the cell sizes, and the CPML boundary parameters. The courant factor has been adjusted to adhere to the stability criterion of a fourth order FDTD simulation (see section 2.2.3.3). The cells sizes are the same as the second order simulation to provide a good comparison. Finally, the CPML boundary parameters are different than the second order simulation. The κ_{max} and σ_{factor} match the optimum values for a fourth order simulation as found in section 4.4.

Listing 4.19: define problem space parameters (Fourth Order Filter)

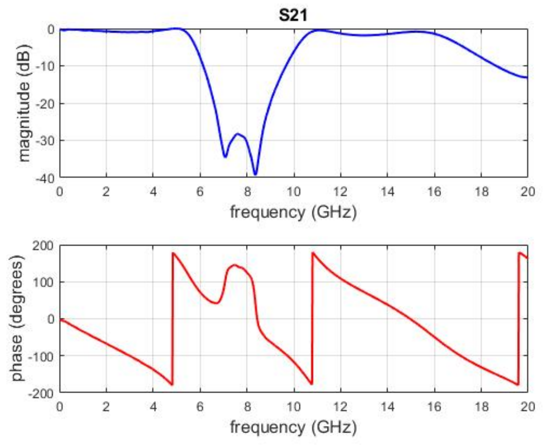
```
disp('defining the problem space parameters');

% maximum number of time steps to run FDTD simulation
number_of_time_steps = 6000; %36585; %127207; %36585 %37037 %565850

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7); %0.5; %0.1484;

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = 20;

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = 0.4064e-3;
dy = 0.4233e-3;
dz = 0.265e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor

pecTony = 0;
tonyCPMLCells = 8;

% airBufferTony = 5;
%
% boundary.type_xp = 'pec';
% boundary.air_buffer_number_of_cells_xp = airBufferTony+1; %5;
%
% boundary.type_xn = 'pec';
% boundary.air_buffer_number_of_cells_xn = airBufferTony+1; %5;
%
% boundary.type_yp = 'pec';
% boundary.air_buffer_number_of_cells_yp = airBufferTony+1; %5;
%
% boundary.type_yn = 'pec';
% boundary.air_buffer_number_of_cells_yn = airBufferTony+1; %5;
%
% boundary.type_zp = 'pec';
% boundary.air_buffer_number_of_cells_zp = airBufferTony+5+2; %10;
%
% boundary.type_zn = 'pec';
% boundary.air_buffer_number_of_cells_zn = airBufferTony-5; %3;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 5;
boundary.cpml_number_of_cells_xn = tonyCPMLCells;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 5;
boundary.cpml_number_of_cells_xp = tonyCPMLCells;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 5;
boundary.cpml_number_of_cells_yn = tonyCPMLCells;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 5;
boundary.cpml_number_of_cells_yp = tonyCPMLCells;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 0;
boundary.cpml_number_of_cells_zn = tonyCPMLCells;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = 10;
boundary.cpml_number_of_cells_zp = tonyCPMLCells;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0.6; %1.3
boundary.cpml_kappa_max = 1; %7
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;


% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% a dielectric
```

```
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% a dielectric
material_types(5).eps_r   = 3.2;
material_types(5).mu_r    = 1.4;
material_types(5).sigma_e = 0.5;
material_types(5).sigma_m = 0.3;
material_types(5).color   = [1 1 0];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

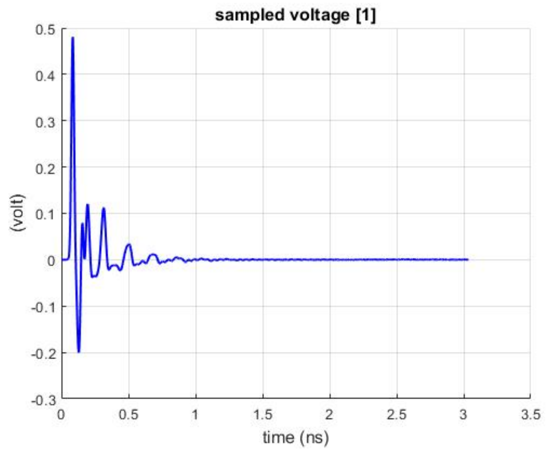
Figure 4.27 shows the results of the fourth order simulation. As shown by looking at Figure 4.27(c) and Figure 4.27(d), the sampled voltage does not zero out and in a uniform fashion. This is not expected and is likely caused by the errors created by the large mask of the fourth order updating equations interacting with the PEC plates that make up the filter. Additionally, the S_{11} and S_{21} coefficients as a function of frequency are not smooth like they should be. Finally, the results for the scattering parameters do not make physical sense. It appears that close to zero Hertz the S_{11} coefficient is close to one, which means almost all the input power is reflected back and not passed through the filter. However, the S_{21} coefficient near zero Hertz is also close to one, implying almost all of the power is allowed to pass through the filter from port one to port two. At higher frequencies the scattering parameter results seem more reasonable and are a close match to the results of the second order simulation. Further analysis into exactly what causes the low frequency errors is presented in section 4.8.



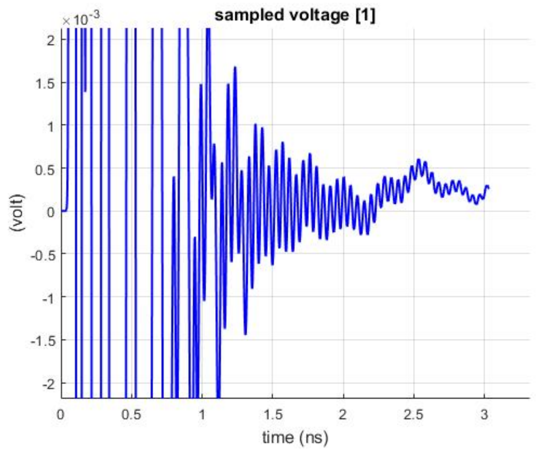
(a)



(b)



(c)

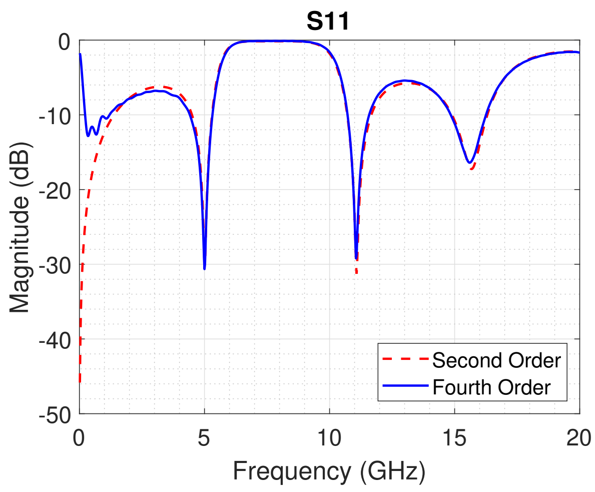


(d)

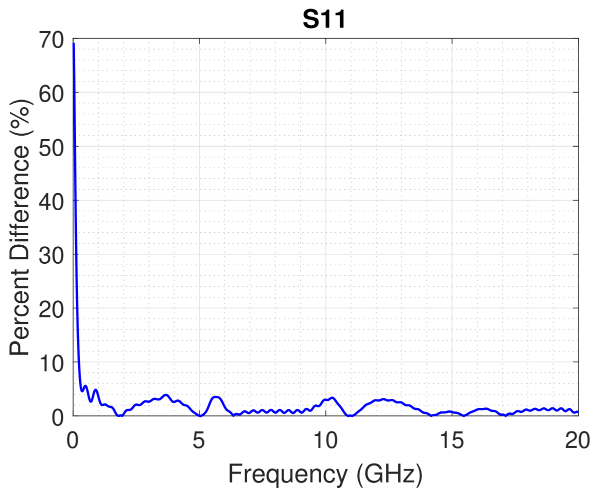
Figure 4.27 Fourth order filter simulation with CPML boundaries.

Additionally, Figure 4.28 directly compares the S_{11} parameter simulated by the second and fourth order simulations. Figure 4.29 directly compares the S_{21} parameter simulated by the second and fourth order simulations. Note that the percent difference in Figure 4.28(b) and Figure 4.29(b) is calculated by taking the absolute value of the difference between the second and fourth order simulations (in linear scale, not dB scale), dividing by the maximum of the absolute value of the second order results, and then multiplying by 100 to arrive at a percent difference value. In equation form the percent difference is calculated by:

$$\frac{\left|10^{2^{nd}_{dB}/10} - 10^{4^{th}_{dB}/10}\right|}{max(10^{2^{nd}_{dB}/10})} \times 100. \tag{4.10}$$

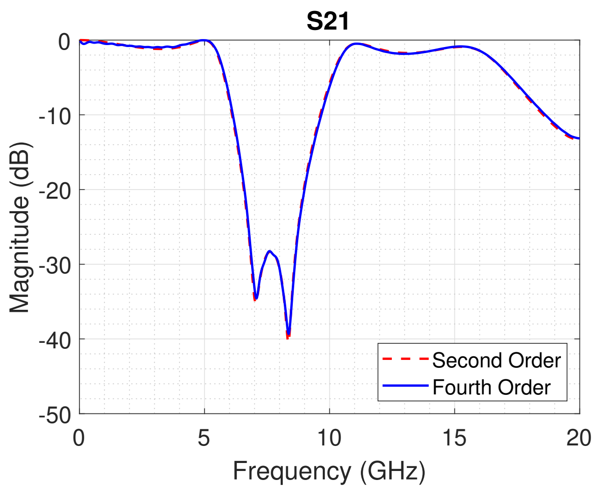


(a)

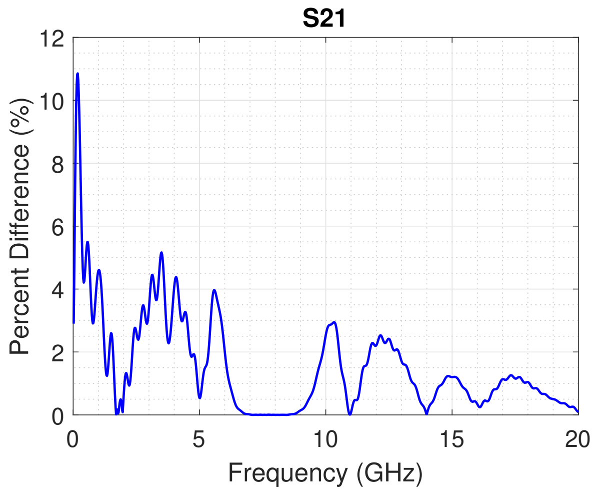


(b)

Figure 4.28 Direct comparison between the second and fourth order simulated S_{11} of a microstrip filter.



(a)



(b)

Figure 4.29 Direct comparison between the second and fourth order simulated S_{21} of a microstrip filter.

By examining Figure 4.28 and Figure 4.29 we see that the second and fourth order results are not the same. Figure 4.28 shows that the second order and fourth order results are most different at the low frequencies (between 0 and 1 GHz). For the higher frequencies (1 to 20 GHz), Figure 4.28 shows that the maximum percent difference between the second and fourth order simulations is 4%. In Figure 4.29 it is again clear the highest error is at the low frequencies (0 to 1 GHz). The percent difference decreases for the higher frequencies with a peak of 5% error around 3.5 GHz and then a maximum error of 4% from 5 to 20 GHz.

4.7.3 Discussions

Sections 4.7.1 and 4.7.2 compare the performance of the thoroughly tested and verified second order FDTD formulation [1] to the fourth order formulation presented in this thesis, in the context of PEC object simulations. In these sections it is shown the second order and fourth order simulated S-parameters do not match exactly, as shown in Figure 4.23, Figure 4.28, and Figure 4.29. These figures show that the fourth order simulation results are different than the verified second order formulation results, especially at low frequencies. At higher frequencies the difference between the second order and fourth order results is low, and the results of the fourth order simulations could be acceptable for some design applications. However, in the interest of developing the most accurate and generalized fourth order FDTD formulation, the next few sections will focus on eliminating the low frequency errors in the fourth order PEC object simulations.

4.8 Second vs fourth order updating equation treatment of PEC objects

The goal of this section is to find out exactly why the fourth order simulation does not handle PEC objects well. Once that problem is understood, different special treatments of the PEC objects in the fourth order simulation are implemented and analyzed with the goal of solving the problem.

4.8.1 Second order updating compared to fourth order updating in and around PEC objects

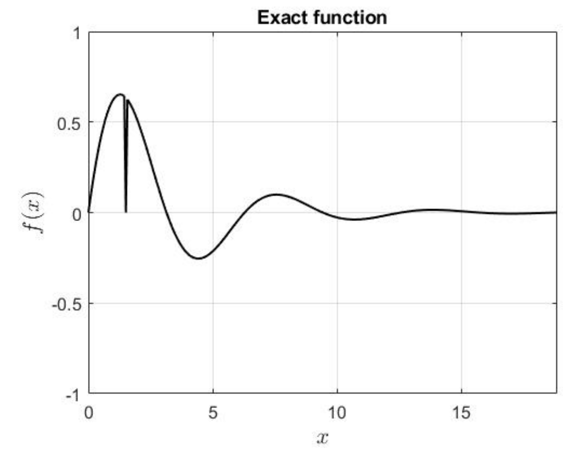
The second order updating equations have balanced derivative terms (for example, $E_y^n(i,j,k+1) - E_y^n(i,j,k)$ with coefficients of 1 and -1) [1] while the fourth order updating equations have unbalanced derivative terms (for example, $-E_z^n(i,j+2,k) + 27E_z^n(i,j+1,k) - 27E_z^n(i,j,k) + E_z^n(i,j-1,k)$ with coefficients of -1, 27, -27, and 1). This unbalance in the fourth order updating equations is problematic when modeling the edges of PEC objects because one or more of the terms is zero.

In order to better understand the effects of this unbalance, a simple discontinuous function derivative was approximated using second order accurate central differencing and fourth order accurate central differencing. See equation 2.33 for the fourth order accurate updating equation and see equation 2.6 for the second order accurate updating equation.

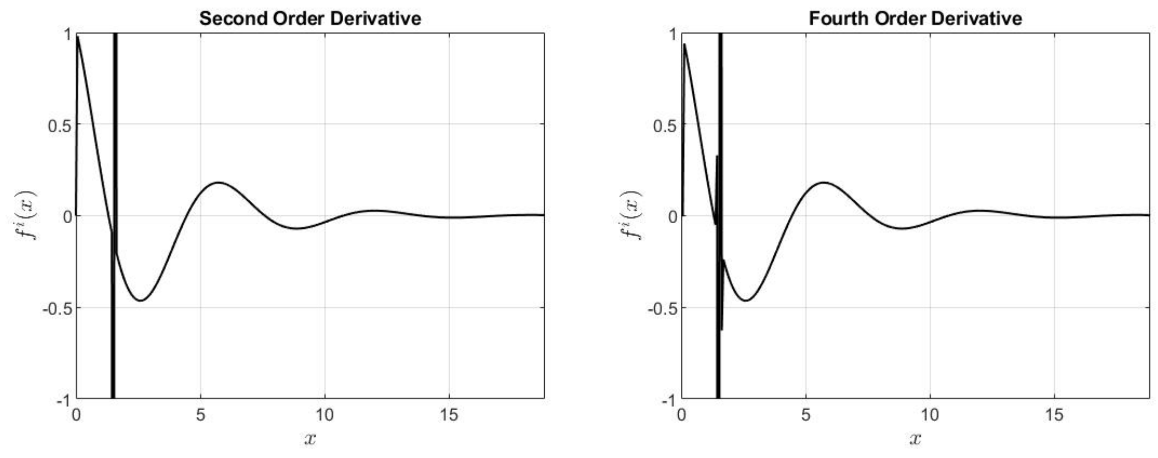
The equation of the function is shown in equation 4.11:

$$f(x) = \sin(x)e^{-0.3x} \tag{4.11}$$

Where the function is artificially set to zero at a single point (see Figure 4.30(a)) or a range of points (see Figure 4.31(a)) to simulate a PEC plate and a PEC object respectively.



(a)



(b)

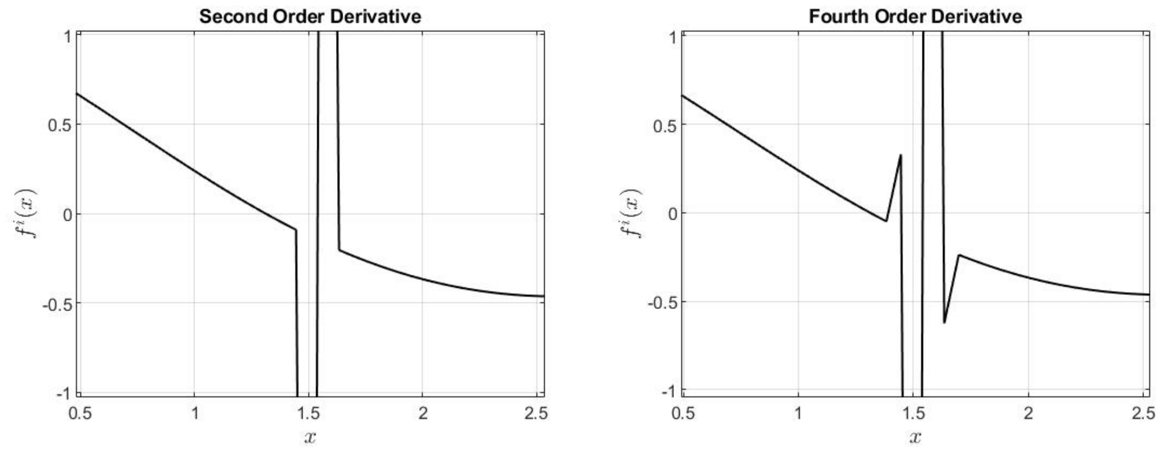
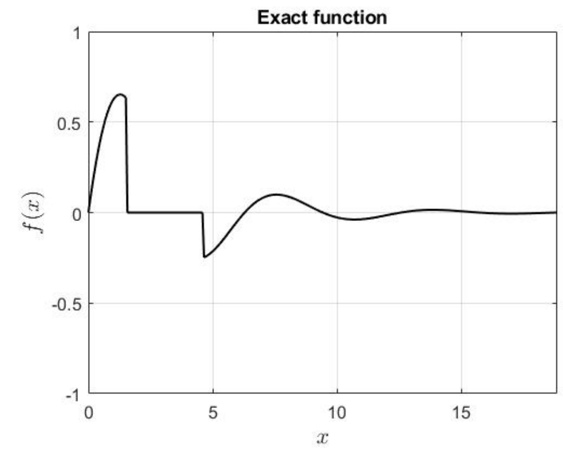
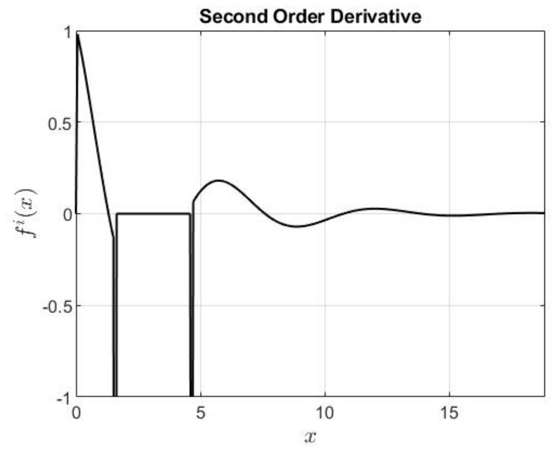


Figure 4.30 Example of second and fourth order derivatives on a function with a discontinuity simulating a PEC plate.

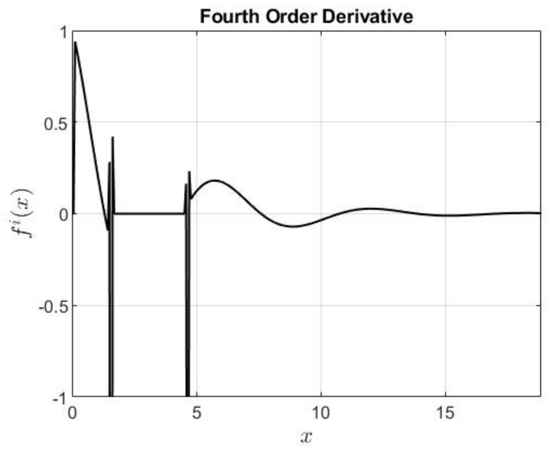
The same function in equation 4.11 was used to simulate a PEC object by having a larger region of zero value.



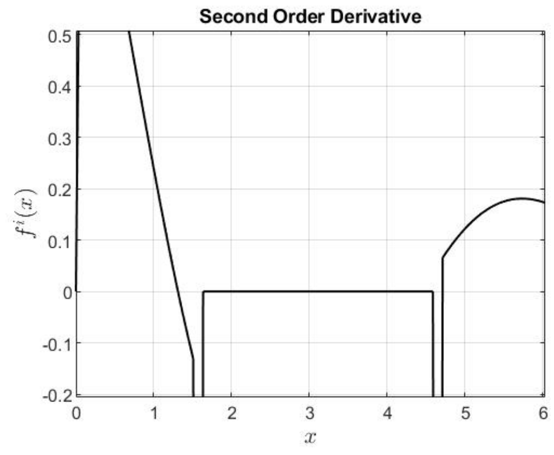
(a)



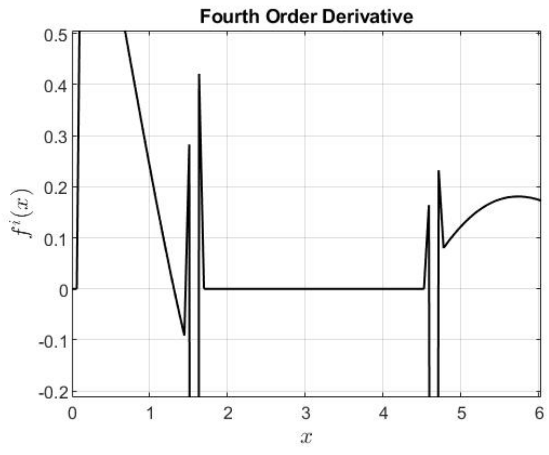
(b)



(c)



(d)



(e)

Figure 4.31 Example of second and fourth order derivatives on a function with a discontinuity simulating a PEC object.

Figure 4.31(d) and Figure 4.31(e) are zoomed in portions of Figure 4.31(b) and Figure 4.31(c) respectively.

By comparing Figure 4.30 and Figure 4.31 it is clear the fourth order derivative approximation has errors around the PEC plate and the edges of the PEC object. For the PEC plate simulation, in Figure 4.30, the discontinuity should have a negative derivative followed immediately by a positive derivative since the slope goes down and then immediately back up. This result is produced correctly with second order approximations and is shown in Figure 4.30(b). The fourth order derivative approximation does not exibit the expected negative then positive derivative prediction. It has one point of a slight positive derivative before going negative and one point of a negative derivative after going positive. This behavior is likely due to the unbalance of the fourth order derivative approximation terms mentioned at the beginning of this section. For the PEC object simulation, the second order derivative approximation (Figure 4.31(b)) is the correct derivative approximation based on the definition of a derivative. The fourth order derivative approximation (Figure 4.31(c)) does not have the expected behavior because it goes positive at the edges of the object when the derivative at those regions is negative.

4.8.2 Investigation into EM Field Distributions

In this section, figures showing the field magnitudes of different simulations that involve PEC object are presented. The goal of this section is visualizing the discontinuities in the fields and to compare the second and fourth order simulations.

4.8.2.1 Filter Problem EM Field Distributions

Figure 4.32 and Figure 4.33 show the Ex, Ey, Ez, Hx, Hy, and Hz components (left to right top to bottom in the figures) at the level of the top layer of the filter in a plane parallel to the XY plane. The voltage source is a Gaussian waveform, and the boundaries are CPML. Both the second and the fourth order simulations are run for 2000 time steps.

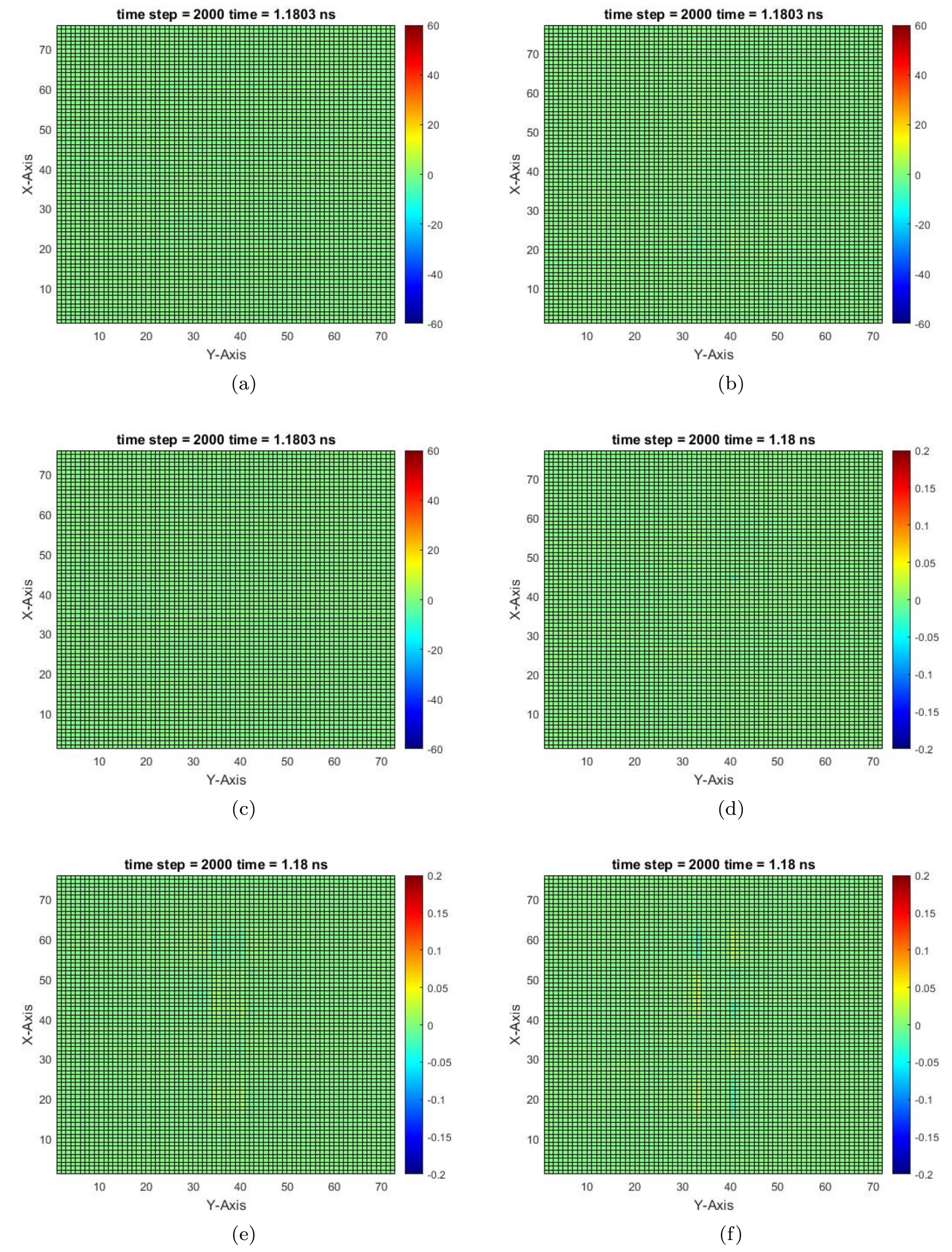


Figure 4.32 Final frames of the second order simulation.

Figure 4.32 shows that after 2000 time steps all field components in the plane cut have zeroed out. This is the expected result as simulation should zero out after an adequate number of time steps. As shown in Figure 4.26(c), 2000 time steps (or 1.18 ns) is enough time for the sampled voltage to zero out and thus all other fields as well.

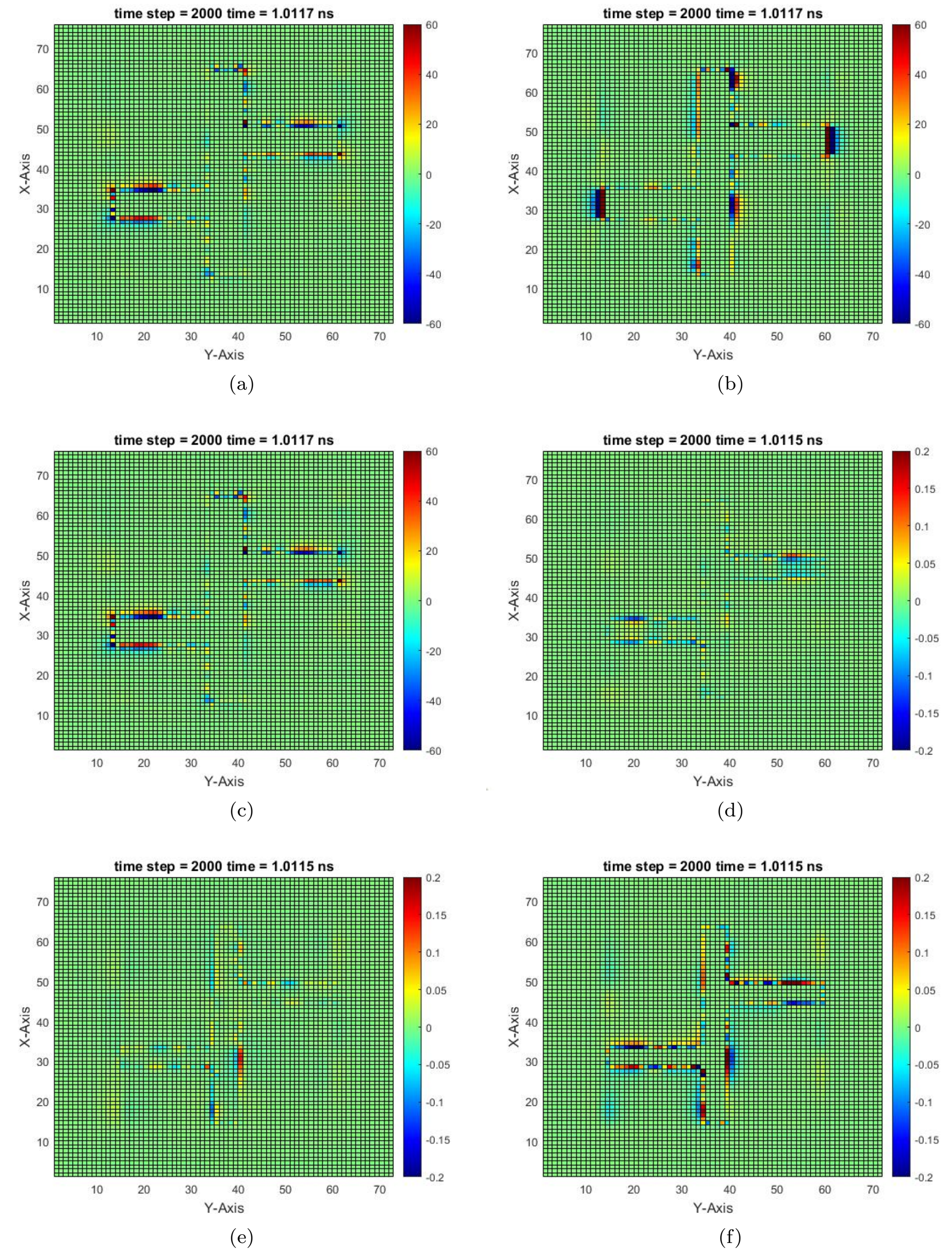
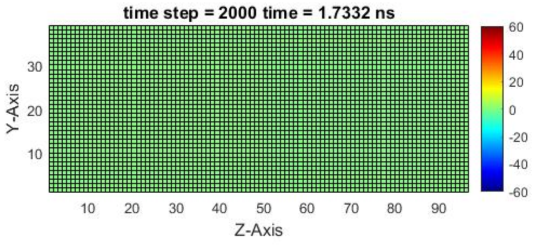


Figure 4.33 Final frames of the fourth order simulation.

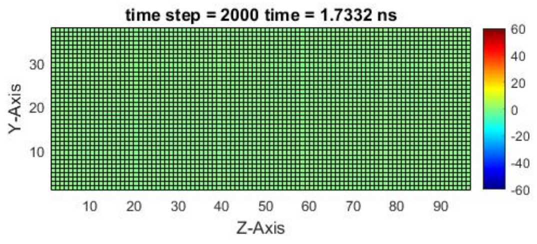
Figure 4.33 shows that after 2000 time steps all field components in the plane cut have not zeroed out. This is not the expected result as the simulation should zero out after an adequate number of time steps. As shown in Figure 4.26(c), 2000 time steps (or 1.18 ns) is enough time for the sampled voltage of a correct simulation to zero out, as well as all other fields. Since the fourth order simulation has some non zero fields after 2000 time steps, the fourth order updating equations are flawed when handling PEC plates.

4.8.2.2 Brick Dipole Problem EM-Fields Videos

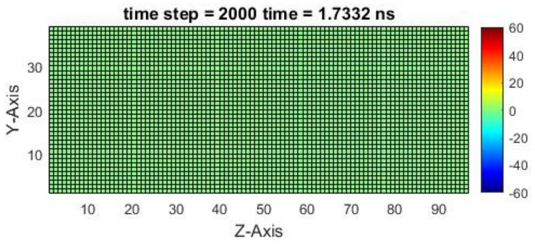
Figure 4.34 and Figure 4.35 show the Ex, Ey, Ez, Hx, Hy, and Hz components (left to right top to bottom in the figure) at the level of the top layer of the brick dipole in a plane parallel to the YZ plane. The simulation setup matches that as described in section 4.7.1.1 exactly except the simulation is only run for 2000 time steps in this section.



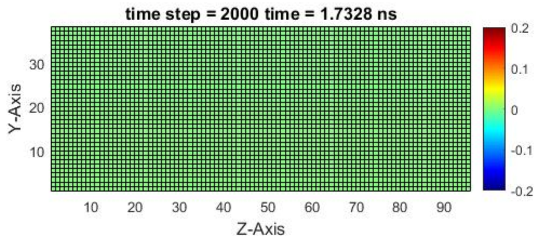
(a)



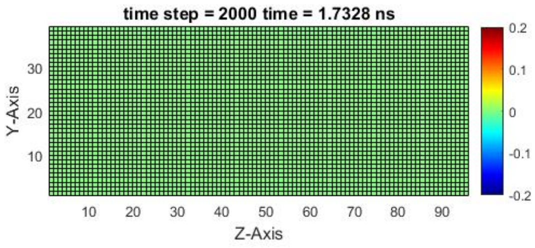
(b)



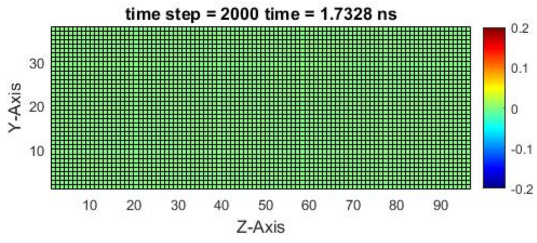
(c)



(d)



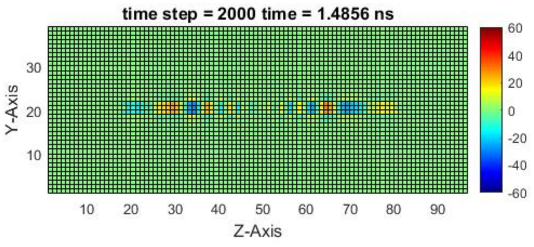
(e)



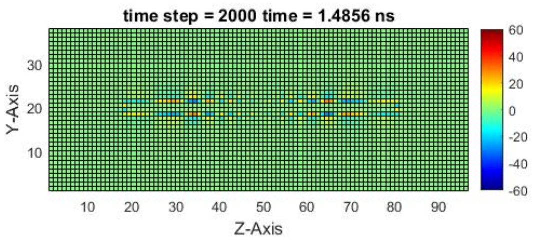
(f)

Figure 4.34 Final frames of the second order simulation.

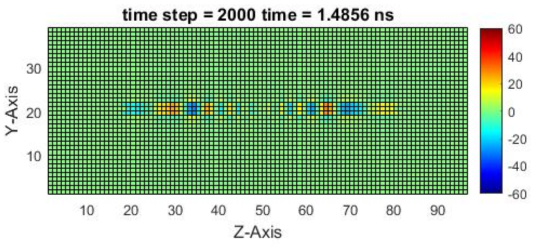
Figure 4.34 shows that after 2000 time steps all field components in the plane cut have zeroed out. This is the expected result as a simulation should zero out after an adequate number of time steps. As shown in Figure 4.21(b), 2000 time steps (or 1.18 ns) is enough time for the sampled voltage to zero out and thus all other fields as well.



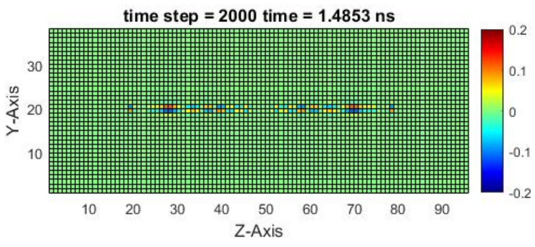
(a)



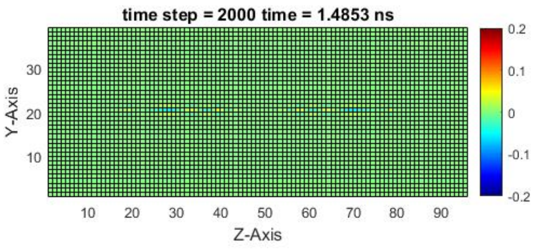
(b)



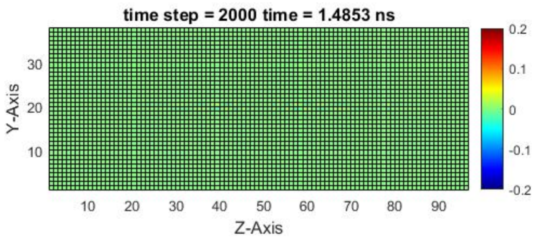
(c)



(d)



(e)



(f)

Figure 4.35 Final frames of the fourth order simulation.

Figure 4.35 shows that after 2000 time steps all field components in the plane cut have not zeroed out. This is not the expected result as the simulation should zero out after an adequate number of time steps. As shown in Figure 4.21(b), 2000 time steps (or 1.18 ns) is enough time for the sampled voltage to zero out and thus all other fields as well, ultimately meaning the fourth order updating equations are flawed when handling PEC objects.

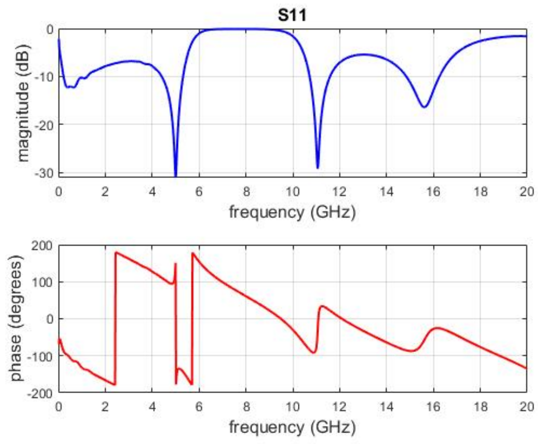
4.8.3 Other Voltage Source Waveforms

The goal of this section is to simulate voltage source waveforms without a DC component to see if that solves the low frequency error seen in the fourth order simulations with PEC objects.

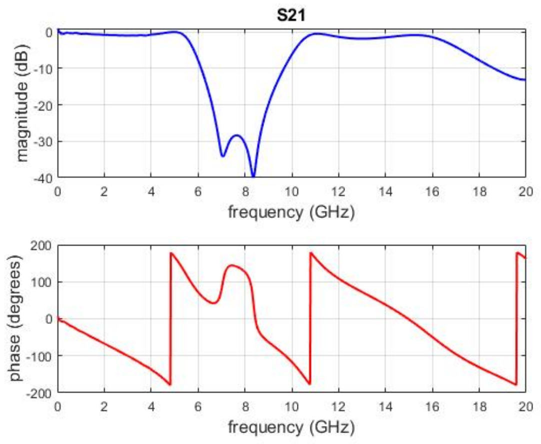
4.8.3.1 Derivative of a Gaussian Voltage Source

The simulation set up is identical to that of the filter problem simulated in section 4.7.2.1 except for the voltage source waveform.

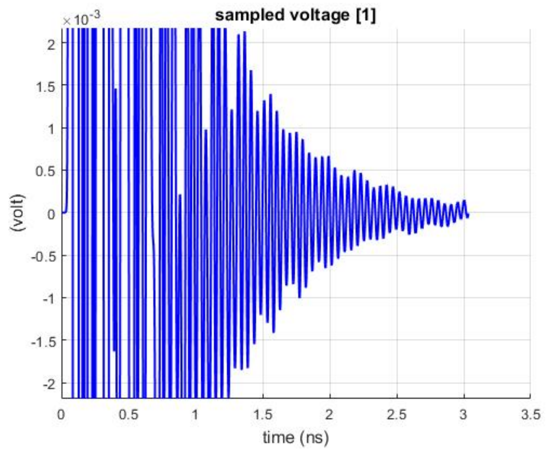
In this section the basic fourth order filter formulation (no PEC special treatment) with a derivative of a Gaussian source voltage waveform is simulated. This new waveform does not have a zero frequency component, and could therefore not produce the low frequency error produced by the previous fourth order filter simulations. Figure 4.36 shows the results for this simulation.



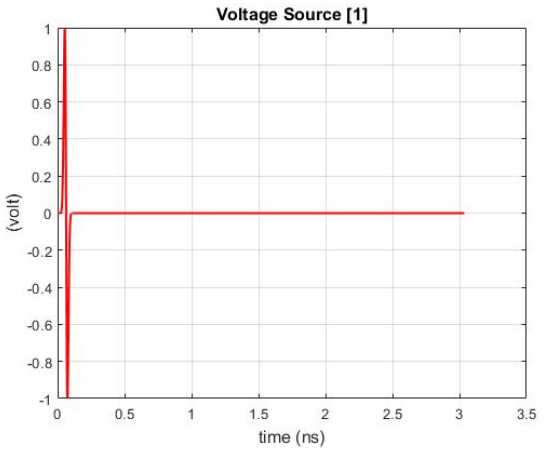
(a)



(b)



(c)



(d)

Figure 4.36 Derivative of a Gaussian voltage source used on the fourth order filter simulation with 6000 time steps.

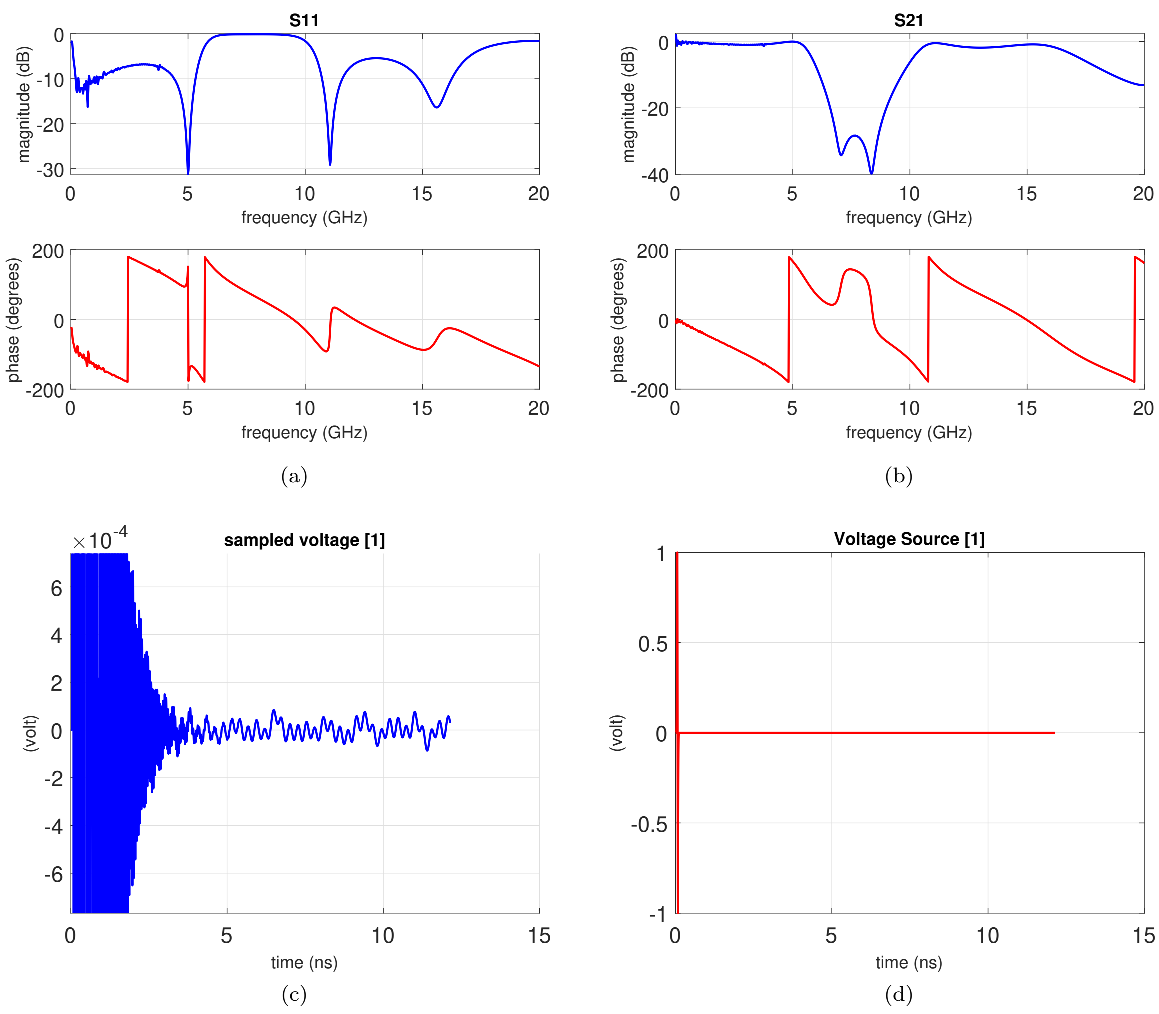


Figure 4.37 Derivative of a Gaussian voltage source used on the fourth order filter simulation with 24000 time steps.

As seen in Figure 4.36, using a derivative of a Gaussian voltage source waveform did not get rid of the low frequency error. However, by comparing Figure 4.27(c) and Figure 4.36(c), the sampled voltage died out more evenly when using a derivative of a Gaussian source rather than using a simple Gaussian source. However, if the simulation time in increased (as shown in Figure 4.37, the voltage source clearly does not continue to die out. Figure 4.37(c) shows how the sampled voltage continous to oscilate at a steady magnitude after roughly 4 ns.

4.8.3.2 Cosine Modulated Gaussian Voltage Source

The simulation set up is identical to that of the filter problem simulated in section 4.7.2.1, except for the voltage source waveform.

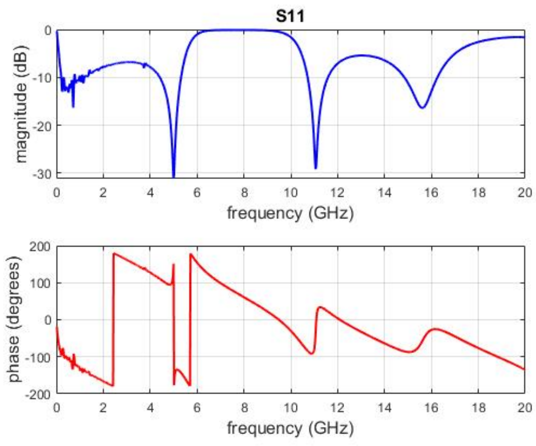
In this section the basic fourth order filter formulation (no PEC special treatment) with a cosine modulated Gaussian source voltage waveform is simulated. The idea here is that this new waveform does not have a zero frequency component and could therefore not produce the low frequency error produced by other fourth order filter simulations.

The simulated source voltage waveform has a bandwidth of 1GHz and a modulation frequency of 6GHz. Code Listing 4.20 shows additional details.

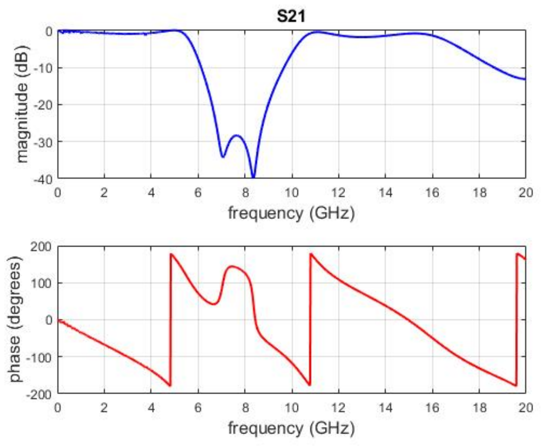
Listing 4.20: define sources and lumped elements (Cosine Modulated Guassian)

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 0;
waveforms.derivative_gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.sinusoidal(1).frequency = 3e9;
waveforms.cosine_modulated_gaussian(1).bandwidth = 1e9;
waveforms.cosine_modulated_gaussian(1).modulation_frequency = 6e9;

% voltage sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude   : volts
voltage_sources(1).min_x = 15*dx;
voltage_sources(1).min_y = 1*dy;
voltage_sources(1).min_z = 1*dz;
voltage_sources(1).max_x = 21*dx;
voltage_sources(1).max_y = 1*dy;
voltage_sources(1).max_z = 4*dz;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'cosine_modulated_gaussian'; %'gaussian';
voltage_sources(1).waveform_index = 1;

% resistors
% direction: 'x', 'y', or 'z'
% resistance : ohms
resistors(1).min_x = 31*dx;
resistors(1).min_y = 47*dy;
resistors(1).min_z = 1*dz;
resistors(1).max_x = 37*dx;
resistors(1).max_y = 47*dy;
resistors(1).max_z = 4*dz;
resistors(1).direction = 'z';
resistors(1).resistance = 50;
```

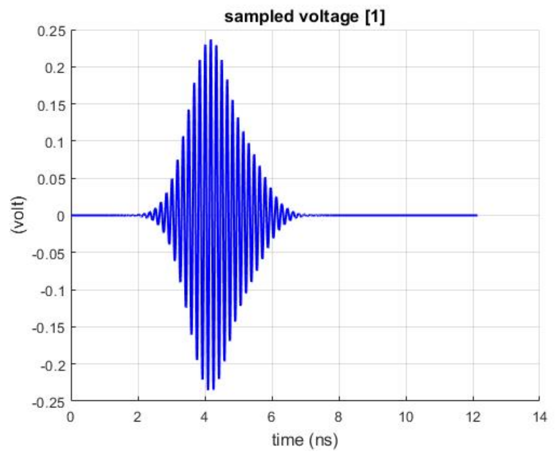
Figure 4.36 shows the results for this simulation:



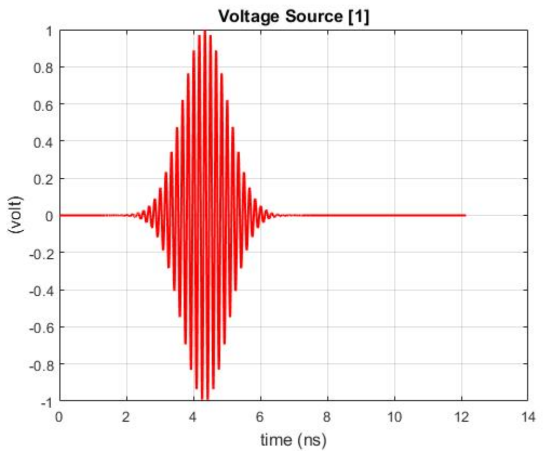
(a)



(b)



(c)



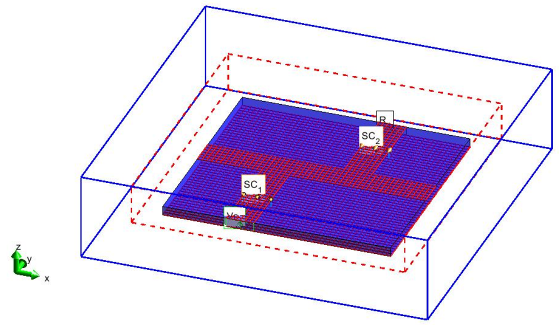
(d)

Figure 4.38 Cosine modulated Gaussian voltage source used on the fourth order filter problem.

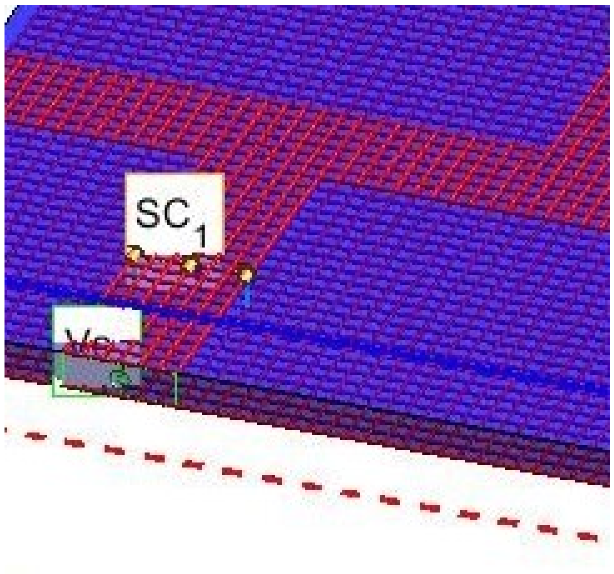
As shown in Figure 4.38, using a cosine modulated Gaussian voltage source waveform did not completely eliminate the low frequency error.

4.8.4 PEC Plates Created using a Thin Wire Mesh

In this section the PEC plates for the filter problem are created with a mesh of thin wires. As is clear in section 4.6 and section 5.1, the thin wire formulation for the fourth order simulation works. The goal of this section is to use the thin wire formulation shown to work to create a PEC plate geometry rather than using the standard PEC plate formulation. The simulation setup matches that of section 4.7.2.1 except that the PEC plates are made from a thin wire mesh. Figure 4.39 shows the miscrostrip low pass filter built from a thin wire mesh.



(a)



(b)

Figure 4.39 A diagram showing how the filter was built out of a thin wire mesh.

Code Listing 4.21 shows how the PEC plates are created out of thin wires. The thin wire formulation presented in section 2.2.4 is used to update the fields around the thin wires. Note section 2.2.4 only shows the fourth order formulation for a Z-direction oriented thin wire. This code uses similar code to impliment thin wires in the X- and Y-directions.

Listing 4.21: create PEC plates (Thin Wire Mesh)

```
disp('creating PEC plates on the material grid');

thin_wires = [];
counterThinWires = 0;

for ind = 1:number_of_bricks

    %mtype = bricks(ind).material_type;
    %sigma_pec = material_types(mtype).sigma_e;

    % convert coordinates to node indices on the FDTD grid
    blx = round((bricks(ind).min_x - fdtd_domain.min_x)/dx-12);
    bly = round((bricks(ind).min_y - fdtd_domain.min_y)/dy-12);
    blz = round((bricks(ind).min_z - fdtd_domain.min_z)/dz-7);

    bux = round((bricks(ind).max_x - fdtd_domain.min_x)/dx-12);
    buy = round((bricks(ind).max_y - fdtd_domain.min_y)/dy-12);
    buz = round((bricks(ind).max_z - fdtd_domain.min_z)/dz-7);

%     % find the zero thickness bricks
%      if (blx == bux)
%          sigma_e_y(blx, bly:buy-1,blz:buz) = sigma_pec;
%           sigma_e_z(blx,  bly:buy,blz:buz-1) = sigma_pec;
%       end
%       if (bly == buy)
%           sigma_e_z(blx:bux,  bly,  blz:buz-1) = sigma_pec;
%           sigma_e_x(blx:bux-1,  bly,  blz:buz) = sigma_pec;
%       end
%       if (blz == buz)
%           sigma_e_x(blx:bux-1,bly:buy,  blz) = sigma_pec;
%           sigma_e_y(blx:bux,bly:buy-1,blz)  = sigma_pec;
%       end

%       if (blx == bux)
%
%       end
%       if (bly == buy)
%
%       end
    if (blz == buz)
        for thinWire_i = bly:buy
            counterThinWires = counterThinWires+1;
            thin_wires(counterThinWires).min_x = blx*dx;
            thin_wires(counterThinWires).min_y = thinWire_i*dy;
            thin_wires(counterThinWires).min_z = blz*dz;
            thin_wires(counterThinWires).max_x = (bux)*dx;
            thin_wires(counterThinWires).max_y = thinWire_i*dy;
            thin_wires(counterThinWires).max_z = blz*dz;
            thin_wires(counterThinWires).radius = 0.05e-3;
            thin_wires(counterThinWires).direction = 'x';
        end
        for thinWire_i = blx:bux
            counterThinWires = counterThinWires+1;
            thin_wires(counterThinWires).min_x = thinWire_i*dx;
            thin_wires(counterThinWires).min_y = bly*dy;
            thin_wires(counterThinWires).min_z = blz*dz;
            thin_wires(counterThinWires).max_x = thinWire_i*dx;
            thin_wires(counterThinWires).max_y = (buy)*dy;
            thin_wires(counterThinWires).max_z = blz*dz;
            thin_wires(counterThinWires).radius = 0.05e-3;
            thin_wires(counterThinWires).direction = 'y';
        end
    end
end
```

Figure 4.40 shows the results of creating the PEC plates of the filter problem out of a mesh of thin wires.

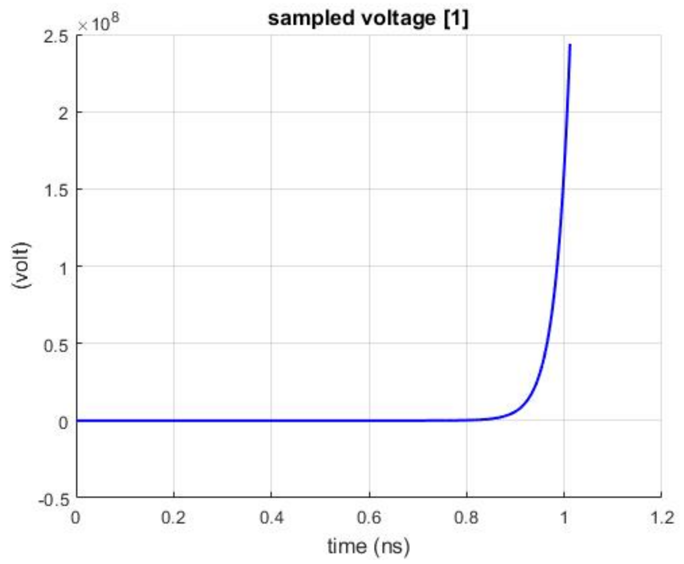


Figure 4.40 Sampled voltage results from building the filter problem out of thin wires.

As is clear from Figure 4.40, creating the PEC plates for the filter problem did not solve the low frequency error. In fact, the simulation diverges, showing that there is a major fundamental issue with trying to simulate a PEC plate as a mesh of thin wires. The thin wire formulation presented in **??** and the adapted fourth order thin wire formulation presented in this thesis is not derived or tested for thin wires that cross each other. As the microstrip lines used to build this filter are made of a mesh of thin wires, it is not surprising the simulation diverged. In order to further examine this type of simulation, a crossed wire approximation needs to be developed.

4.8.5 The use of a large ϵ_r for PEC plates

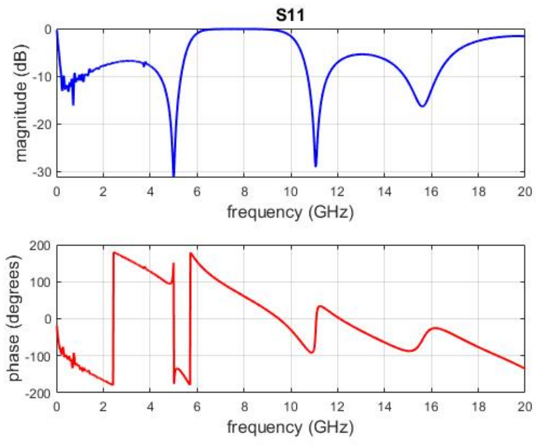
The goal of this section is to see how fourth order simulations handle PEC plates with a large ϵ_r in addition to the excepted large conductivity σ^e. The idea behind this is to remove the low frequency error by using an incredibly large relative permativity to characterize the PEC plates. Lines 28 and 30 of code Listing 4.22 shows that the relative permativity inside the PEC plate is set to the same value as the conductivity of the PEC plate. The conductivity of the PEC plate is generally very large, on the order of 10^{10}.

Listing 4.22: create PEC plates (Large ϵ_r)

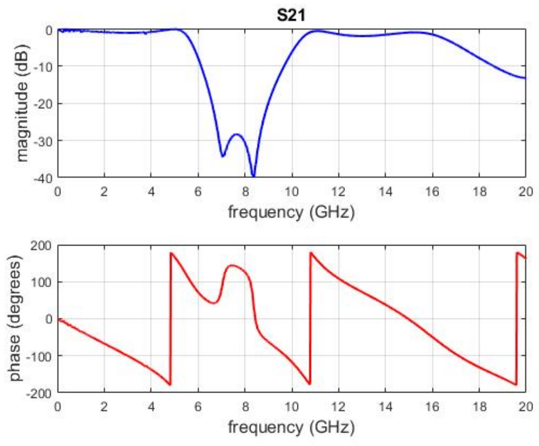
```
4.1 disp('creating PEC plates on the material grid');
4.2
4.3 for ind = 1:number_of_bricks
4.4
4.5     mtype = bricks(ind).material_type;
```

```
    sigma_pec = material_types(mtype).sigma_e;

    % convert coordinates to node indices on the FDTD grid
    blx = round((bricks(ind).min_x - fdtd_domain.min_x)/dx)+1;
    bly = round((bricks(ind).min_y - fdtd_domain.min_y)/dy)+1;
    blz = round((bricks(ind).min_z - fdtd_domain.min_z)/dz)+1;

    bux = round((bricks(ind).max_x - fdtd_domain.min_x)/dx)+1;
    buy = round((bricks(ind).max_y - fdtd_domain.min_y)/dy)+1;
    buz = round((bricks(ind).max_z - fdtd_domain.min_z)/dz)+1;

    % find the zero thickness bricks
    if (blx == bux)
        sigma_e_y(blx, bly:buy-1,blz:buz) = sigma_pec;
        sigma_e_z(blx, bly:buy,blz:buz-1) = sigma_pec;
    end
    if (bly == buy)
        sigma_e_z(blx:bux, bly, blz:buz-1) = sigma_pec;
        sigma_e_x(blx:bux-1, bly, blz:buz) = sigma_pec;
    end
    if (blz == buz)
        sigma_e_x(blx:bux-1,bly:buy, blz) = sigma_pec;
        eps_r_x(blx:bux-1,bly:buy, blz) = sigma_pec;
        sigma_e_y(blx:bux,bly:buy-1,blz)  = sigma_pec;
        eps_r_y(blx:bux,bly:buy-1,blz)  = sigma_pec;
    end
end
```

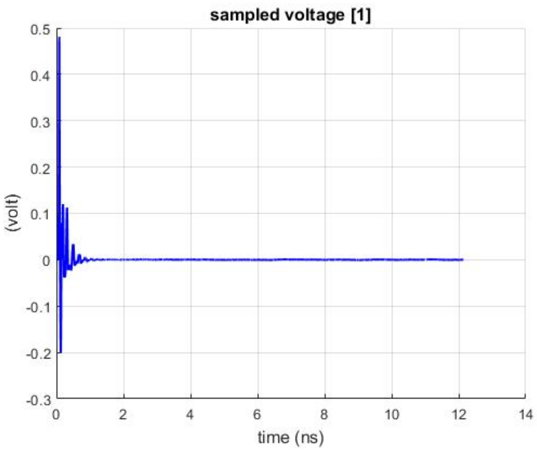
Figure 4.41 shows the results of using a large permitivity in the region of the PEC plates.



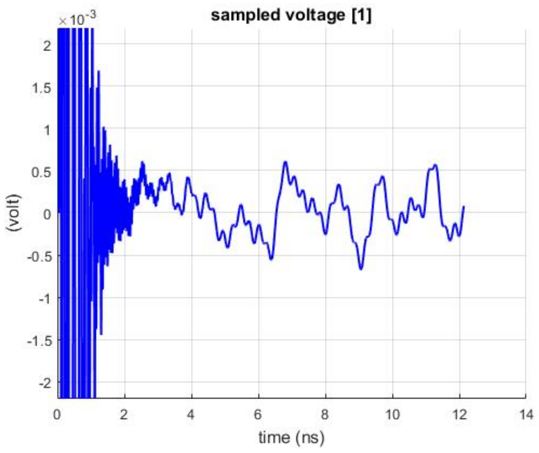
(a)



(b)



(c)



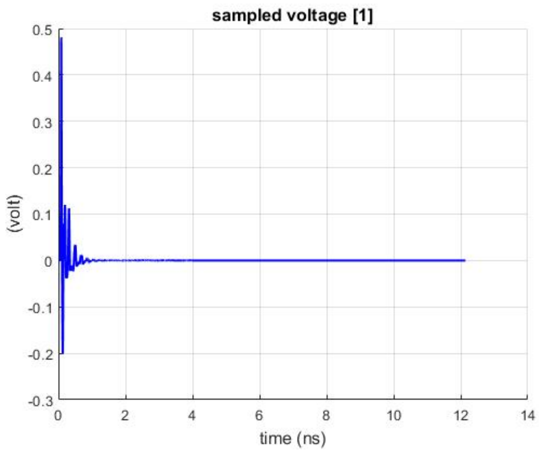
(d)

Figure 4.41 Results from using a large ϵ_r to help define PEC plate.

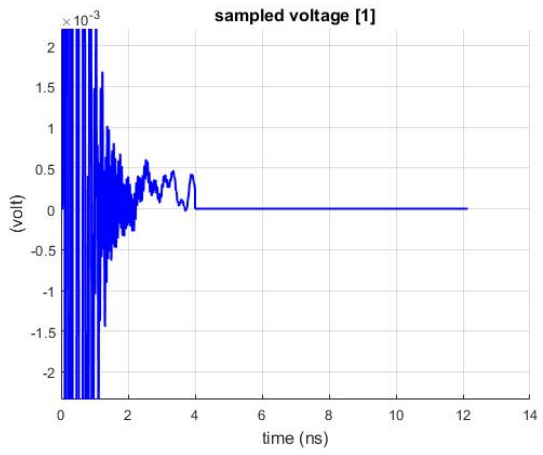
As seen from Figure 4.41, there is almost zero effect from setting ϵ_r to a very large value inside the PEC rather than 1. The low frequency error is still present and therefore using a large permativity with a large conductivity inside the PEC plates must not be considered a solution to fourth order simulations correctly simulating PEC materials.

4.8.6 Zero out the Sampled Voltage/Current for the Filter Problem

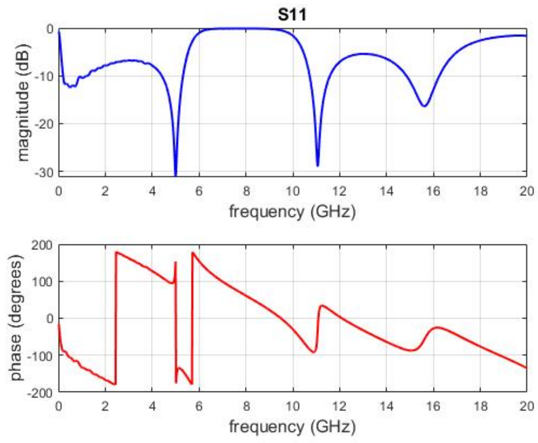
In this section the sampled voltage and current are artificially forced to zero after a certain amount of time. The scattering parameters are then calculated from the artificially modified sampled voltages and currents. The goal here is to see if the low frequency errors are coming from an effect that only occurs after a long period of time or if the error is there for the entire length of the simulation.



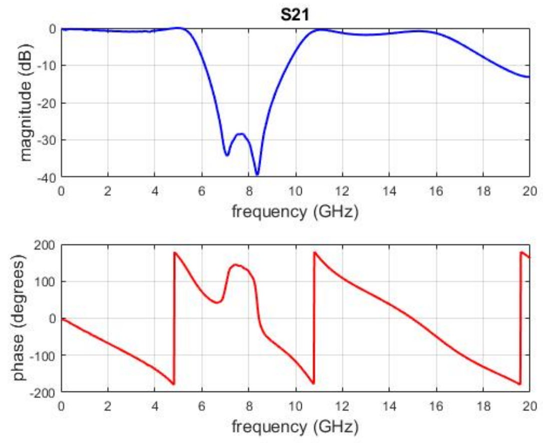
(a)



(b)



(c)



(d)

Figure 4.42 Results from zeroing out the sampled voltage and current after 4 ns.

As evident from Figure 4.42, the low frequency error appears to be there for the entire length of the simulation in the time domain, not just after the sampled voltage should have settled out. However, it appears setting the sampled voltage and current to zero after a given amount of time decreases the low frequency error, but it does not remove it completely.

4.8.7 Coefficient Method for PEC Special Treatment

The goal of this section is to attempt a special treatment for the fourth order updating equations to make them properly simulate PEC objects. The simplest special treatment of these fourth order updating equations is to transform them to second order updating equations. This general technique is used in [29] and [23]. Section 4.8.1 shows that the unbalanced fourth order updating equations incorrectly handle PEC interfaces due to an abrupt change in the field values. If most of the domain is updated using the fourth order updating equations and only the field components directly around the PEC interfaces are updated using second order, then the simulation should retain the fourth order accuracy and correctly simulate the PEC objects.

4.8.7.1 Formulation

The idea of this formulation is to use coefficients to handle the transition from regular fourth order FDTD to the special treatment around the PEC objects. In order to understand this method, the fourth order updating equations need to be examined. For reference, equation 4.12 is the analytically derived fourth order E_x updating equation.

$$\begin{aligned} E_x^{n+1}(i,j,k) = & \frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \frac{-H_z^{n+\frac{1}{2}}(i,j+1,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i,j-1,k)+H_z^{n+\frac{1}{2}}(i,j-2,k)}{24\Delta y} \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} \frac{-H_y^{n+\frac{1}{2}}(i,j,k+1)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i,j,k-1)+H_y^{n+\frac{1}{2}}(i,j,k-2)}{24\Delta z} \\ & +\frac{2\epsilon_x(i,j,k)-\Delta t\sigma_x^e(i,j,k)}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} E_x^n(i,j,k) \\ & -\frac{2\Delta t}{2\epsilon_x(i,j,k)+\Delta t\sigma_x^e(i,j,k)} J_x^{n+\frac{1}{2}}(i,j,k) \end{aligned} \tag{4.12}$$

For use in the code, this equation gets simplified in two ways. One, the coefficients are calculated in a separate part of the code and assigned new variables. Two, J_x is assumed to be zero. If there is a current source, it will be handled with the coefficients. Equation 4.13 shows the results of re-writing equation 4.12 in the form of the MATLAB code.

$$\begin{aligned} E_x^{n+1}(i,j,k) = & Cexhz(i,j,k) \frac{-H_z^{n+\frac{1}{2}}(i,j+1,k)+27H_z^{n+\frac{1}{2}}(i,j,k)-27H_z^{n+\frac{1}{2}}(i,j-1,k)+H_z^{n+\frac{1}{2}}(i,j-2,k)}{24} \\ & +Cexhy(i,j,k) \frac{-H_y^{n+\frac{1}{2}}(i,j,k+1)+27H_y^{n+\frac{1}{2}}(i,j,k)-27H_y^{n+\frac{1}{2}}(i,j,k-1)+H_y^{n+\frac{1}{2}}(i,j,k-2)}{24} \\ & +Cexe(i,j,k)\cdot E_x^n(i,j,k) \end{aligned} \tag{4.13}$$

For the following method of special treatment around PEC objects, the coefficients Cexhy, Cexhz, and Cexe do not change. The new coefficients will be applied to the derivative terms only.

One way to handle the PEC boundaries while using fourth order updating equations is to simply convert to second order updating equations for the field components near the boundary. This conversion is done by initializing a new set of coefficients to avoid using “IF” statements in the electric/magnetic field components during each time step. Avoiding “IF” statements in that stage keeps the code running fast.

Equation 4.14 shows an example of these coefficients placed in the E_x updating equation (equation 4.13). The field indices are omitted for clarity but the indices of the new coefficients (of the form

"S24_cx_xxxx") match that of the field component being updated. The other updating equations have the same form as equation 4.14 and details can be found in code Listing 4.23 and Listing 4.24.

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & Cexhz \frac{-S24\_c1\_exhz * H_z^{n+\frac{1}{2}} + 27H_z^{n+\frac{1}{2}} - 27H_z^{n+\frac{1}{2}} + S24\_c1\_exhz * H_z^{n+\frac{1}{2}}}{S24\_c2\_exhz * 24} \\ & + Cexhy \frac{-S24\_c1\_exhy * H_y^{n+\frac{1}{2}} + 27H_y^{n+\frac{1}{2}} - 27H_y^{n+\frac{1}{2}} + S24\_c1\_exhy * H_y^{n+\frac{1}{2}}}{S24\_c2\_exhy * 24} \\ & + Cexe \cdot E_x^n \end{aligned} \tag{4.14}$$

The second order E_x updating equation has the following form, see section 2.2.1 for the derivation:

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & Cexhz \left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k) \right) \\ & + Cexhy \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1) \right) \\ & + Cexe \cdot E_x^n \end{aligned} \tag{4.15}$$

In order to transform from fourth order updating (equation 4.14) to second order updating (equation 4.15), the coefficient S24_c1_exhz and S24_c1_exhy are set to zero and the coefficients S24_c2_exhz and S24_c2_exhy are set to 27/24 to cancel out the 27 in the numerator and the 24 in the denominator.

Implementation of this fourth order to second order updating equation transformation using coefficients is relatively simple. After the problem space is initialized by assigning values to the material arrays a single loop over the entire domain can set the values for the fourth to second order transformation coefficients. Since fourth order has been shown to not update correctly around PEC boundaries, the material conductance array ($\sigma(i,j,k)$) will determine when the fourth order updating equations need to be converted to second order updating equations. Whenever adjacent conductance values change discontinuously, or in other words when their difference is very large, the fourth to second order transformation coefficients are set to 0 and 27/24 as applicable to make the updating equations second order. Otherwise, the fourth to second order transformation coefficients are all set to one keeping the updating equations fourth order.

As shown in Figure 2.14, each magnetic field uses electric field components in two perpendicular directions to be updated. For example, E_z components in the Y direction and E_y components in the Z direction are used to update H_x. In the case of fourth order updating, both directions will use four electric field components. In the case of a magnetic field component near a PEC boundary, only the direction of electric field components normal to the boundary will need to be converted to second order and use two electric field components. The direction parallel to the boundary can still use all four electric field components and be considered fourth order. From here on forward this type of updating will be refereed to as mixed second/fourth order updating. Figure 4.43, Figure 4.44, and Figure 4.45 detail further. Additionally, mixed second/fourth order updating only needs to be applied to electric fields that are parallel to a PEC boundary. Electric fields that are normal to the PEC boundary can use pure fourth order updating since neither direction of sampled magnetic field components cross the PEC boundary.

Figure 4.43 and Figure 4.44 show how the fourth order updating equations are modified when updating the tangential electric and magnetic fields near a PEC boundary. In both Figure 4.43 and Figure 4.44, the PEC boundary is in the XY plane.

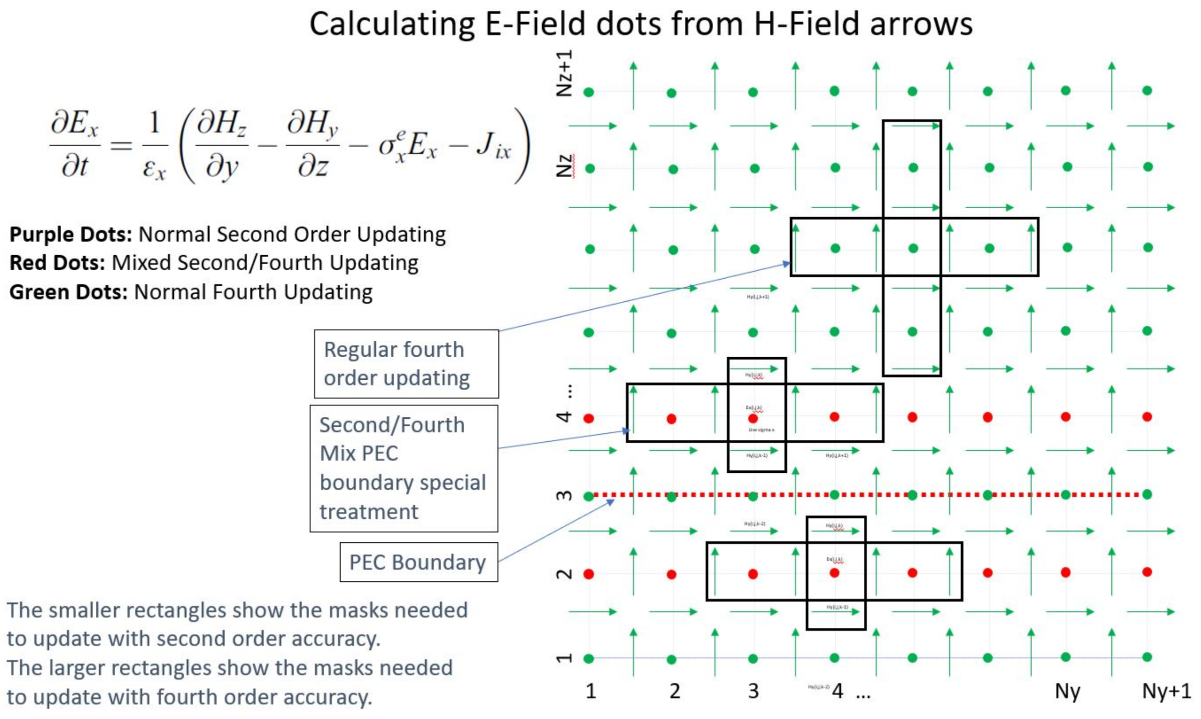
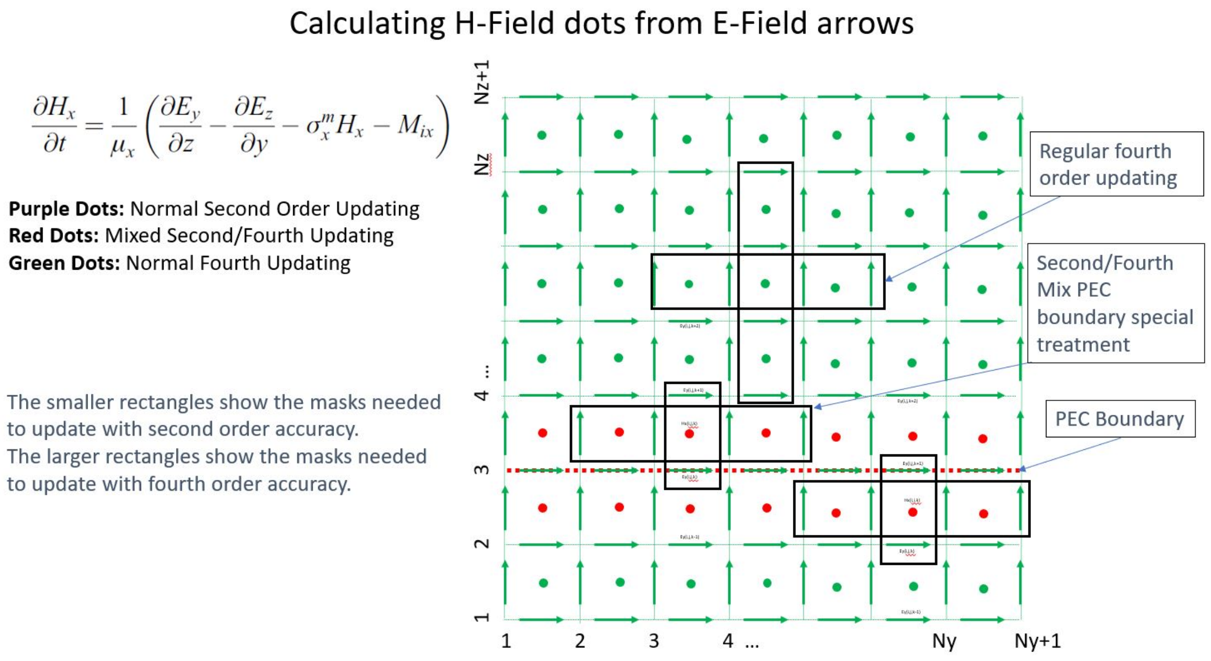

Calculating E-Field dots from H-Field arrows
$\frac{\partial E_x}{\partial t} = \frac{1}{\varepsilon_x}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma_x^e E_x - J_{ix}\right)$
Purple Dots: Normal Second Order Updating
Red Dots: Mixed Second/Fourth Updating
Green Dots: Normal Fourth Updating
Regular fourth order updating
Second/Fourth Mix PEC boundary special treatment
PEC Boundary
The smaller rectangles show the masks needed to update with second order accuracy.
The larger rectangles show the masks needed to update with fourth order accuracy.
Nz+1
Nz
4 …
3
2
1
1
2
3
4 …
Ny
Ny+1

Figure 4.43 Updating tangential electric fields near a PEC boundary.



Calculating H-Field dots from E-Field arrows

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu_x}\left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} - \sigma_x^m H_x - M_{ix}\right)$$

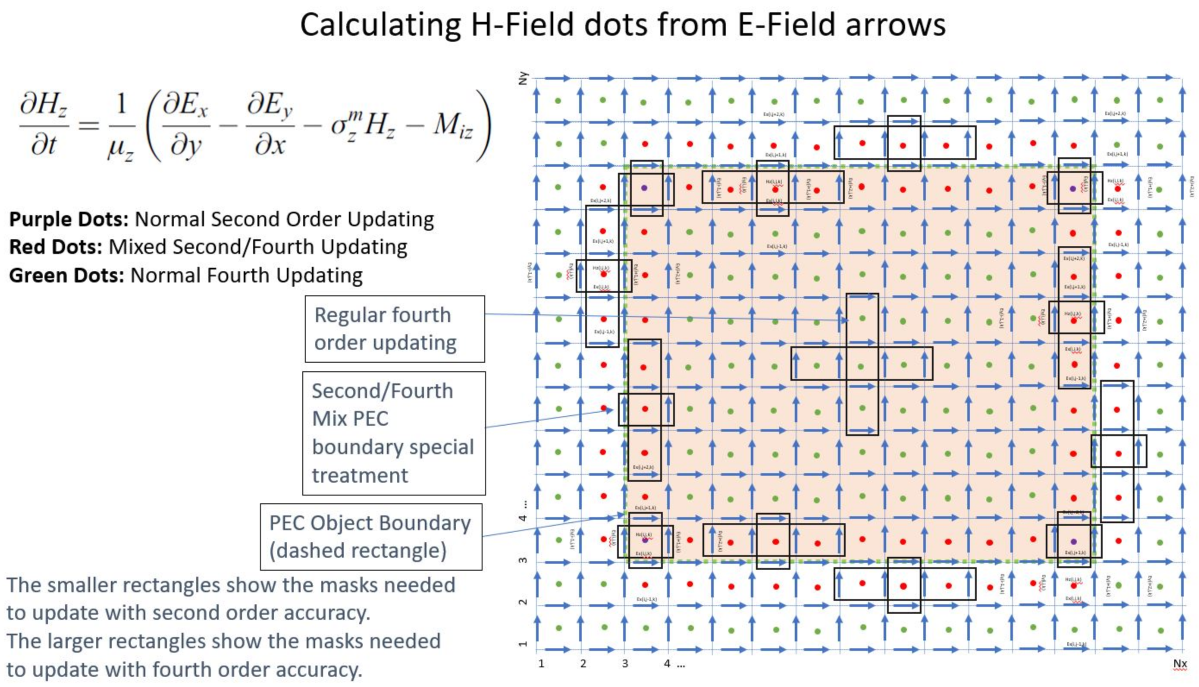
Purple Dots: Normal Second Order Updating
Red Dots: Mixed Second/Fourth Updating
Green Dots: Normal Fourth Updating

The smaller rectangles show the masks needed to update with second order accuracy.
The larger rectangles show the masks needed to update with fourth order accuracy.

Nz+1
Nz
4 …
3
2
1
1
2
3
4 …
Ny
Ny+1
Regular fourth order updating
Second/Fourth Mix PEC boundary special treatment
PEC Boundary

Figure 4.44 Updating tangential magnetic fields near a PEC boundary.

Figure 4.45 shows how the fourth order updating equations are modified when updating the perpendicular magnetic field near a PEC boundary. In this specific case, a PEC plate is shown in the XY plane and the updating equation masks for H_z are shown.



Calculating H-Field dots from E-Field arrows

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu_z}\left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} - \sigma_z^m H_z - M_{iz}\right)$$

Purple Dots: Normal Second Order Updating
Red Dots: Mixed Second/Fourth Updating
Green Dots: Normal Fourth Updating

The smaller rectangles show the masks needed to update with second order accuracy.
The larger rectangles show the masks needed to update with fourth order accuracy.

Regular fourth order updating
Second/Fourth Mix PEC boundary special treatment
PEC Object Boundary (dashed rectangle)
Ny
Nx
1
2
3
4 ...

Figure 4.45 Updating perpendicular magnetic fields near a PEC boundary.

Code listings Listing 4.23 and Listing 4.24 show exactly how the fourth to second order transformation coefficients are initiallized and applied to the updating of the electric fields. The code for updating the magnetic fields is very similar.

Listing 4.23: initialize S24 PEC coefficients

```
disp('initializing S24 PEC coefficients');

Hx = zeros(nxp1,ny,nz);
Hy = zeros(nx,nyp1,nz);
Hz = zeros(nx,ny,nzp1);
Ex = zeros(nx,nyp1,nzp1);
Ey = zeros(nxp1,ny,nzp1);
Ez = zeros(nxp1,nyp1,nz);

S24_c1_exhy     = ones (nxp1, nyp1 , nzp1);
S24_c1_exhz     = ones (nxp1, nyp1 , nzp1);
S24_c1_eyhx     = ones (nxp1, nyp1 , nzp1);
S24_c1_eyhz     = ones (nxp1, nyp1 , nzp1);
S24_c1_ezhx     = ones (nxp1, nyp1 , nzp1);
S24_c1_ezhy     = ones (nxp1, nyp1 , nzp1);
S24_c1_hxey     = ones (nxp1, nyp1 , nzp1);
S24_c1_hxez     = ones (nxp1, nyp1 , nzp1);
S24_c1_hyex     = ones (nxp1, nyp1 , nzp1);
S24_c1_hyez     = ones (nxp1, nyp1 , nzp1);
S24_c1_hzex     = ones (nxp1, nyp1 , nzp1);
S24_c1_hzey     = ones (nxp1, nyp1 , nzp1);

```

```
S24_c2_exhy      = ones (nxp1, nyp1 , nzp1);
S24_c2_exhz      = ones (nxp1, nyp1 , nzp1);
S24_c2_eyhx      = ones (nxp1, nyp1 , nzp1);
S24_c2_eyhz      = ones (nxp1, nyp1 , nzp1);
S24_c2_ezhx      = ones (nxp1, nyp1 , nzp1);
S24_c2_ezhy      = ones (nxp1, nyp1 , nzp1);
S24_c2_hxey      = ones (nxp1, nyp1 , nzp1);
S24_c2_hxez      = ones (nxp1, nyp1 , nzp1);
S24_c2_hyex      = ones (nxp1, nyp1 , nzp1);
S24_c2_hyez      = ones (nxp1, nyp1 , nzp1);
S24_c2_hzex      = ones (nxp1, nyp1 , nzp1);
S24_c2_hzey      = ones (nxp1, nyp1 , nzp1);

for i = 3:nx-2
    for j = 3:ny-2
        for k = 3:nz-2
%Updating Electric Fields
            if (abs(sigma_e_x(i,j,k+1)-sigma_e_x(i,j,k))>1e5) || (abs(sigma_e_x(i,j,
                k)-sigma_e_x(i,j,k-1))>1e5) %Update Ex: hy/dz
                S24_c1_exhy(i,j,k) = 0;
                S24_c2_exhy(i,j,k) = 27/24;
            end
            if (abs(sigma_e_x(i,j+1,k)-sigma_e_x(i,j,k))>1e5) || (abs(sigma_e_x(i,j,
                k)-sigma_e_x(i,j-1,k))>1e5) %Update Ex: hz/dy
                S24_c1_exhz(i,j,k) = 0;
                S24_c2_exhz(i,j,k) = 27/24;
            end
            if (abs(sigma_e_y(i,j,k+1)-sigma_e_y(i,j,k))>1e5) || (abs(sigma_e_y(i,j,
                k)-sigma_e_y(i,j,k-1))>1e5) %Update Ey: hx/dz
                S24_c1_eyhx(i,j,k) = 0;
                S24_c2_eyhx(i,j,k) = 27/24;
            end
            if (abs(sigma_e_y(i+1,j,k)-sigma_e_y(i,j,k))>1e5) || (abs(sigma_e_y(i,j,
                k)-sigma_e_y(i-1,j,k))>1e5) %Update Ey: hz/dx
                S24_c1_eyhz(i,j,k) = 0;
                S24_c2_eyhz(i,j,k) = 27/24;
            end
%Updating Magnetic Fields
            if (abs(sigma_e_y(i,j,k+1)-sigma_e_y(i,j,k))>1e5) %Update Hx: ey/dz (
                Outside border of PEC normal to X, top/bottom of PEC normal to Z)
                S24_c1_hxey(i,j,k) = 0;
                S24_c2_hxey(i,j,k) = 27/24;
            end
            if (sigma_e_y(i,j,k+2)-sigma_e_y(i,j,k+1)<-1e5) || (sigma_e_y(i,j,k)-
                sigma_e_y(i,j,k-1)>1e5) %Update Hx: ey/dz (inside border of PEC
                normal to X)
                S24_c1_hxey(i,j,k) = 0;
                S24_c2_hxey(i,j,k) = 27/24;
            end
            if (abs(sigma_e_z(i,j+1,k)-sigma_e_z(i,j,k))>1e5) %Update Hx: ez/dy (
                Outside border of PEC normal to X, top/bottom of PEC normal to Y)
                S24_c1_hxez(i,j,k) = 0;
                S24_c2_hxez(i,j,k) = 27/24;
            end
            if (sigma_e_z(i,j+2,k)-sigma_e_z(i,j+1,k)<-1e5) || (sigma_e_z(i,j,k)-
                sigma_e_z(i,j-1,k)>1e5) %Update Hx: ez/dy (inside border of PEC
                normal to X)
                S24_c1_hxez(i,j,k) = 0;
                S24_c2_hxez(i,j,k) = 27/24;
            end
            if (abs(sigma_e_x(i,j,k+1)-sigma_e_x(i,j,k))>1e5) %Update Hy: ex/dz (
                Outside border of PEC normal to Y, top/bottom of PEC normal to Z)
                S24_c1_hyex(i,j,k) = 0;
                S24_c2_hyex(i,j,k) = 27/24;
            end
            if (sigma_e_x(i,j,k+2)-sigma_e_x(i,j,k+1)<-1e5) || (sigma_e_x(i,j,k)-
                sigma_e_x(i,j,k-1)>1e5) %Update Hy: ex/dz (Inside border of PEC
                normal to Y)
                S24_c1_hyex(i,j,k) = 0;
                S24_c2_hyex(i,j,k) = 27/24;
            end
            if (abs(sigma_e_z(i+1,j,k)-sigma_e_z(i,j,k))>1e5) %Update Hy: ez/dx (
                Outside border of PEC normal to Y, top/bottom of PEC normal to X)
                S24_c1_hyez(i,j,k) = 0;
                S24_c2_hyez(i,j,k) = 27/24;
            end
            if (sigma_e_z(i+2,j,k)-sigma_e_z(i+1,j,k)<-1e5) || (sigma_e_z(i-1,j,k)-
                sigma_e_z(i,j,k)<-1e5) %Update Hy: ez/dx (Inside border of PEC normal
                 to Y)
                S24_c1_hyez(i,j,k) = 0;
                S24_c2_hyez(i,j,k) = 27/24;
            end
            if (abs(sigma_e_x(i,j+1,k)-sigma_e_x(i,j,k))>1e5) %Update Hz: ex/dy (
                outside border of PEC normal to Z)
                S24_c1_hzex(i,j,k) = 0;
                S24_c2_hzex(i,j,k) = 27/24;
            end
            if (sigma_e_x(i,j+2,k)-sigma_e_x(i,j+1,k)<-1e5) || (sigma_e_x(i,j-1,k)-
                sigma_e_x(i,j,k)<-1e5) %Update Hz: ex/dy (inside border of PEC normal
                 to Z)
                S24_c1_hzex(i,j,k) = 0;
                S24_c2_hzex(i,j,k) = 27/24;
            end
            if  (abs(sigma_e_y(i+1,j,k)-sigma_e_y(i,j,k))>1e5) %Update Hz: ey/dx (
                outside border of PEC normal to Z, top/bottom of PEC normal to X)
                S24_c1_hzey(i,j,k) = 0;
                S24_c2_hzey(i,j,k) = 27/24;
            end
            if (sigma_e_y(i+2,j,k)-sigma_e_y(i+1,j,k)<-1e5) || (sigma_e_y(i-1,j,k)-
                sigma_e_y(i,j,k)<-1e5) %Update Hz: ey/dx (inside border of PEC normal
                 to Z)
                S24_c1_hzey(i,j,k) = 0;
                S24_c2_hzey(i,j,k) = 27/24;
            end
        end
    end
end

figure(213)
tonyPlot = pcolor(squeeze(S24_c1_hzex(:,:,17)));
Tonyc = jet(200);
colormap(Tonyc);
colorbar;
caxis([0 1])
xlabel('Y-Axis','fontsize',12);
ylabel('X-Axis','fontsize',12);

figure(214)
```

```
tonyPlot = pcolor(squeeze(sigma_e_y(:,:,17)));
Tonyc = jet(200);
colormap(Tonyc);
colorbar;
caxis([0 1e10])
xlabel('Y-Axis','fontsize',12);
ylabel('X-Axis','fontsize',12);
```

Listing 4.24: update electric fields

```
% update electric fields except the tangential components
% on the boundaries
counterSample = counterSample + 1;
current_time  = current_time + dt/2;

%second order:
% Ex(1:nx,2:ny,2:nz) = Cexe(1:nx,2:ny,2:nz).*Ex(1:nx,2:ny,2:nz) ...
%                       + Cexhz(1:nx,2:ny,2:nz).*...
%                       (Hz(1:nx,2:ny,2:nz)-Hz(1:nx,1:ny-1,2:nz)) ...
%                       + Cexhy(1:nx,2:ny,2:nz).*...
%                       (Hy(1:nx,2:ny,2:nz)-Hy(1:nx,2:ny,1:nz-1));
%
% Ey(2:nx,1:ny,2:nz)=Ceye(2:nx,1:ny,2:nz).*Ey(2:nx,1:ny,2:nz) ...
%                        + Ceyhx(2:nx,1:ny,2:nz).*    ...
%                        (Hx(2:nx,1:ny,2:nz)-Hx(2:nx,1:ny,1:nz-1)) ...
%                        + Ceyhz(2:nx,1:ny,2:nz).*    ...
%                        (Hz(2:nx,1:ny,2:nz)-Hz(1:nx-1,1:ny,2:nz));
%
% Ez(2:nx,2:ny,1:nz)=Ceze(2:nx,2:ny,1:nz).*Ez(2:nx,2:ny,1:nz) ...
%                        + Cezhy(2:nx,2:ny,1:nz).*    ...
%                        (Hy(2:nx,2:ny,1:nz)-Hy(1:nx-1,2:ny,1:nz)) ...
%                        + Cezhx(2:nx,2:ny,1:nz).*...
%                        (Hx(2:nx,2:ny,1:nz)-Hx(2:nx,1:ny-1,1:nz));

%Fourth Order
%+1, 0, -1, -2

ExXb = 1; %1
ExXt = nx; %nx
ExYb = 3;
ExYt = ny-1;
ExZb = 3;
ExZt = nz-1;

Ex(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt) = Cexe(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Ex(ExXb:
    ExXt,ExYb:ExYt,ExZb:ExZt) ...
                             + (Cexhz(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt)./(24*S24_c2_exhz(ExXb:
                                 ExXt,ExYb:ExYt,ExZb:ExZt))).*... %dy
                             (-S24_c1_exhz(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Hz(ExXb:ExXt,
                                 ExYb+1:ExYt+1,ExZb:ExZt)... %dy
                             +27*Hz(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt)-27*Hz(ExXb:ExXt,ExYb-1:
                                 ExYt-1,ExZb:ExZt)... %dy
                             +S24_c1_exhz(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Hz(ExXb:ExXt,ExYb
                                 -2:ExYt-2,ExZb:ExZt)) ... %dy
                             + (Cexhy(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt)./(24*S24_c2_exhy(ExXb:
                                 ExXt,ExYb:ExYt,ExZb:ExZt))).*... %dz
```

```
                              (-S24_c1_exhy(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Hy(ExXb:ExXt,
                                 ExYb:ExYt,ExZb+1:ExZt+1)... %dz
                              +27*Hy(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt)-27*Hy(ExXb:ExXt,ExYb:
                                 ExYt,ExZb-1:ExZt-1)... %dz
                              +S24_c1_exhy(ExXb:ExXt,ExYb:ExYt,ExZb:ExZt).*Hy(ExXb:ExXt,ExYb
                                 :ExYt,ExZb-2:ExZt-2)); %dz

EyXb = 3;
EyXt = nx-1;
EyYb = 1; %1
EyYt = ny; %ny
EyZb = 3;
EyZt = nz-1;

Ey(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)=Ceye(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Ey(EyXb:EyXt,
   EyYb:EyYt,EyZb:EyZt) ...
                              + (Ceyhx(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)./(24*S24_c2_eyhx(EyXb:
                                 EyXt,EyYb:EyYt,EyZb:EyZt))).*    ... %dz
                              (-S24_c1_eyhx(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Hx(EyXb:EyXt,
                                 EyYb:EyYt,EyZb+1:EyZt+1)... %dz
                              +27*Hx(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)-27*Hx(EyXb:EyXt,EyYb:
                                 EyYt,EyZb-1:EyZt-1)... %dz
                              +S24_c1_eyhx(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Hx(EyXb:EyXt,EyYb
                                 :EyYt,EyZb-2:EyZt-2))  ... %dz
                              + (Ceyhz(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)./(24*S24_c2_eyhz(EyXb:
                                 EyXt,EyYb:EyYt,EyZb:EyZt))).*    ...
                              (-S24_c1_eyhz(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Hz(EyXb+1:EyXt
                                 +1,EyYb:EyYt,EyZb:EyZt)...
                              +27*Hz(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt)-27*Hz(EyXb-1:EyXt-1,EyYb
                                 :EyYt,EyZb:EyZt)...
                              +S24_c1_eyhz(EyXb:EyXt,EyYb:EyYt,EyZb:EyZt).*Hz(EyXb-2:EyXt-2,
                                 EyYb:EyYt,EyZb:EyZt)); %dx


EzXb = 3;
EzXt = nx-1;
EzYb = 3;
EzYt = ny-1;
EzZb = 1; %1
EzZt = nz; %nz

Ez(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)=Ceze(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Ez(EzXb:EzXt,
   EzYb:EzYt,EzZb:EzZt) ...
                              + (Cezhy(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)./(24*S24_c2_ezhy(EzXb:
                                 EzXt,EzYb:EzYt,EzZb:EzZt))).*    ... %dx
                              (-S24_c1_ezhy(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Hy(EzXb+1:EzXt
                                 +1,EzYb:EzYt,EzZb:EzZt)... %dx
                              +27*Hy(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)-27*Hy(EzXb-1:EzXt-1,EzYb
                                 :EzYt,EzZb:EzZt)... %dx
                              +S24_c1_ezhy(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Hy(EzXb-2:EzXt-2,
                                 EzYb:EzYt,EzZb:EzZt))  ... %dx
                              + (Cezhx(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)./(24*S24_c2_ezhx(EzXb:
                                 EzXt,EzYb:EzYt,EzZb:EzZt))).*...   %dy
                              (-S24_c1_ezhx(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Hx(EzXb:EzXt,
                                 EzYb+1:EzYt+1,EzZb:EzZt)...   %dy
                              +27*Hx(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt)-27*Hx(EzXb:EzXt,EzYb-1:
                                 EzYt-1,EzZb:EzZt)...   %dy
                              +S24_c1_ezhx(EzXb:EzXt,EzYb:EzYt,EzZb:EzZt).*Hx(EzXb:EzXt,EzYb
                                 -2:EzYt-2,EzZb:EzZt)); %dy
```

```

%S24_PEC_E2;
%Ex_sample(counterSample) = Ex(round(nx/3),round(ny/3),round((bricks(1).max_z -
    fdtd_domain.min_z)/dz)+1+5);

% figure(210)
% tonyPlot = pcolor(Ex(:,:,round((bricks(2).max_z - fdtd_domain.min_z)/dz)+1));
% Tonyc = jet(200);
% colormap(Tonyc);
% colorbar;
% caxis([-60 60])
% xlabel('Y-Axis','fontsize',12);
% ylabel('X-Axis','fontsize',12);
% title(['time step = ', num2str(counterSample), ' time = ', num2str(current_time*1
    e9), ' ns'],'fontsize',12);
% %set(tonyPlot, 'EdgeColor', 'none')
% F(counterSample) = getframe(gcf);
% drawnow;
%
% figure(211)
% tonyPlot = pcolor(Ey(:,:,round((bricks(2).max_z - fdtd_domain.min_z)/dz)+1));
% Tonyc = jet(200);
% colormap(Tonyc);
% colorbar;
% caxis([-60 60])
% xlabel('Y-Axis','fontsize',12);
% ylabel('X-Axis','fontsize',12);
% title(['time step = ', num2str(counterSample), ' time = ', num2str(current_time*1
    e9), ' ns'],'fontsize',12);
% %set(tonyPlot, 'EdgeColor', 'none')
% F1(counterSample) = getframe(gcf);
% drawnow;
%
% figure(212)
% tonyPlot = pcolor(Ez(:,:,round((bricks(1).max_z - fdtd_domain.min_z)/dz)+1));
% Tonyc = jet(200);
% colormap(Tonyc);
% colorbar;
% caxis([-60 60])
% xlabel('Y-Axis','fontsize',12);
% ylabel('X-Axis','fontsize',12);
% title(['time step = ', num2str(counterSample), ' time = ', num2str(current_time*1
    e9), ' ns'],'fontsize',12);
% %set(tonyPlot, 'EdgeColor', 'none')
% F2(counterSample) = getframe(gcf);
% drawnow;
```

4.8.7.2 Filter Results

Figure 4.46 shows the results of simulating the microstrip low-pass filter problem with the methodology presented in section 4.8.7.1. The geometry of the filter problem simulated in this section matches that of section 4.7.2.1 except for the dz cell size is set to one half the value and the simulation is run for 3243 time steps. The cell size in the Z-direction is reduced for this simulation because the original filter problem as defined in section 4.7.2.1 only has four cells descritizing the substrate. That does not allow much space to apply the fourth order special treatment of the PEC presented in this section. The cell size was decreased to allow the substrate to be descirtized by more cells and allow the PEC special treatment to take form.

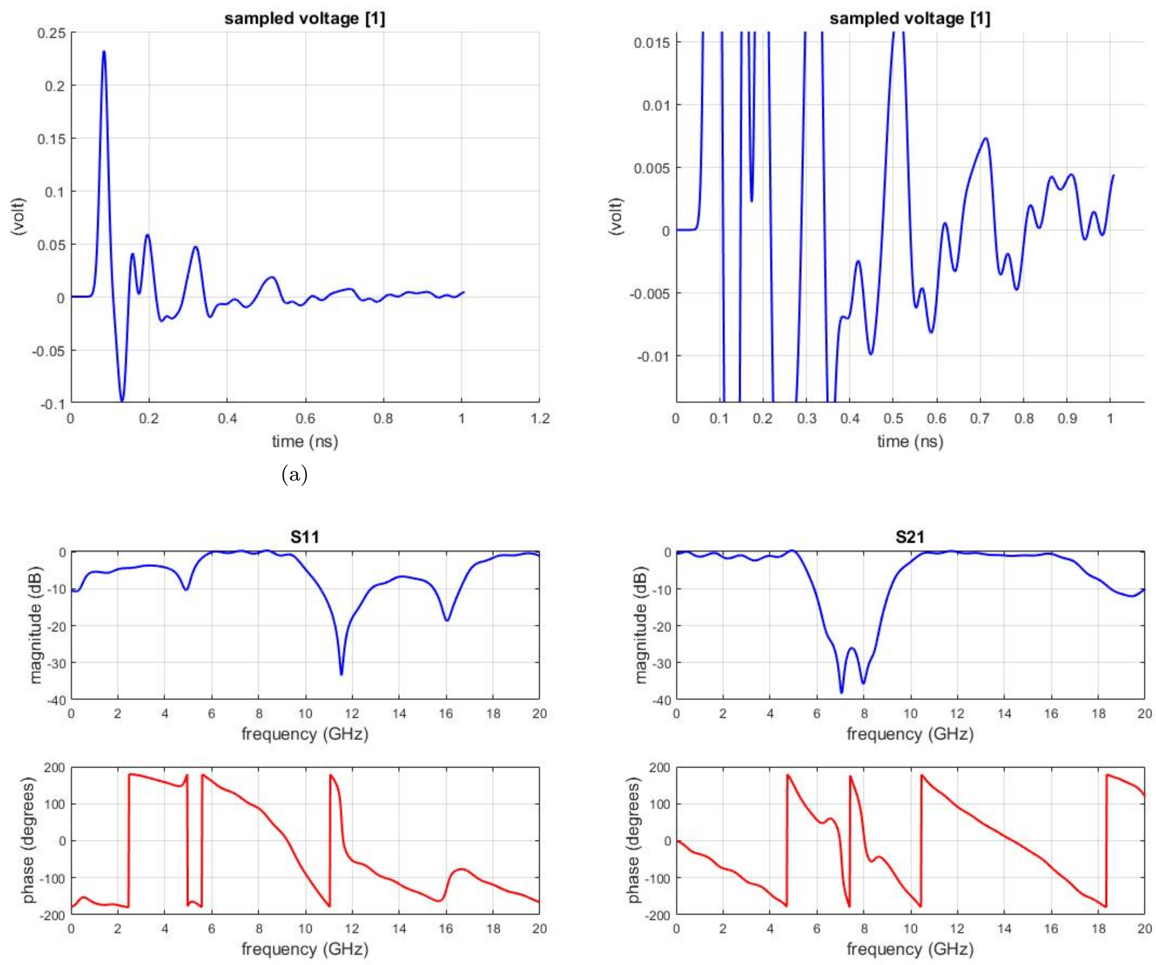
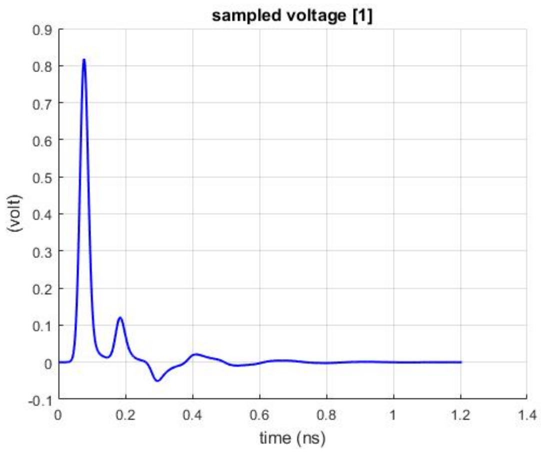


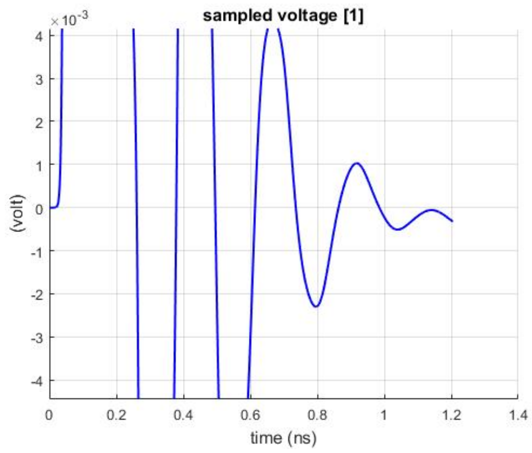
Figure 4.46 Fourth order PEC special treatment applied to the microstrip low-pass filter problem.

As is clear by examining Figure 4.46, converting the fourth order to second order updating around the PEC plates did not solve the low frequency error completely. In fact, it has introduced new errors. The filter problem in this simulation is only run for 3243 time steps, or 1 ns. By looking at Figure 4.46(a) specifically, it is clear the sampled voltage has not had enough time to die out. However, if the number of time steps is increased, the simulation begins to diverge and the results are even worse than what is shown in Figure 4.46. A working FDTD simulation should not diverge that quickly, implying the fourth order PEC plate special treatment presented in section 4.8.7.1 did not completely work out. More investigation is required to potentially fix the divergence issue.

4.8.7.3 Dipole Results

Figure 4.47 shows the results of simulating the PEC brick dipole antenna problem with the methodology presented in section 4.8.7.1. The geometry of the dipole problem simulated in this section matches that of section 4.7.1.1 except for the dz cell size is set to one half the value and the simulation is run for 1622 time steps.





(b)

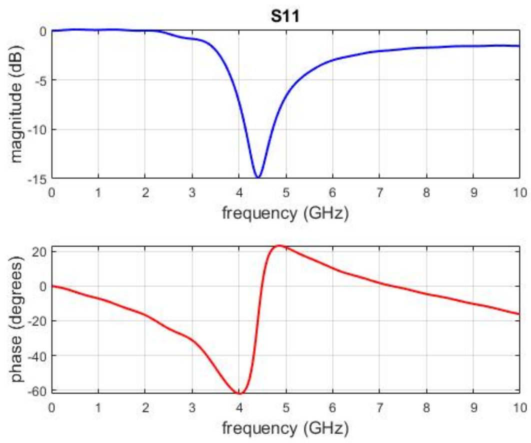


Figure 4.47 Fourth order PEC special treatment applied to the brick dipole problem.

As is clear by examining Figure 4.47, converting the fourth order to second order around the PEC bricks did not solve the low frequency error completely. In fact, it has introduced new errors. The brick dipole problem in this simulation was only run for 1622 time steps, or 1.2 ns. By looking at Figure 4.47(b) specifically, it is clear the sampled voltage has not had enough time to die out. However, if the number of time steps is increased, then the simulation begins to diverge, and the results are even worse than what is shown in Figure 4.47. A working FDTD simulation should not diverge that quickly, implying the fourth order PEC brick special treatment presented in section 4.8.7.1 does not work correctly, yet.

4.9 Verification of Fourth Order Dielectric Material Handling

Thus far, it has been shown that fourth order FDTD can correctly simulate free space (sections 4.1 and 4.2) and thin wires (section 4.6). Fourth order FDTD has not yet been shown to correctly simulate PEC objects (section 4.7). The goal of this section is to explore how well the fourth order FDTD code handles dielectric objects. Homogeneous dielectric materials will be simulated (section 4.9.1) as well as heterogeneous dielectric material (section 4.9.2) to explore the dielectric-dielectric material interface. This section also shows that a fourth order plane wave formulation has been implemented and is working as expected.

4.9.1 Radar Cross Section of a Dielectric Cube

In this section the basic Radar Cross Section (RCS) of a dielectric cube is calculated with second and fourth order accurate simulations. The goal of this section is to compare the fourth order simulation results to the second order simulation results. As shown in section 4.7, the fourth order accurate code does not handle PEC interfaces very well. Since the larger mask of the fourth order FDTD formulation does not handle PEC interfaces correctly, it is also possible that the larger mask does not handle other material interfaces. The goal of this section is to explore the results of this specific fourth order formulation when dielectric materials are involved.

4.9.1.1 Simulation Setup

The dielectric cube shown in Figure 5.25 is one of the simplest geometries to implement using a FDTD code. The blue cube is the dielectric (with $\epsilon_r = 5$), the red dashed box shows the inner boundary of the CPML layers, and the outer blue box shows the outer boundary of the domain. The code used to simulate this geometry impliments the plane wave formulation presented in section 2.2.5 and [1]. The RCS of the cube is calculated as a post process which is not effected by the switch to fourth order.

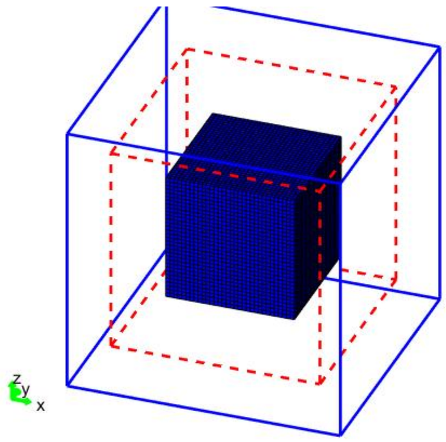


Figure 4.48 Problem space of the dielectric cube simulation.

Code Listing 4.25 shows the specifics of the geometry shown in Figure 5.25 while code Listing 4.26 defines the rest of the problem space parameters.

Listing 4.25: define geometry (RCS)

```
disp('defining the problem geometry');

bricks  = [];
spheres = [];
thin_wires = [];

% define dielectric
bricks(1).min_x = -80e-3;
bricks(1).min_y = -80e-3;
bricks(1).min_z = -80e-3;
bricks(1).max_x = 80e-3;
bricks(1).max_y = 80e-3;
bricks(1).max_z = 80e-3;
bricks(1).material_type = 4;
```

Listing 4.26: define problem space parameters (RCS)

```
disp('defining the problem space parameters');

% maximum number of time steps to run FDTD simulation
number_of_time_steps = 5400; %3000 %5400

tonyCellScaleFactor = 1;

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7)/tonyCellScaleFactor;

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = floor(20/tonyCellScaleFactor);
```

```

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = (5*tonyCellScaleFactor)*1e-3;
dy = (5*tonyCellScaleFactor)*1e-3;
dz = (5*tonyCellScaleFactor)*1e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 8;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 10;
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 10;
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 10;
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 10;
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 10;
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = 10;
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0.6;
boundary.cpml_kappa_max = 1;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

```

```
% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 5;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

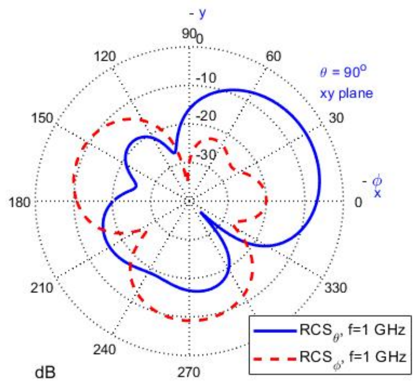
Finally, code Listing 4.27 defines the specifics of the plane wave incident on the dielectric cube.

Listing 4.27: define sources and lumped elements (RCS)

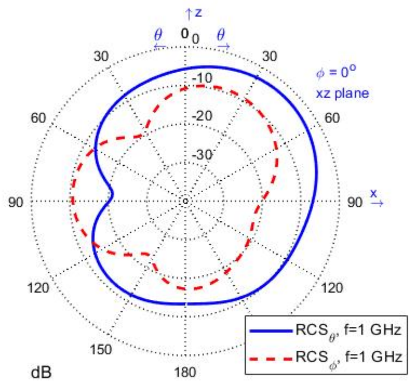
```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];
incident_plane_wave = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% Define incident plane wave, angles are in degrees
incident_plane_wave.E_theta = 1;
incident_plane_wave.E_phi = 0;
incident_plane_wave.theta_incident = 45;
incident_plane_wave.phi_incident = 30;
incident_plane_wave.waveform_type = 'gaussian';
incident_plane_wave.waveform_index = 1;
```

4.9.1.2 Simulation Results

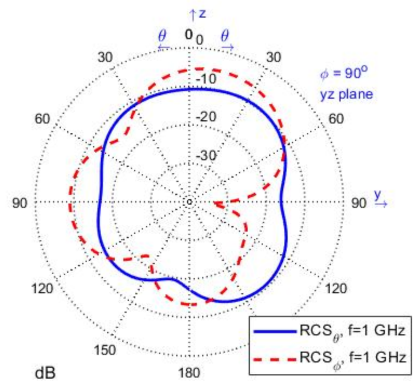
This section compares the second and fourth order simulation results for the RCS calculation of the dielectric cube described in section 4.9.1.1. Figure 4.49 shows the second order RCS in all three planes as well as the sampled electric field. Figure 4.50 shows the fourth order RCS in all three planes as well as the sampled electric field.



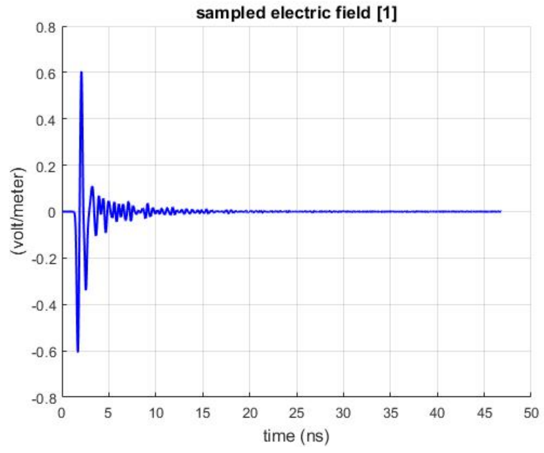
(a)



(b)

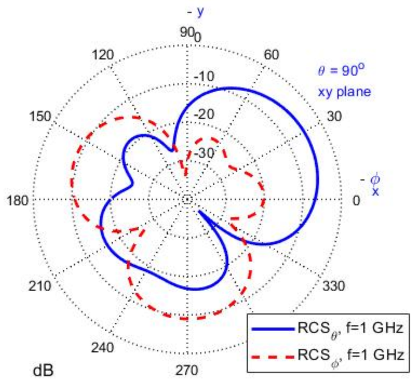


(c)

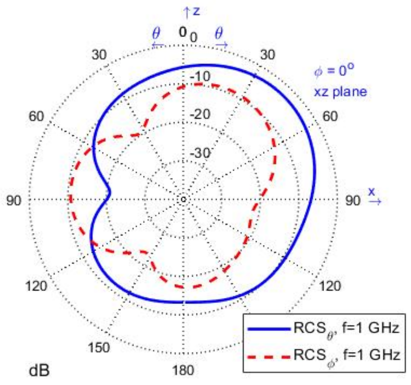


(d)

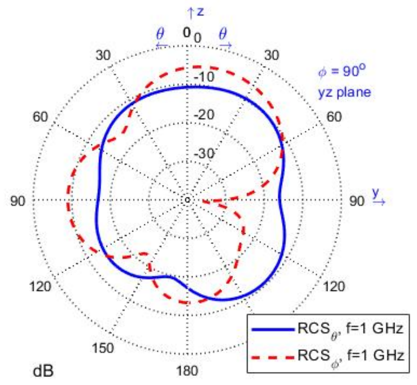
Figure 4.49 Second order accurate simulation calculating the bistatic RCS of a cube.



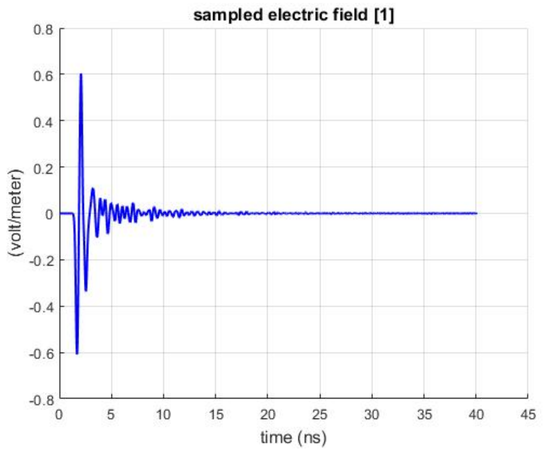
(a)



(b)



(c)



(d)

Figure 4.50 Fourth order accurate simulation calculating the bistatic RCS of a cube.

By comparing Figure 4.49 and Figure 4.50 it is clear the fourth order code produced the same results as the second order code. Both the second and fourth order results also agree with the results shown in section 11.5.2 in [1].

4.9.2 Radar Cross Section of a Muti-Material Dielectric Cube

In this section the basic RCS of a cube built from multiple dielectric materials is calculated with second and fourth order accurate simulations. This section will compare the fourth order simulation results to the second order simulation results. As shown in section 4.7, the fourth order accurate code does not handle PEC interfaces very well. However, section 4.9.1 shows that the fourth order code can simulate a homogeneous dielectric material without producing errors. A material interface between two dielectric materials of different permativities is another situation where the fourth order simulation could produce errors due to a discontinuous electric field. Even in a the standard second order FDTD scheme, careful attention to dielectric material interfaces is necessary [30]. This work by Hwang and Cangellaris details how the values of μ_r and ϵ_r are calculated at a material interface [30]. Since the stencils of the fourth order FDTD updating equations are larger than in the second order case, numerical errors may be introduced. The goal of this section is to explore the results of this specific fourth order formulation when dielectric material interfaces are involved.

4.9.2.1 Simulation Setup

The dielectric cube shown in Figure 4.51 shows the heterogeneous dielectric cube geometry. The multi-colored cube is the dielectric (with varying dielectric constants), the red dashed box shows the inner boundary of the CPML layers, and the outer blue box shows the outer boundary of the domain. The code used to simulate this geometry uses the plane wave formulation presented in section 2.2.5 and [1]. The RCS of the cube is calculated as a post process, which is not effected by the switch to fourth order.

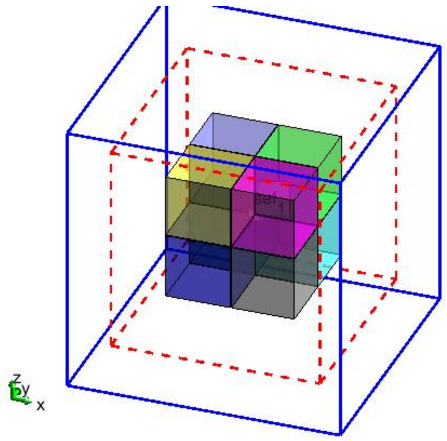


Figure 4.51 Problem space of a heterogeneous dielectric cube simulation.

Code Listing 4.28 shows the specifics of the geometry shown in Figure 4.51 while code Listing 4.29 defines the rest of the problem space parameters. Note that Listing 4.29 applies to the fourth order code where the number of times steps is 5400*7/6. For the second order code, only 5400 time steps are used to ensure both simulations are run for the same amount of total time.

Listing 4.28: define geometry (RCS)

```
disp('defining the problem geometry');

bricks   = [];
spheres = [];
thin_wires = [];
```

```

% define dielectric
bricks(1).min_x = -80e-3;
bricks(1).min_y = -80e-3;
bricks(1).min_z = -80e-3;
bricks(1).max_x = 1e-3;
bricks(1).max_y = 1e-3;
bricks(1).max_z = 1e-3;
bricks(1).material_type = 4;

bricks(2).min_x = 2e-3;
bricks(2).min_y = -80e-3;
bricks(2).min_z = -80e-3;
bricks(2).max_x = 80e-3;
bricks(2).max_y = 1e-3;
bricks(2).max_z = 1e-3;
bricks(2).material_type = 5;

bricks(3).min_x = -80e-3;
bricks(3).min_y = 2e-3;
bricks(3).min_z = -80e-3;
bricks(3).max_x = 1e-3;
bricks(3).max_y = 80e-3;
bricks(3).max_z = 1e-3;
bricks(3).material_type = 6;

bricks(4).min_x = 2e-3;
bricks(4).min_y = 2e-3;
bricks(4).min_z = -80e-3;
bricks(4).max_x = 80e-3;
bricks(4).max_y = 80e-3;
bricks(4).max_z = 1e-3;
bricks(4).material_type = 7;


bricks(5).min_x = -80e-3;
bricks(5).min_y = -80e-3;
bricks(5).min_z = 2e-3;
bricks(5).max_x = 1e-3;
bricks(5).max_y = 1e-3;
bricks(5).max_z = 80e-3;
bricks(5).material_type = 8;

bricks(6).min_x = 2e-3;
bricks(6).min_y = -80e-3;
bricks(6).min_z = 2e-3;
bricks(6).max_x = 80e-3;
bricks(6).max_y = 1e-3;
bricks(6).max_z = 80e-3;
bricks(6).material_type = 9;

bricks(7).min_x = -80e-3;
bricks(7).min_y = 2e-3;
bricks(7).min_z = 2e-3;
bricks(7).max_x = 1e-3;
bricks(7).max_y = 80e-3;
bricks(7).max_z = 80e-3;
bricks(7).material_type = 10;

bricks(8).min_x = 2e-3;
bricks(8).min_y = 2e-3;
bricks(8).min_z = 2e-3;
bricks(8).max_x = 80e-3;
bricks(8).max_y = 80e-3;
bricks(8).max_z = 80e-3;
bricks(8).material_type = 11;
```

Listing 4.29: define problem space parameters (RCS)

```
disp('defining the problem space parameters');

% maximum number of time steps to run FDTD simulation
number_of_time_steps = floor(54000*7/6); %3000 %5400

tonyCellScaleFactor = 1;

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7);

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = floor(20/tonyCellScaleFactor);

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = (5*tonyCellScaleFactor)*1e-3;
dy = (5*tonyCellScaleFactor)*1e-3;
dz = (5*tonyCellScaleFactor)*1e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 8;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 10;
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 10;
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 10;
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 10;
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 10;
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = 10;
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0.6;
boundary.cpml_kappa_max = 1;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 5;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

material_types(5).eps_r   = 10;
material_types(5).mu_r    = 1;
material_types(5).sigma_e = 0;
material_types(5).sigma_m = 0;
material_types(5).color   = [0.5 0.5 0.5];

material_types(6).eps_r   = 15;
material_types(6).mu_r    = 1;
material_types(6).sigma_e = 0;
material_types(6).sigma_m = 0;
material_types(6).color   = [0 0.5 1];

material_types(7).eps_r   = 20;
material_types(7).mu_r     = 1;
material_types(7).sigma_e = 0;
material_types(7).sigma_m = 0;
material_types(7).color    = [0.25 1 1];

material_types(8).eps_r    = 25;
material_types(8).mu_r     = 1;
material_types(8).sigma_e = 0;
material_types(8).sigma_m = 0;
material_types(8).color    = [1 1 0.25];

material_types(9).eps_r    = 30;
material_types(9).mu_r     = 1;
material_types(9).sigma_e = 0;
material_types(9).sigma_m = 0;
material_types(9).color    = [1 0 1];

material_types(10).eps_r    = 35;
material_types(10).mu_r     = 1;
material_types(10).sigma_e = 0;
material_types(10).sigma_m = 0;
material_types(10).color    = [0.5 0.5 1];

material_types(11).eps_r    = 40;
material_types(11).mu_r     = 1;
material_types(11).sigma_e = 0;
material_types(11).sigma_m = 0;
material_types(11).color    = [0.25 1 0.25];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

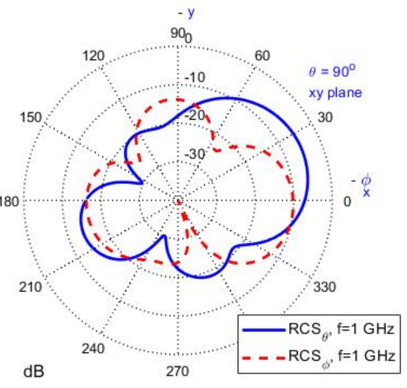
Finally, code Listing 4.30 defines the specifics of the plane wave incident on the dielectric cube.

Listing 4.30: define sources and lumped elements (RCS)

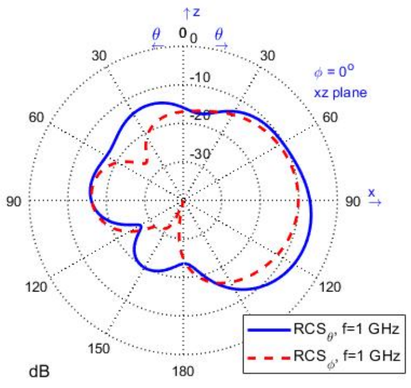
```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];
incident_plane_wave = [];

% define source waveform types and parameters
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;

% Define incident plane wave, angles are in degrees
incident_plane_wave.E_theta = 1;
incident_plane_wave.E_phi = 0;
incident_plane_wave.theta_incident = 45;
incident_plane_wave.phi_incident = 30;
incident_plane_wave.waveform_type = 'gaussian';
incident_plane_wave.waveform_index = 1;
```

4.9.2.2 Simulation Results

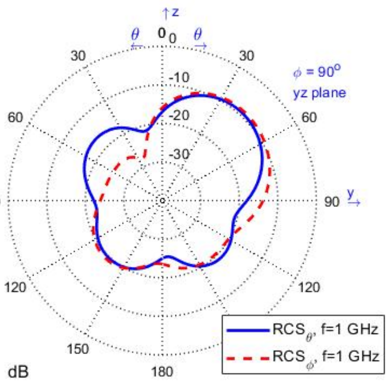
This section compares the second and fourth order simulation results for the RCS calculation of the dielectric cube described in section 4.9.2.1. Figure 4.52 shows the second order RCS in all three planes as well as the sampled electric field. Figure 4.53 shows the fourth order RCS in all three planes as well as the sampled electric field.



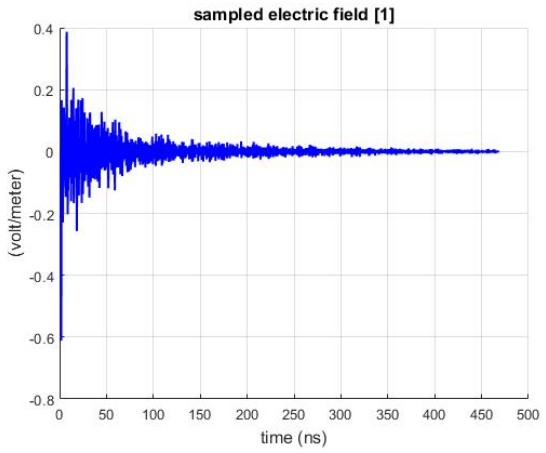
(a)



(b)

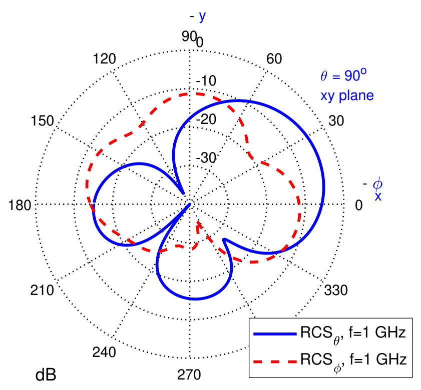


(c)

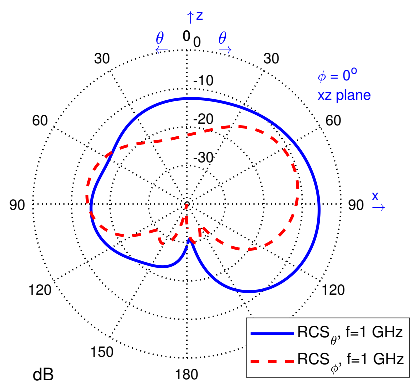


(d)

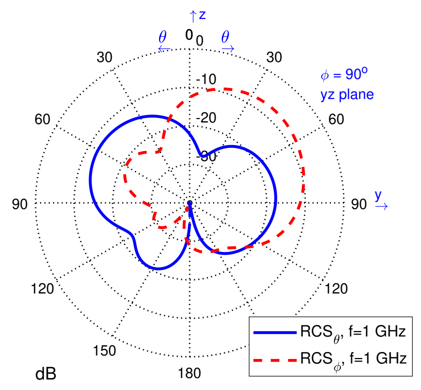
Figure 4.52 Second order accurate simulation calculating the RCS of a cube.



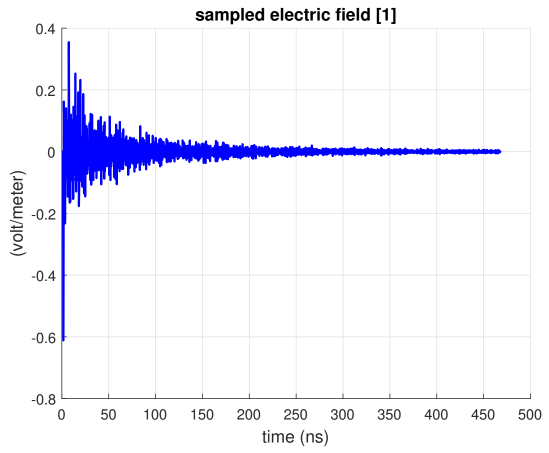
(a)



(b)



(c)



(d)

Figure 4.53 Fourth order accurate simulation calculating the RCS of a cube.

By comparing Figure 4.52 and Figure 4.53 it is clear the fourth order code produced different results than the second order code. Note that the permativities used to create the heterogeneous dielectric cube go up to $\epsilon_r = 40$. This is a very large relative permativity compared to when $\epsilon_r = 5$ in section 4.9.1. As shown in section 4.8.1, the unbalanced mask of the fourth order updating equation does not handle abrupt changes in field components. As the relative permativity of the dielectric material increases, the abruptness of the field change increases at the interfaces of the dielectric material. This fact explains why the fourth order results of section 4.9.1 matched the second order results while in this section the fourth order results do not match the second order results.

CHAPTER 5
PRACTICAL EXAMPLES

The goal of this chapter is to present fourth order simulations of practical electrically large structures. The advantage of using fourth order updating equations rather than second order accurate updating equations is increased accuracy. The increased accuracy of the fourth order updating equations allow for the size of a unit cell to increase without losing accuracy of the final simulation results. Increasing the cell size allows for less cells to be used to descritize a FDTD domain, and makes the simulation much more computationally efficient.

Section 4.1 shows very clearly that as the number of cells per wavelength decreases (or in other words the cell size increases), the error in the second order simulation is greater than the error in the fourth order simulation. In this chapter practical devices are simulated with both second and fourth order updating equations with different cell sizes to analyze the accuracy of each simulation's results.

5.1 Thin Wire Dipole

The first practical application of fourth order FDTD presented here is a thin wire dipole. In this section both a half-wavelength thin wire dipole and a long thin wire dipole are analyzed. Although a half wavelength dipole is not generally considered an electrically large problem, it is a problem with a well know solution. This makes it a good example to compare the accuracy of the second and fourth order simulations. A long dipole antenna can be considered an electrically larger problem, and due to the large range of frequencies analyzed, a good comparison can be made between the second and fourth order simulations. In order to make a good measurement of the fourth order and second order simulations, the analytical solution of the dipole antenna much be understood. One of the simplest antenna parameters the FDTD formulations should be able to calculate accurately is the S_{11} parameter of the dipole. Additionally, the analytical solution for the input impedance of a dipole antenna is well understood [28] and the minimums in the S_{11} curves are easy to compare. This S_{11} parameter comparison ultimately shows that at higher frequency (and smaller cells per wavelength ratio) the fourth order simulation is more accurate than the second order simulation.

5.1.1 Analytical Dipole

This section uses the results found in [28] to calculate the input impedance and S_{11} reflection coefficient of a dipole antenna.

Equations 5.1 and 5.2 show how the radiation resistance is calculated [28].

$$\begin{aligned} R_r =& \frac{\eta}{2\pi}\left(\gamma + \ln(kl) - C_i(kl) + \frac{1}{2}\sin(kl)\left(S_i(2kl) - 2S_i(kl)\right)\right) \\ &+ \frac{\eta}{2\pi}\left(\frac{1}{2}\cos(kl)\left(\gamma + \ln\left(\frac{kl}{2}\right) + C_i(2kl) - 2C_i(kl)\right)\right) \end{aligned} \tag{5.1}$$

$$\begin{aligned} X_r =& \frac{\eta}{2\pi}\left(2S_i(kl) + \cos(kl)\left(2S_i(kl) - S_i(2kl)\right)\right) \\ &+ \frac{\eta}{2\pi}\left(-\sin(kl)\left(2C_i(kl) - C_i(2kl) - C_i\left(\frac{2ka^2}{l}\right)\right)\right) \end{aligned} \tag{5.2}$$

where l is the length of the dipole, a is the diameter of the thin wire, $\eta = 120\pi$, and γ is the Euler-Mascheroni constant. Once the radiation impedance is calculated, the next step is to calculate the input impedance of the dipole. Equations 5.3 and 5.4 show the calculation of input impedance from radiation impedance.

$$R_{in} = \frac{R_r}{\sin^2(kl/2)} \tag{5.3}$$

$$X_{in} = \frac{X_r}{\sin^2(kl/2)} \tag{5.4}$$

In order to calculate the input impedance shown, the sine and cosine integrals must be evaluated. Equations 5.5 and 5.6 show the equations for these integrals [28].

$$S_i(x) = \int_0^x \frac{\sin(t)}{t}dt \tag{5.5}$$

$$C_i(x) = \int_0^x \frac{\cos(t) - 1}{t}dt + ln(x) + \gamma \tag{5.6}$$

Where γ is the Euler-Mascheroni constant.

Figure 5.1 shows the calculation of the sine and cosine integrals required to calculate the radiation resistance.

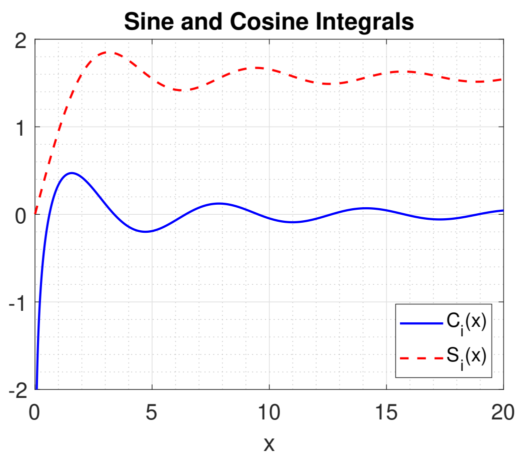


Figure 5.1 The evaluation of the sinusoidal integrals required to calculate the input impedance of a dipole antenna.

Figure 5.1 agrees very will with the results found in [28]. It is important to ensure equations 5.5 and 5.6 are calculated correctly since these integrals show up multiple times in the radiation resistance equations.

Figure 5.2 presents the analytically calculated input impedance of a dipole antenna with a radius of 0.5 mm and a length of 160 mm.

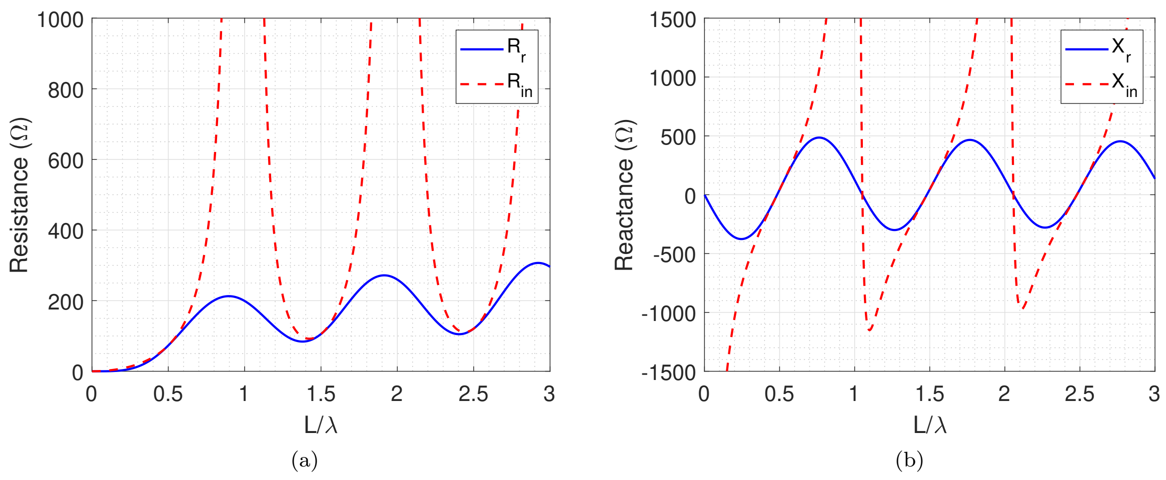


Figure 5.2 The analytical input resistance, radiation resistance, input reactance, and radiation reactance of a dipole antenna.

Figure 5.2 agrees very well with the results found in [28].

Finally, the S_{11} reflection coefficient can now be calculated with equation 5.7:

$$S_{11} = \frac{Z_{in} - Z_0}{Z_{in} + Z_0} \tag{5.7}$$

where Z_0 is the port impedance of the dipole and assumed to be 50Ω. Z_{in} is the input impedance of the dipole.

Figure 5.3 shows the S_{11} reflection coefficients for two different sized dipoles. Figure 5.3(a) shows the S_{11} coefficient of a 20 mm long dipole while Figure 5.3(b) shows the S_{11} coefficient for a 160 mm long dipole.

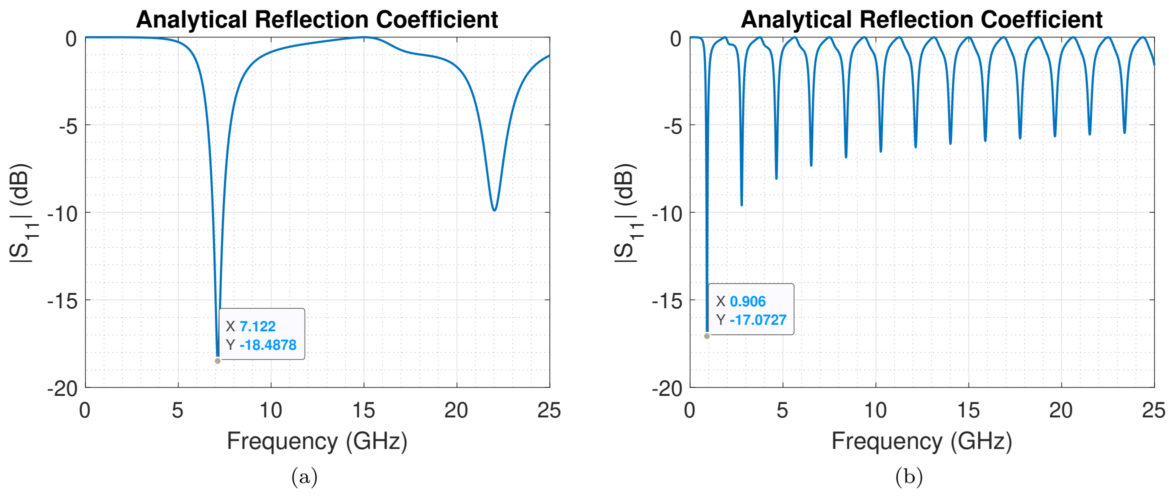


Figure 5.3 The analytical S_{11} coeficient of two different length dipole antennas.

The results of Figure 5.3 are used to evaluate the accuracy of the second and fourth order FDTD simulations in the following sections.

Listing 5.1 shows the matlab code used to produce the results shown in Figure 5.1, Figure 5.2 and Figure 5.3.

Listing 5.1: Calculating the S11 minimum of a dipole antenna

```
close all;
clear all;

setfigures;

a = 0.05e-3;
L = 20e-3; %160 or 20
freq = (0:0.001:25)*1e9;
lambda = 3e8./freq;
k = 2*pi./lambda;

x_axis = L./lambda;

%Sine and cosine integrals!!
%https://mathworld.wolfram.com/CosineIntegral.html
%https://keisan.casio.com/exec/system/1180573420

```

```
num_rectangles = 3000;
%%%%%%%%%%%%%%%% Si kl %%%%%%%%%%%%
x_int = k*L;

si_kl = zeros(1,length(x_int));
si_kl_temp = 0;

for i = 1:length(x_int)
    for j = 1:num_rectangles
        time = j*(x_int(i)/num_rectangles);
        si_kl_temp = si_kl_temp + (sin(time)/time)*(x_int(i)/num_rectangles);
    end
    si_kl(i) = si_kl_temp;
    si_kl_temp = 0;
end

%%%%%%%%%%%%%%%% Si 2*kl %%%%%%%%%%%%
x_int = 2*k*L;

si_2kl = zeros(1,length(x_int));
si_2kl_temp = 0;

for i = 1:length(x_int)
    for j = 1:num_rectangles
        time = j*(x_int(i)/num_rectangles);
        si_2kl_temp = si_2kl_temp + (sin(time)/time)*(x_int(i)/num_rectangles);
    end
    si_2kl(i) = si_2kl_temp;
    si_2kl_temp = 0;
end

%%%%%%%%%%%%%%%% Ci kl %%%%%%%%%%%%
x_int = k*L;

ci_kl = zeros(1,length(x_int));
ci_kl_temp = 0;

for i = 1:length(x_int)
    for j = 1:num_rectangles
        time = j*(x_int(i)/num_rectangles);
        ci_kl_temp = ci_kl_temp + ((cos(time)-1)/time)*(x_int(i)/num_rectangles);
    end
    ci_kl(i) = ci_kl_temp + log(x_int(i))+0.57721566; %Euler-Mascheroni Constant
    ci_kl_temp = 0;
end

%%%%%%%%%%%%%%%% Ci 2*kl %%%%%%%%%%%%
x_int = 2*k*L;

ci_2kl = zeros(1,length(x_int));
ci_2kl_temp = 0;

for i = 1:length(x_int)
    for j = 1:num_rectangles
        time = j*(x_int(i)/num_rectangles);
        ci_2kl_temp = ci_2kl_temp + ((cos(time)-1)/time)*(x_int(i)/num_rectangles);
    end
    ci_2kl(i) = ci_2kl_temp + log(x_int(i))+0.57721566; %Euler-Mascheroni Constant
    ci_2kl_temp = 0;
```

```
end

%%%%%%%%%%%%%%%% Ci 2*k*a %%%%%%%%%%%%%
x_int = 2*k*a^2/L;

ci_2ka = zeros(1,length(x_int));
ci_2ka_temp = 0;

for i = 1:length(x_int)
    for j = 1:num_rectangles
        time = j*(x_int(i)/num_rectangles);
        ci_2ka_temp = ci_2ka_temp + ((cos(time)-1)/time)*(x_int(i)/num_rectangles);
    end
    ci_2ka(i) = ci_2ka_temp + log(x_int(i))+0.57721566; %Euler-Mascheroni Constant
    ci_2ka_temp = 0;
end

x_int = k*L;
figure;
plot(x_int,ci_kl,'b-')
hold on;
plot(x_int,si_kl,'r--')
xlim([0 20])
ylim([-2 2])
legend('C_i(x)','S_i(x)','Location','southeast')
xlabel('x')
title('Sine and Cosine Integrals')
grid on;
grid minor;

%%%%%%% R_r %%%%%%%%
c_val = 0.57721566;
R_r = (120*pi/(2*pi))*(c_val + log(k*L)-ci_kl+(1/2)*sin(k*L).*(si_2kl-2*si_kl)+(1/2)
    *cos(k*L).*(c_val+log(k*L/2)+ci_2kl-2*ci_kl));
R_in = R_r./sin(k*L/2).^2;

%%%%%%% X_m %%%%%%%%
X_m = (120*pi/(4*pi))*(2*si_kl+cos(k*L).*(2*si_kl-si_2kl)-sin(k*L).*(2*ci_kl-ci_2kl-
    ci_2ka));
X_in = X_m./sin(k*L/2).^2;

%%%%%% R_r and X_m plots to match page 427 of tectbook %%%%%%%%
figure;
plot(x_axis,R_r,'b-')
hold on;
plot(x_axis,R_in,'r--')
xlim([0 3])
ylim([0 1000])
legend('R_r','R_{in}')
xlabel('L/\lambda')
ylabel('Resistance (\Omega)')
grid on;
grid minor;

figure;
plot(x_axis,X_m,'b-')
hold on;
plot(x_axis,X_in,'r--')
xlim([0 3])
```

```
ylim([-1500 1500])
legend('X_r','X_{in}')
xlabel('L/\lambda')
ylabel('Reactance (\Omega)')
grid on;
grid minor;

R_in(floor((3e8/0.320)/0.001e9))
X_in(floor((3e8/0.320)/0.001e9))

R_in(floor((7.151e9)/0.001e9))
X_in(floor((7.151e9)/0.001e9))


reflection = 20*log10(abs(((R_in+(1j)*X_in)-50)./(50+(R_in+(1j)*X_in))));

figure;
plot(freq/1e9,reflection)
ylim([-20, 0])
grid on;
grid minor;
xlabel('Frequency (GHz)')
ylabel('|S_{11}| (dB)')
title('Analytical Reflection Coefficient')
```

5.1.2 Half Wavelength Dipole

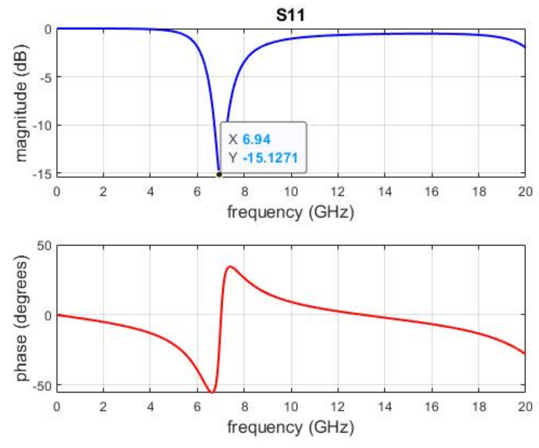
This section simulates the dipole antenna described in section 4.6, but the cell size is changed. The more accurate fourth order simulation should still produce good results at larger cell sizes while the second order simulation is expected to produce less accurate results. The dimensions of the dipole are preserved as the cell size is increased. For example, if the cell size doubles, the number of cells used to discretize the dipole halves. The CPML remains 8 cells thick and the air buffer remains 10 cells thick even when the cell size increases. Additionally, the Gaussian pulse of the voltage source does not change based on cell size. The variable that defines the maximum number of cells per wavelength of the Gaussian pulse changes such that no matter what the cell size is, the waveform of the voltage source doesn't change. For example, if the cell size doubles, the maximum number of cells per wavelength of the Gaussian source halves. Listing 5.2 shows how a cell size scale is implemented in the fourth order code.

Listing 5.2: define problem space parameters (Fourth Order Practical Dipole)

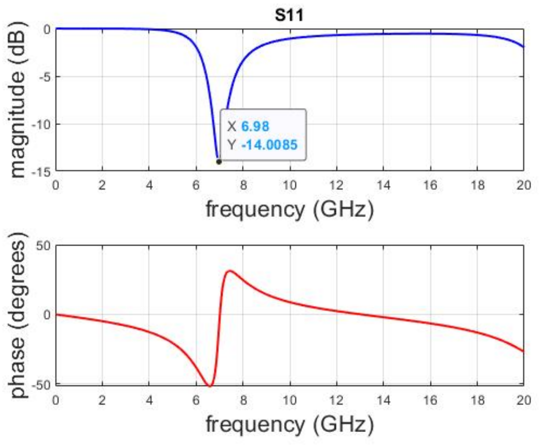
```
disp('defining the problem space parameters');

cellSizeScale = 10;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%parameters for creating 2D array in XZ plane.
array_dx = 20e-3;
array_dz = 30e-3;
```

```
n_elements_x = 1;
n_elements_z = 1;

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

% maximum number of time steps to run FDTD simulation
number_of_time_steps = floor(4000/cellSizeScale); %4000

% A factor that determines duration of a time step
% wrt CFL limit
courant_factor = 0.9*(6/7);

% A factor determining the accuracy limit of FDTD results
number_of_cells_per_wavelength = floor(20/cellSizeScale);

% Dimensions of a unit cell in x, y, and z directions (meters)
dx = cellSizeScale*0.25e-3;
dy = cellSizeScale*0.25e-3;
dz = cellSizeScale*0.25e-3;

% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

pecTony = 0;
air_buffer_Tony = 10;

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = floor(air_buffer_Tony);
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 0.6;
boundary.cpml_kappa_max = 1;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

% substrate
material_types(4).eps_r   = 2.2;
material_types(4).mu_r    = 1;
material_types(4).sigma_e = 0;
material_types(4).sigma_m = 0;
material_types(4).color   = [0 0 1];

% index of material types defining air, PEC, and PMC
material_type_index_air = 1;
material_type_index_pec = 2;
material_type_index_pmc = 3;
```

Figure 5.4 and Figure 5.5 show the difference between the second order and the fourth order thin wire dipole simulation using two different cell sizes. Figure 5.4 simulates the dipole with a cell size of 0.25 mm while Figure 5.5 simulates the dipole with a cell size of 1.25 mm. Figure 5.4(a) and Figure 5.5(a) show the S_{11} reflection coefficient for the dipole antenna simulated with second order updating equations. Figure 5.4(b) and Figure 5.5(b) show the S_{11} for the dipole antenna using the fourth order code that updates the field components near the thin wire with pure second order updating equations (see listing Listing 2.4).



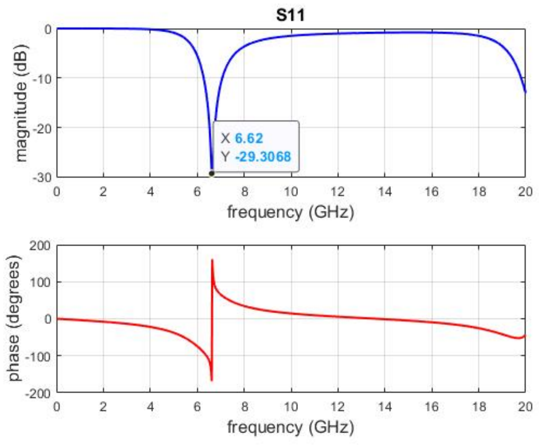
(a)



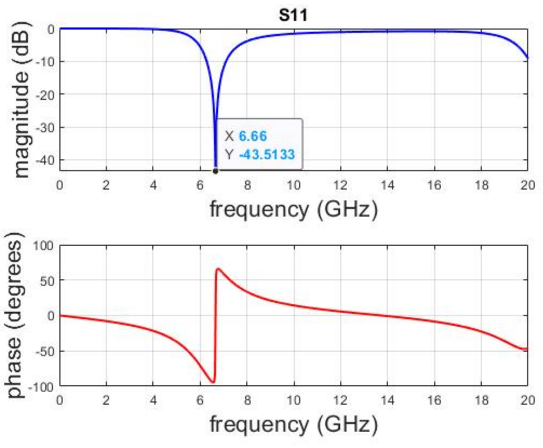
(b)

Figure 5.4 S11 coefficient for the thin wire dipole with a cell size of 0.25 mm.

As shown in Figure 5.4, when the cell size is 0.25 mm in all directions, the second order (Figure 5.4(a)) and the fourth order with the pure second order updating near the thin wire (Figure 5.4(b)) have almost the same S_{11}. Additionally, the expected minimum S_{11} for a 20 mm long dipole is 7.15 GHz as shown in figure Figure 5.3(a). The simulated minimum S_{11} in the second order simulation is 6.94 GHz (2.9% error) and the simulated S_{11} minimum in the fourth order simulation is 6.98 (2.4 % error). Both the second and fourth order simulations accurately predicted the frequency of the minimum S_{11} coefficient. This result is expected because both the second and fourth order simulations should produce good results when the simulation uses a cell size of 0.25mm.



(a)



(b)

Figure 5.5 S_{11} coefficient for the thin wire dipole with a cell size of 1.25 mm.

As shown in Figure 5.5, when the cell size is 1.25 mm in all directions the second order (Figure 5.5(a)) and the fourth order with the pure second order updating near the thin wire (Figure 5.5(b)) have almost the same S_{11}. The expected minimum S_{11} for a 20 mm long dipole is 7.15 GHz as shown in figure Figure 5.3(a). The simulated minimum S_{11} in this second order simulation is 6.62 GHz (7.4% error) and the simulated S_{11} minimum in this fourth order simulation is 6.66 (6.9 % error). Both the second and fourth order simulations have very similar percent errors and neither result matches the results of Figure 5.4 when the cell size is smaller.

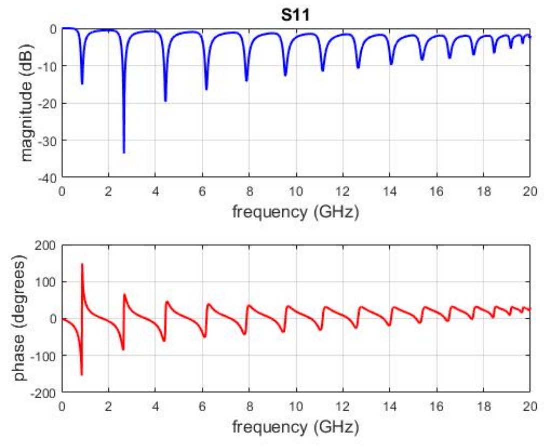
A likely conclusion is the increased cell size resulted in the geometry of the dipole not being represented by a sufficient number of cells (roughly 10 cells per thin wire). The feed of the dipole antenna is always kept to be two cells wide, regardless of the cell size. This means of course that as the cell size increases, the feed length also increases. Additionally, with a cell size of 0.25mm the dipole is descritized with 40 cells per thin wire. With a cell size of 1.25mm, only 8 cells are used per thin wire. Finally, the number of cells per wavelength at the frequency of interest (7.5 GHz) is high in both simulations. For a cell size of 0.25mm, there are 160 cells per wavelength and for 1.25mm cell size there are 32 cells per wavelength. In [1] it is shown that a second order code is accurate down to a 20 cells per wavelength resolution and section 4.1 shows that a fourth order simulation is accurate down to 5 cells per wavelength. To get down to 5 cells per wavelength at 7.5 GHz, the cell size would have to be 8mm and the dipole geometry itself would be descritized by one cell per thin wire. This would not be adequate, and it can be concluded a fourth order simulation is not advantageous over a second order simulation for a half wavelength dipole antenna.

5.1.3 Long Dipole Antenna

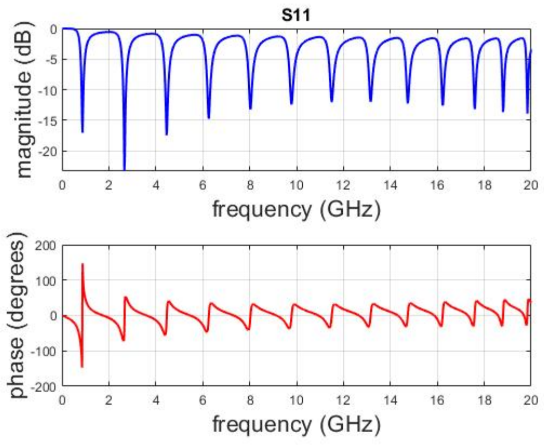
In this section, a practical example that shows the advantages of using fourth order updating over using second order updating is presented. As a long dipole antenna can be considered and electrically large problem, the advantage of fourth order FDTD over second order FDTD is expected to be more clear.

The total length of the dipole in this section is 160mm and it is oriented in the Z-direction. The voltage source is located at the center of the dipole and is 2 cells long. This voltage source produces a Gaussian pulse with a max frequency spanning 4 cells per wavelength. The cells size is set to 5mm, additionally, the simulation includes 10 cells of air buffer and 8 cells of CPML.

Figure 5.6 shows the S_{11} coefficient of the dipole as a function of frequency.



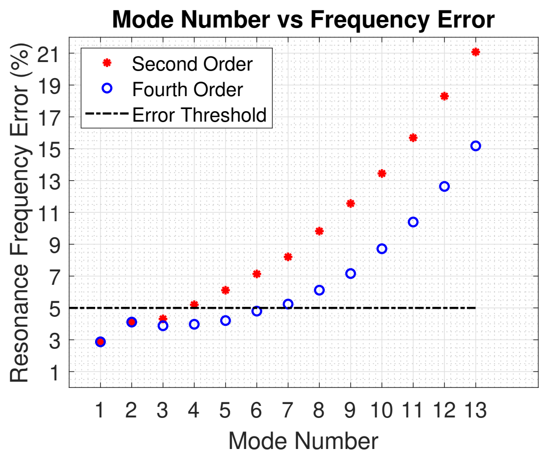
(a)



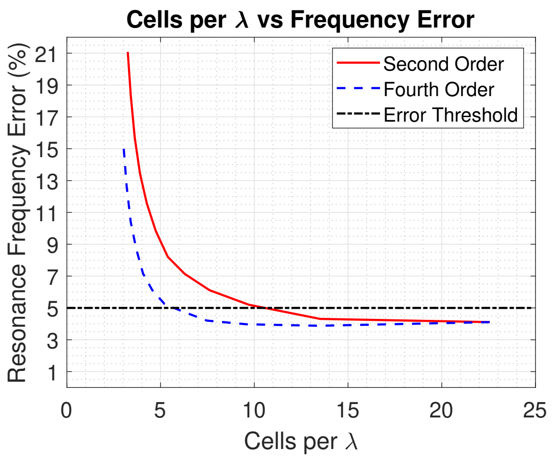
(b)

Figure 5.6 The S_{11} coefficient of the dipole antenna simulated with second order and fourth order codes.

Figure 5.6(a) are the second order simulation results and Figure 5.6(b) are the fourth order simulation results. The simulated results shown in Figure 5.6 are compared to the analytical expected results shown in Figure 5.3(b). Figure 5.7 and Figure 5.9 show the error in the simulated S_{11} minimums of the second and fourth order simulations compared to the analytical results. Specifically, Figure 5.7 shows the error in the frequency value of each simulated S_{11} minimum. Figure 5.9 shows the error in the S_{11} minimum's frequency spacing, as in the frequency difference between each minimum.



(a)



(b)

Figure 5.7 The total error in the resonant frequency of the dipole, plotted against peak number and the cells per wavelength of the peak.

Note that the data that generated the plotts in Figure 5.7(a) and Figure 5.7(b) is the same, the difference is the x-axis. In Figure 5.7(a), the x-axis is the mode number of the S_{11} minimum, while the x-axis in Figure 5.7(b) is the number of cells per wavelength at the frequency of the S_{11} minimum. Figure 5.7(b) is particularly interesting because it shows that as the number of cells per wavelength decreases (meaning the resolution of the FDTD grid is decreasing), the error in the simulated S_{11} minimums increase.

Another key parameter we can look at to measure the accuracy of the second and fourth order FDTD codes is the frequency spacing between the modes of the long dipole antenna. This frequency spacing is calculated by measuring the difference between the minimums in the S_{11} curve. Figure 5.8 visually shows this calculation.

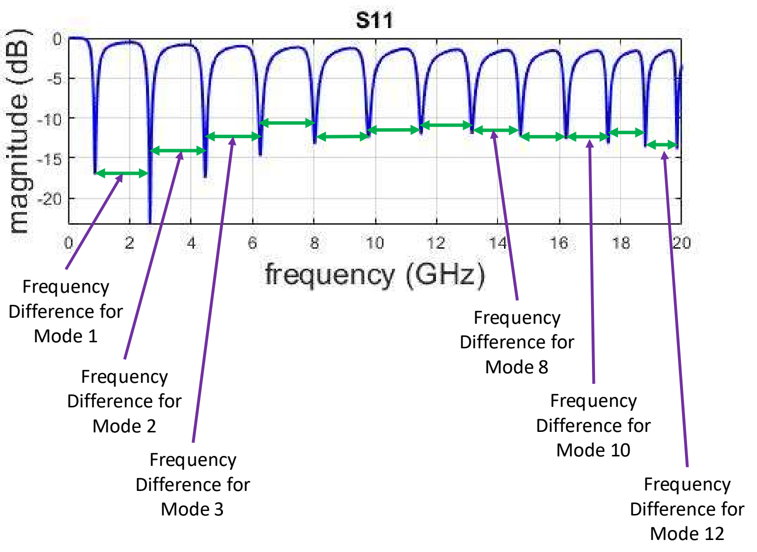
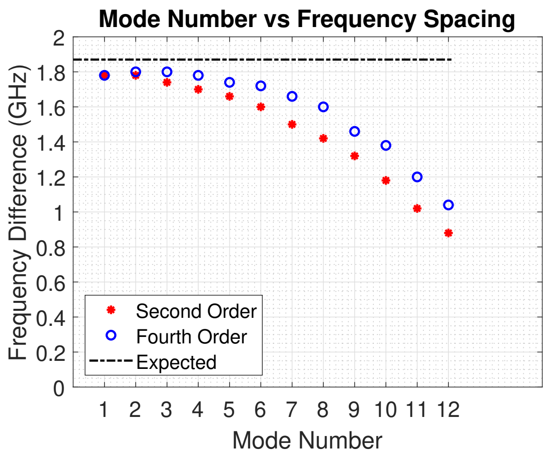
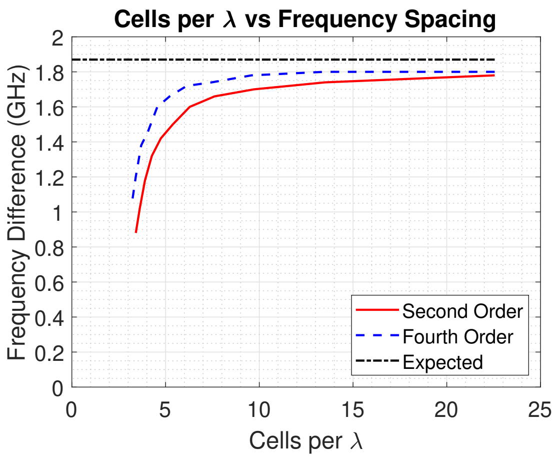


Figure 5.8 The definition of how frequency spacing is calculated between the modes.

Figure 5.9 shows the second and fourth order results of this frequency difference analysis.



(a)



(b)

Figure 5.9 The difference in frequency between peaks, plotted against peak number and the cells per wavelength of the peak.

Note the data plotted in Figure 5.9(a) and Figure 5.9(b) is the same, the difference is the x-axis. In Figure 5.9(a), the x-axis is the mode number of the S_{11} minimum, while the x-axis in Figure 5.9(b) is the number of cells per wavelength at the frequency of the S_{11} minimum. Figure 5.9(b) is particularly interesting because it shows how as the number of cells per wavelength decreases (meaning the resolution of the FDTD grid is decreasing), the error in the simulated S_{11} minimums increase.

Table 5.1 summarizes the second order expected and simulated S_{11} minimums.

Table 5.1 Mode frequencies of the long dipole antenna simulated with the second order code

Mode Number	Analytical f (GHz)	FDTD f (GHz)	Cells/λ	Δf (GHz)	f Error (%)
1	0.91	0.88	68.18	1.78	2.87
2	2.77	2.66	22.56	1.78	4.11
3	4.64	4.44	13.51	1.74	4.31
4	6.52	6.18	9.71	1.7	5.2
5	8.39	7.88	7.61	1.66	6.11
6	10.27	9.54	6.29	1.6	7.14
7	12.14	11.14	5.39	1.5	8.21
8	14.02	12.64	4.75	1.42	9.82
9	15.9	14.06	4.27	1.32	11.56
10	17.77	15.38	3.9	1.18	13.44
11	19.64	16.56	3.62	1.02	15.69
12	21.52	17.58	3.41	0.88	18.3
13	23.39	18.46	3.25	NA	21.08

Table 5.2 summarizes the fourth order expected and simulated S_{11} minimums.

Table 5.2 Mode frequencies of the long dipole antenna simulated with the fourth order code

Mode Number	Analytical f (GHz)	FDTD f (GHz)	Cells/λ	Δf (GHz)	f Error (%)
1	0.91	0.88	68.18	1.78	2.87
2	2.77	2.66	22.56	1.8	4.11
3	4.64	4.46	13.45	1.8	3.88
4	6.52	6.26	9.58	1.78	3.97
5	8.39	8.04	7.46	1.74	4.21
6	10.27	9.78	6.13	1.72	4.8
7	12.14	11.5	5.22	1.66	5.24
8	14.02	13.16	4.56	1.6	6.11
9	15.9	14.76	4.07	1.46	7.16
10	17.77	16.22	3.7	1.38	8.72
11	19.64	17.6	3.41	1.2	10.4
12	21.52	18.8	3.19	1.04	12.63
13	23.39	19.84	3.02	NA	15.18

Figure 5.10 provides strong evidence that fourth order FDTD is computationally advantageous over second order FDTD when simulating long dipole antennas. Figure 5.10 shows the total error in frequency of the simulated minimums in the S_{11} plots. As shown, the fourth order code exceeds a 5% error at roughly 5.5 cells per wavelength. The second order code exceeds a 5% error at roughly 10.5 cells per wavelength. 10.5/5.5 is 1.91, meaning the fourth order code can still produce accurate results with a cell size 1.91 times greater than the largest cell size that the second order code can simulate. Assuming we are working with a 3D simulation, the number of cells will be proportional to the cube of the cells size, and 1.91 cubed is 6.97. This means the fourth order simulation can use a factor of 6.97 less cells to run a simulation compared to a second order simulation. Since the fourth order code implementation does not include any additional arrays compared to second order FDTD, the cell size increase cubed is in fact the memory reduction achieved by using fourth order FDTD. This means a fourth order simulation can reduce memory needs by a factor of 6.97 compared to a second order simulation.

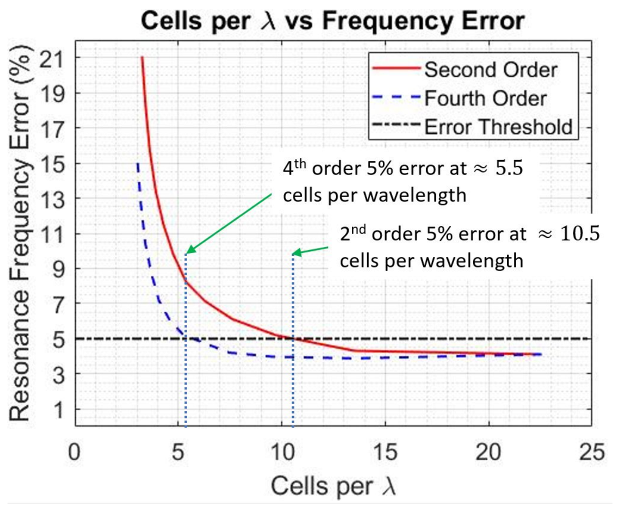


Figure 5.10 A diagram highlighting the computational memory savings of fourth order FDTD.

5.2 Dipole Antenna Arrays

The goal of this section is to simulate electrically large antenna arrays and compare the results between the second and fourth order simulations. The array element in this section is a simple thin wire dipole. Each element of the arrays shown in this section are excited with the voltage sources having the same magntude and phase.

5.2.1 Linear Arrays

The first array simulated is a linear array due to it's simplicity. Figure 5.11 diagrams how this antenna array is created.

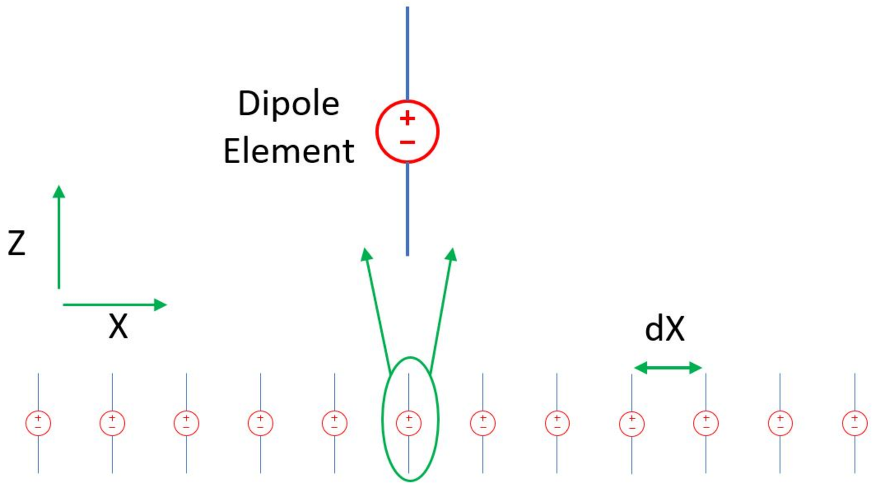
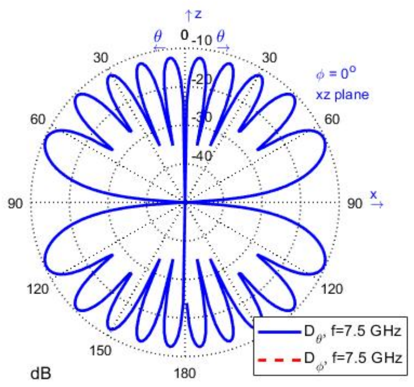


Figure 5.11 Diagram of the 10 element linear array.

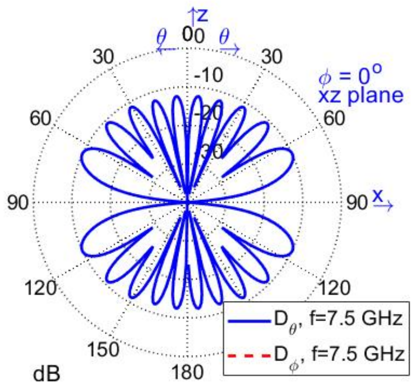
For the simulations in this section, dx = 20 mm and the length of the dipole elements are 20 mm. This configuration produces and array with a broadside array factor [28].

The computation problem space is descretized with two different cell sizes. Note 10 cells of air buffer and 8 cells of CPML are used regardless of cell size.

Figure 5.12 and Figure 5.13 show the far field radiation pattern in the XZ plane of the linear array. These results are shown for the second order and fourth order FDTD sumaltions with two different cell sizes each.



(a)



(b)

Figure 5.12 XZ radiation pattern of a 10 element linear array of dipole antennas with a cell size of 0.25 mm.

Figure 5.12(a) shows the second order simulation results and Figure 5.12(b) shows the fourth order simulation results. Figure 5.12 shows that with a cell size of 0.25 mm, there is very little difference between the second and forth order simulations.

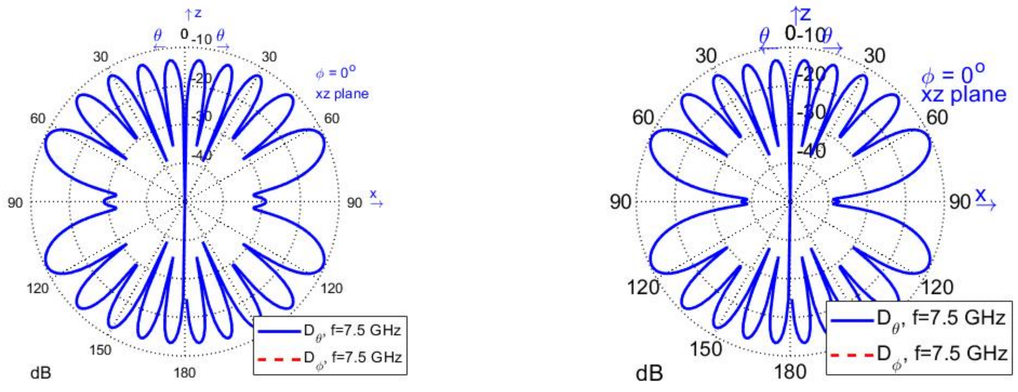


Figure 5.13 XZ radiation pattern of a 10 element linear array of dipole antennas with a cell size of 2.5 mm.

Figure 5.13(a) shows the second order simulation results and Figure 5.13(b) shows the fourth order simulation results. Figure 5.12 shows that with a cell size of 2.5 mm, there is again very little difference between the second and forth order simulations.

Figure 5.14 and Figure 5.15 show the far field radiation pattern in the XY plane of the linear array. These results are shown for the second order and fourth order FDTD sumaltions with two different cell sizes each.

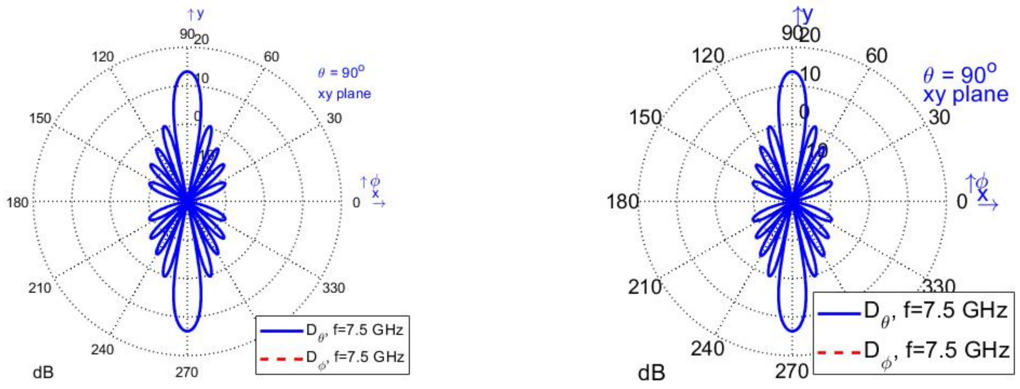
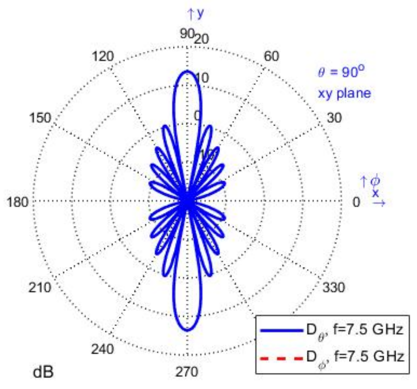
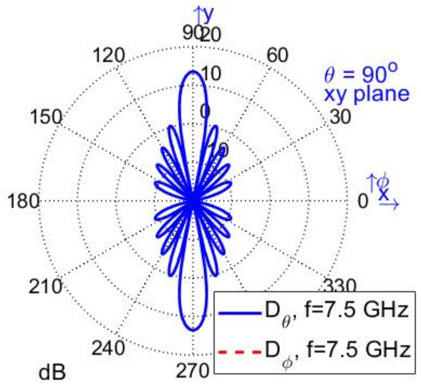


Figure 5.14 XY radiation pattern of a 10 element linear array of dipole antennas with a cell size of 0.25 mm.

Figure 5.14(a) shows the second order simulation results and Figure 5.14(b) shows the fourth order simulation results. Figure 5.14 shows that with a cell size of 0.25 mm, there is very little difference between the second and forth order simulations.



(a)



(b)

Figure 5.15 XY radiation pattern of a 10 element linear array of dipole antennas with a cell size of 2.5 mm.

Figure 5.15(a) shows the second order simulation results and Figure 5.15(b) shows the fourth order simulation results. Figure 5.14 shows that with a cell size of 2.5 mm, there is again very little difference between the second and forth order simulations.

It can be concluded from this section that fourth order FDTD is not advantagious compared to second order FDTD when simulating closely spaced (roughly $\lambda/2$) linear dipole arrays. As the dipole element used to create the linear arry in this section is the same as in section 5.1.2, the same discretization argument applies here. In order to increase the cell size to around 5 cells per wavelength to show the benefit of fourth order FDTD over second order, the dipole geometry would not be discretized by a large sufficient number of cells (roughly 10 cells per thin wire).

5.2.2 Planar Arrays

The goal of this section is to analyze the performance of the fourth order code when simulating electrically large antenna problems such as large 2D antenna arrays.

5.2.2.1 4 by 8 Array

Figure 5.16 diagrams how this 4 by 8 planar array is built.

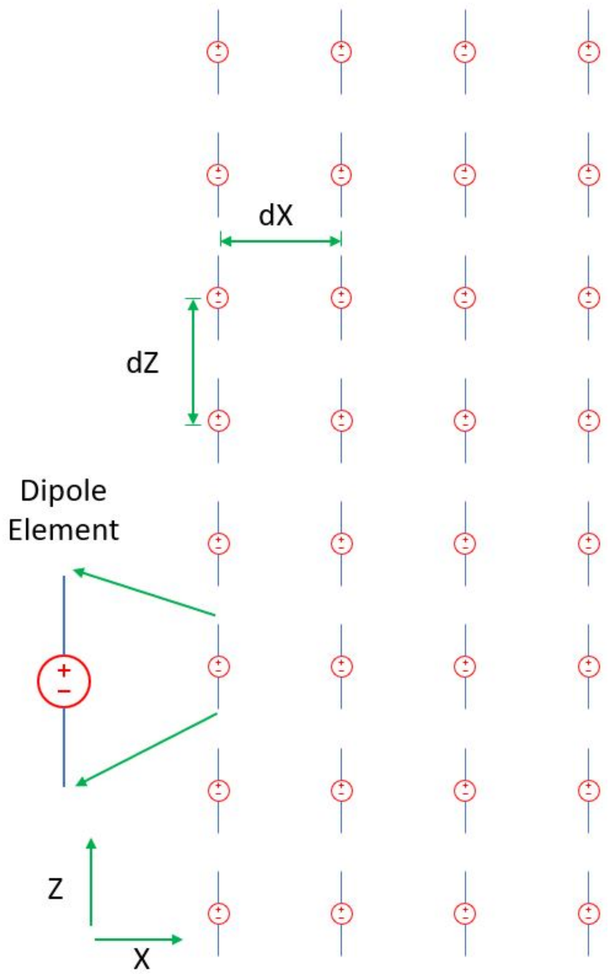
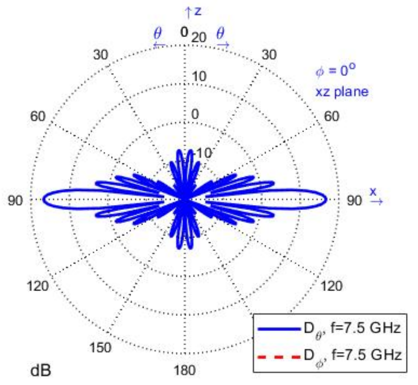
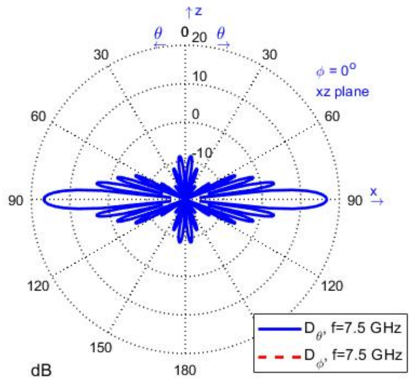


Figure 5.16 Geometry of the 4 by 8 dipole antenna planar array.

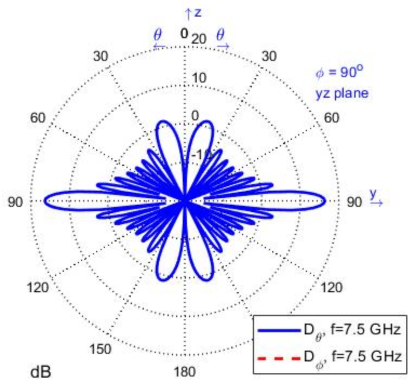
The dipole array lays in the XZ plane while the dipoles are oriented in the Z direction. The total length of each dipole is 20 mm meaning they radiate as a half wavelength dipole antennas at 7.5 GHz. The spacing of the array is set up to create a broadside array factor with $dz = dx = 40mm$ (1λ).



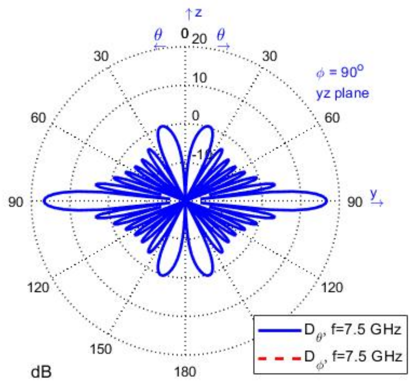
(a)



(b)



(c)



(d)

Figure 5.17 XZ and YZ radiation pattern of a 32 element (4 by 8) uniform planar array of dipole antennas with a cell size of 5mm (8 cells per λ).

Figure 5.17(a) and Figure 5.17(c) are second order simulation results while Figure 5.17(b) and Figure 5.17(d) are fourth order simulation results. Note that at a cell size of 5 mm, there is very litte difference between the second and fourth order simulation results. As the dipole element used to create the planar arry in this section is the same as in section 5.1.2. the same discretization argument applies here. In order to increase the cell size to around 5 cells per wavelength to show the benefit of fourth order FDTD over second order, the dipole geometry would not be discretized by a sufficient number of cells (roughly 10 cells per thin wire).

5.2.2.2 8 by 8 Array

Figure 5.18 diagrams how this 8 by 8 planar array is built.

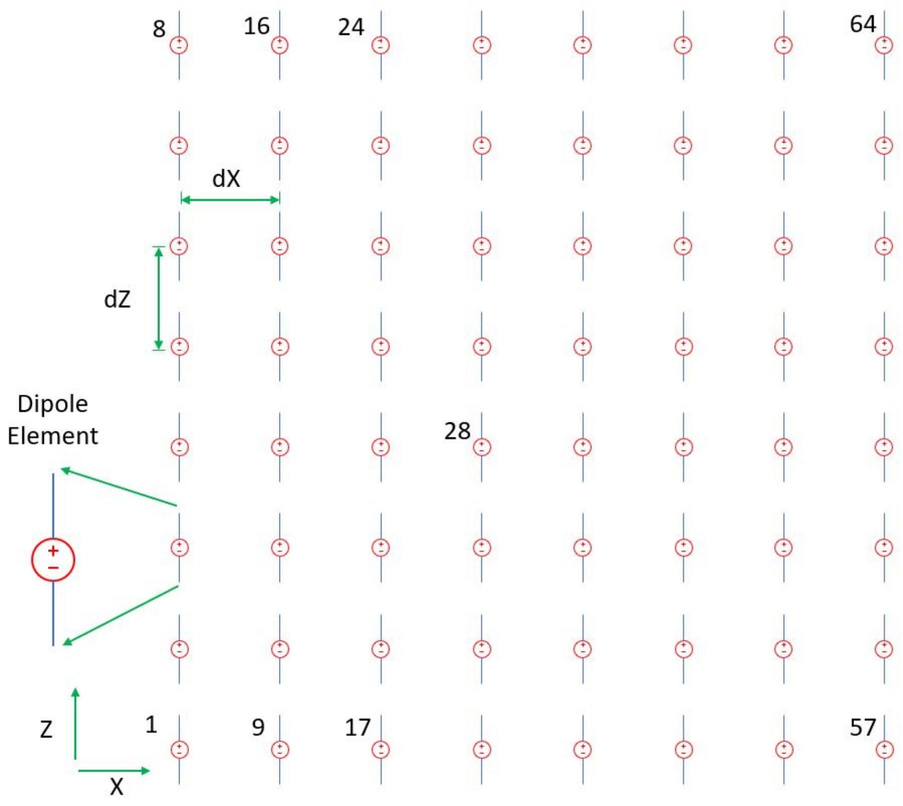
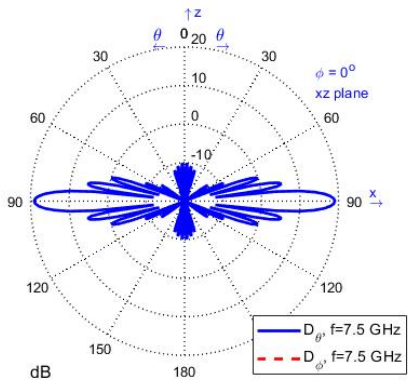
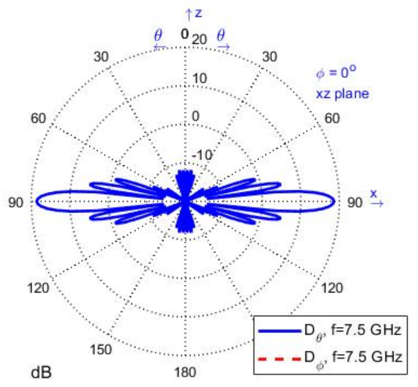


Figure 5.18 Geometry of the 8 by 8 dipole antenna planar array.

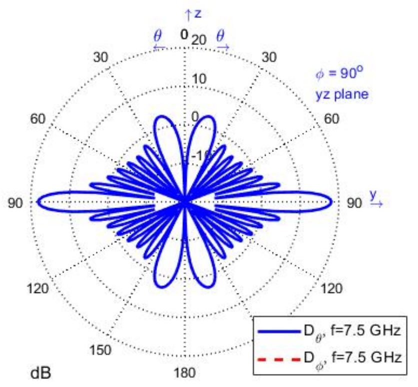
The dipole array lays in the XZ plane while the dipoles are oriented in the Z direction. The total length of each dipole is 20 mm meaning they radiate as a half wavelength dipole antennas at 7.5 GHz.



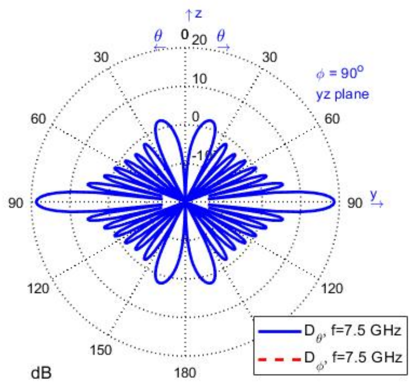
(a)



(b)



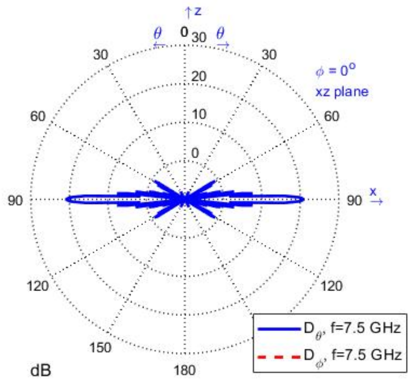
(c)



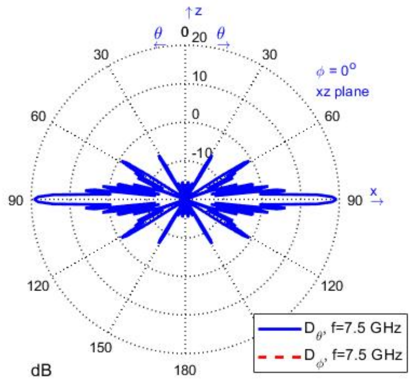
(d)

Figure 5.19 XZ and YZ radiation pattern of a 64 element (8 by 8) uniform planar array of dipole antennas with a cell size of 5mm (8 cells per λ) and a spacing of $dz = dx = 40mm$ (1 λ).

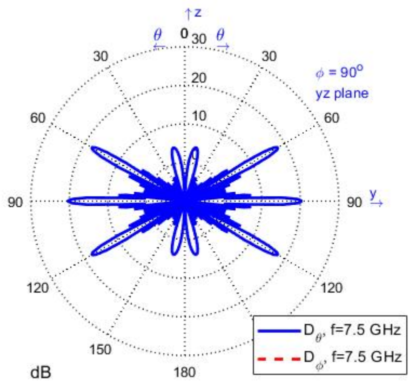
Figure 5.19(a) and Figure 5.19(c) are second order simulation results while Figure 5.19(b) and Figure 5.19(d) are fourth order simulation results.



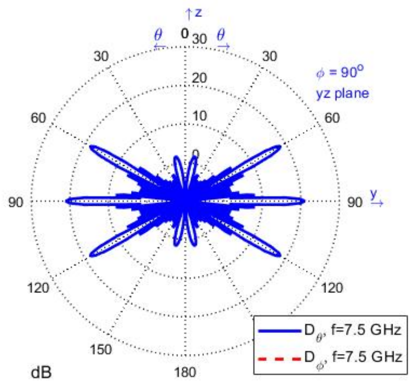
(a)



(b)



(c)

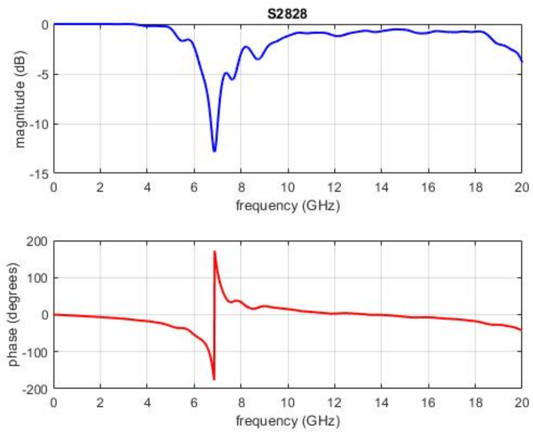


(d)

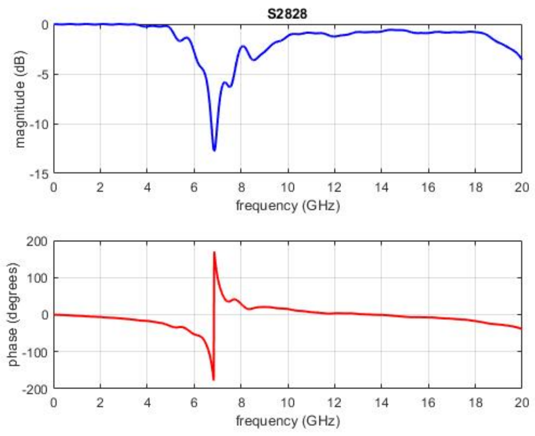
Figure 5.20 XZ and YZ radiation pattern of a 64 element (8 by 8) uniform planar array of dipole antennas with a cell size of 5mm (8 cells per λ) and spacing of $dz = dx = 80mm$ (2 λ).

Figure 5.20(a) and Figure 5.20(c) are second order simulation results while Figure 5.20(b) and Figure 5.20(d) are fourth order simulation results.

As shown in Figure 5.19 and Figure 5.20, there is little far field change between the second and fourth order simulations. In order to see more minute differences, near field parameters such as S parameters can be analyzed. Figure 5.21, Figure 5.22, and Figure 5.23 show this analysis:



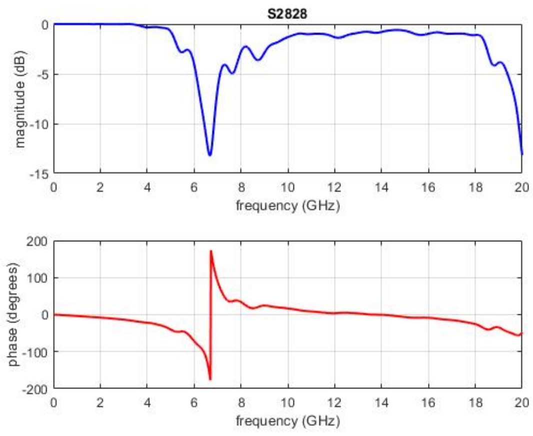
(a)



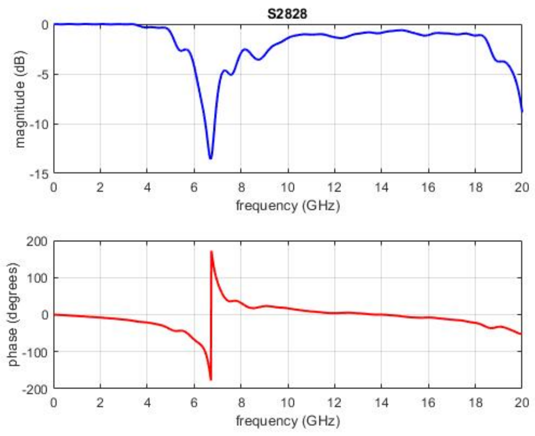
(b)

Figure 5.21 S2828 parameter of (8 by 8) uniform planar array of dipole antennas with a cell size of 0.625mm (64 cells per λ) and spacing of 80 mm (2λ).

Figure 5.21(a) is the second order simulation results and Figure 5.21(b) is the fourth order simulation results.



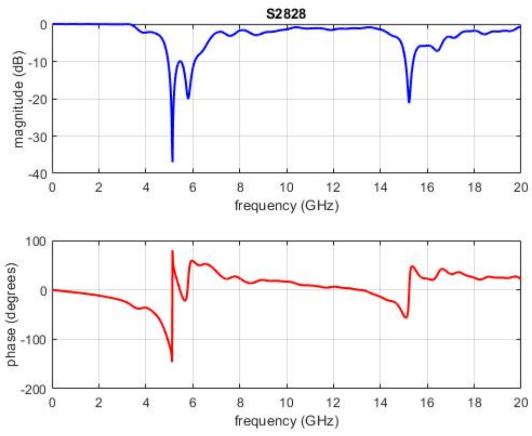
(a)



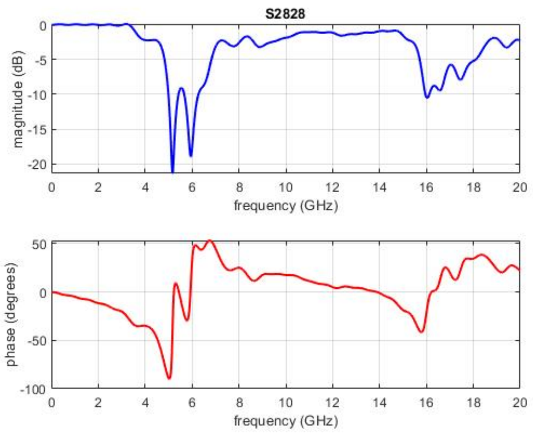
(b)

Figure 5.22 S2828 parameter of (8 by 8) uniform planar array of dipole antennas with a cell size of 1.25mm (32 cells per λ) and spacing of 80 mm (2λ).

Figure 5.22(a) is the second order simulation results and Figure 5.22(b) is the fourth order simulation results.



(a)



(b)

Figure 5.23 S2828 parameter of (8 by 8) uniform planar array of dipole antennas with a cell size of 5mm (8 cells per λ) and spacing of 80 mm (2λ).

Figure 5.23(a) is the second order simulation results and Figure 5.23(b) is the fourth order simulation results.

Figure 5.21, Figure 5.22, and Figure 5.23 compare the second order simulation to the fourth order simulation as cell size increases. As shown, there is little difference between the two simulation as cell size increases. It is shown in section 5.1.3 that the fourth order simulation should be more accurate than the second order simulation at a cell size of 8 cells per wavelength (Figure 5.23 in this analysis). However, there is nothing to suggest the fourth order simulation is more accurate than the second order simulation by looking at Figure 5.23. The reason for this is the cell size of 8 cells per wavelength is actually so large the geometry of the dipole elements are not represented by a sufficient number of cells (roughly 10 per thin wire). Since the geometry is not descitized accurately enough, the increased accuracy of the fourth order simulation cannot be seen. This same problem occurs in section 5.1.2 (half wavelength dipole) as well.

5.2.3 Discussions

Ultimately sections 5.2.1 and 5.2.2 show that the fourth order code can accurately simulate dipole antenna arrays, but they do not show that the fourth order code is advantageous over the second order code in these applications. The arrays simulated are all relatively closely spaced and built from dipole elements. Increasing the cell size to a point where fourth order FDTD will be advantageous over second order in this application forces the dipoles and the space between the dipoles to not be well represented on the Yee grid. This is why when the cell size gets large enough, the accuracy of both the second and fourth order FDTD results is negatively effected in the case of dipole arrays.

5.3 Radar Cross Section

The goal of this section is to compare the second order and fourth order simulations when calculating Radar Cross Section (or RCS). In order to perform this calculation, an incident plane wave with a Gaussian waveform is incident on the object.

5.3.1 RCS of a Dielectric Cube

In this first example, a plane wave is incident on a dielectric cube. The simulation setup matches the dielectric cube RCS simulation presented in [1] and section 4.9.1.1 of this paper. Figure 5.24 shows the dimensions of the cube and the orientation of the incident Gaussian plane wave.

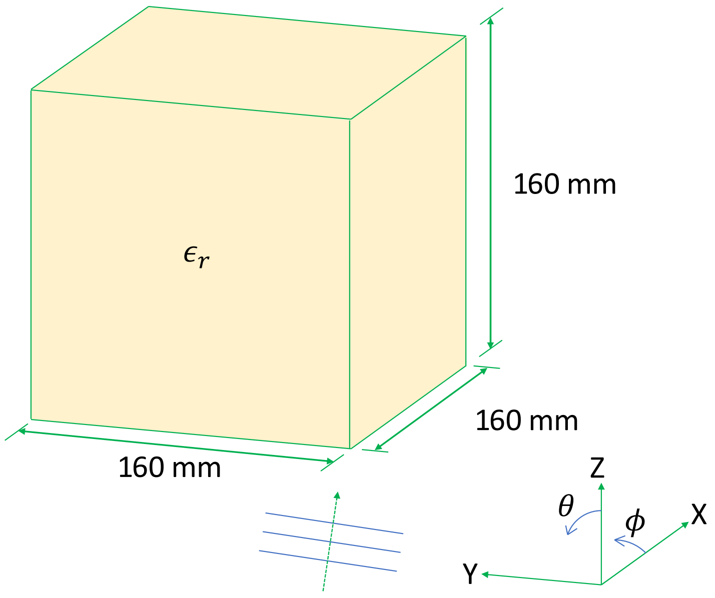
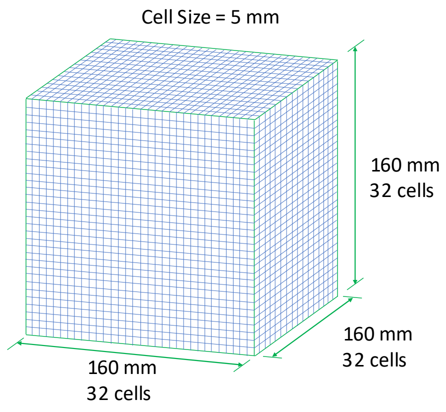


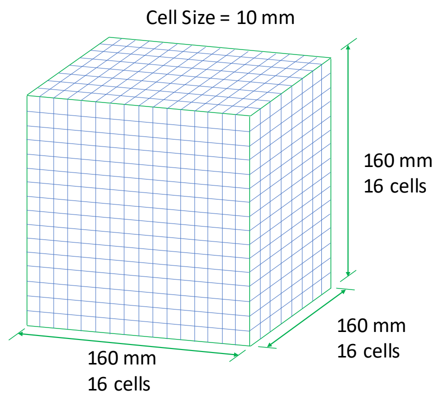
Figure 5.24 The vertical polarized incident plane wave with respect to the geometry of the dielectric cube.

Specifically, the plane wave shown in Figure 5.24 is incident on the cube at a $\theta = 45°$ and $\phi = 35°$.

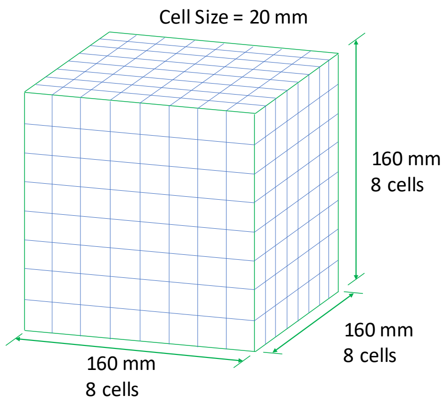
In order to properly compare the accuracy of the second and fourth order simulations, the cell size of the FDTD domain is varied. The simulation time is held constant despite a varying cell size by decreasing the number of time steps as cell size increases. Figure 5.25 shows how the cube is discretized as cell size increases.



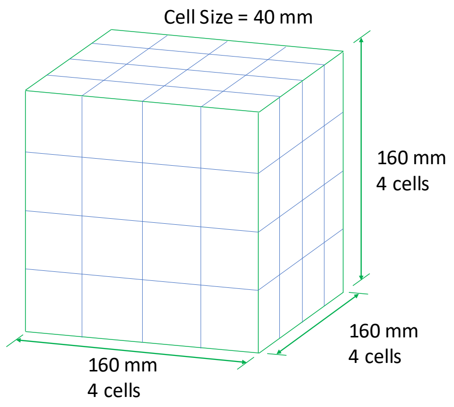
(a)



(b)



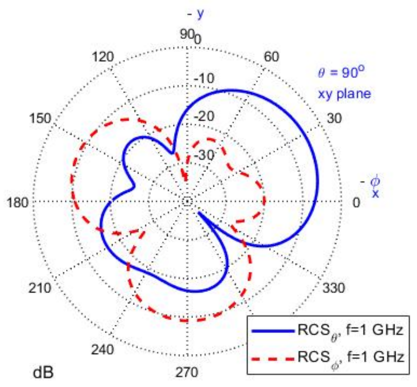
(c)



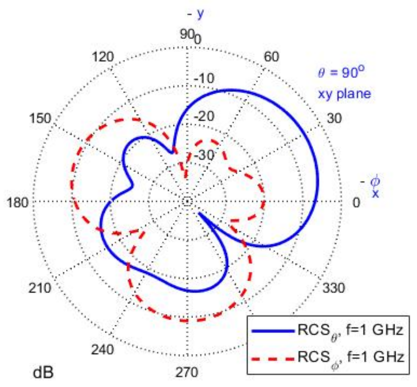
(d)

Figure 5.25 The geometry of the dielectric cube shown discetized with four different cell sizes.

Figure 5.26, Figure 5.27, Figure 5.28, and Figure 5.29 visually compare the second order and fourth order calculated RCS in the XY-plane.



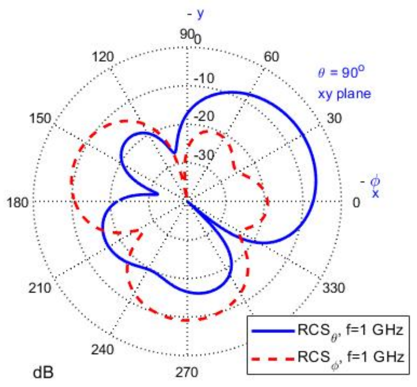
(a)



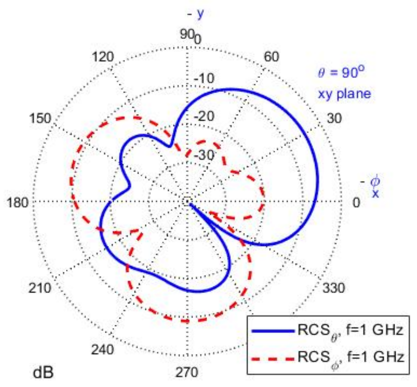
(b)

Figure 5.26 XY-plane RCS of a dielectric cube at 1 GHz with a cell size of 5 mm (60 cells per λ).

Figure 5.26(a) are the second order simulation results and Figure 5.26(b) are the fourth order simulation results.



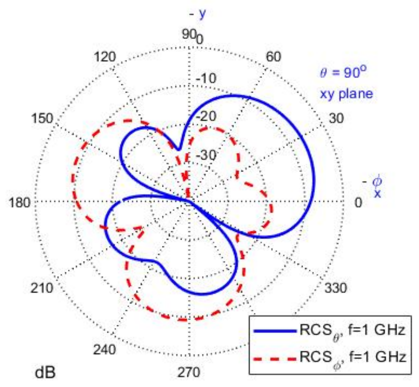
(a)



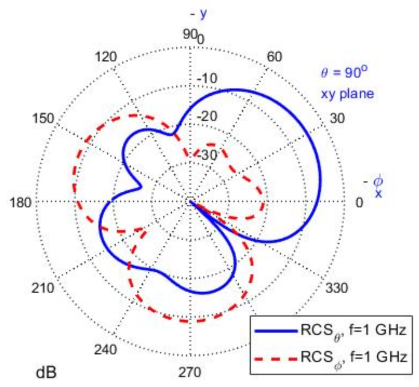
(b)

Figure 5.27 XY-plane RCS of a dielectric cube at 1 GHz with a cell size of 20 mm (15 cells per λ).

Figure 5.27(a) are the second order simulation results and Figure 5.27(b) are the fourth order simulation results.



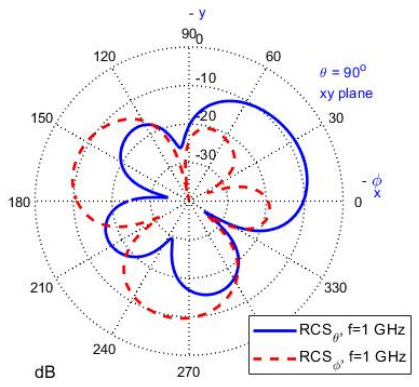
(a)



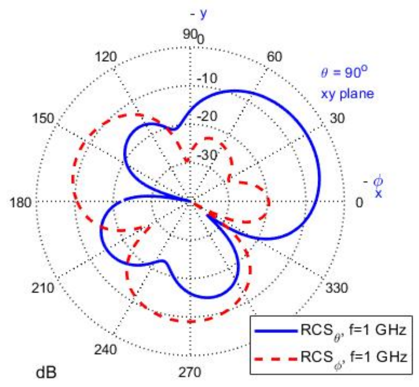
(b)

Figure 5.28 XY-plane RCS of a dielectric cube at 1 GHz with a cell size of 32 mm (9.38 cells per λ).

Figure 5.28(a) are the second order simulation results and Figure 5.28(b) are the fourth order simulation results.



(a)



(b)

Figure 5.29 XY-plane RCS of a dielectric cube at 1 GHz with a cell size of 40 mm (7.5 cells per λ).

Figure 5.29(a) are the second order simulation results and Figure 5.29(b) are the fourth order simulation results.

Taking Figure 5.26(b) to be the correct results (which also match the results in [1]), it is clear that the fourth order code maintains accuracy longer than the second order code as cell size increases. Figure 5.28 clearly shows that the fourth order results at a cell size of 32 mm still roughly match the results with a cell size of 5 mm. The second order results at a cell size of 32 mm are visually different than the fourth order results at cell size of 5 mm. Both the second and fourth order results start to look different at a cell size of 40 mm, but this is likely due to geometric errors since the cube is only discretized by 4x4x4 cells.

Although the visual comparison between second and fourth order in Figure 5.26, Figure 5.27, Figure 5.28, and Figure 5.29 is convincing, a numerical comparison can quantify the errors accurately as cell size increases.

Code Listing 5.3 shows how the cell size scale is varied. Note that the cell size scale loop in Listing 5.3 changes the cell size such that the dielectric cube is first discretized by 32x32x32 cells, then 31x31x31, then 30x30x30, then 29x29x29 all the way down to 4x4x4.

Listing 5.3: run cell size sweep

```
5.1  %initialize the matlab workspace
5.2  clear all; close all; clc;
5.3
5.4  for tony_ind = 1:29
5.5      cell_size_sweep = 32/(33-tony_ind);
5.6      fdtd_solve;
5.7      close all;
5.8  end
```

The cell size scale defined by Listing 5.3 is pulled into the problem space in line 3 of the define problem space parameters code (Listing 5.4) within the fdtd_solve code.

Listing 5.4: define problem space parameters (second order)

```
5.1  disp('defining the problem space parameters');
5.2
5.3  tonyCellScaleFactor = cell_size_sweep;
5.4
5.5  % maximum number of time steps to run FDTD simulation
5.6  number_of_time_steps = floor(5400/tonyCellScaleFactor); %4000
5.7
5.8  % A factor that determines duration of a time step
5.9  % wrt CFL limit
5.10 courant_factor = 0.9; %0.9
5.11
5.12 % A factor determining the accuracy limit of FDTD results
5.13 number_of_cells_per_wavelength = floor(20/tonyCellScaleFactor);
5.14
5.15 % Dimensions of a unit cell in x, y, and z directions (meters)
5.16 dx = (5*tonyCellScaleFactor)*1e-3;
5.17 dy = (5*tonyCellScaleFactor)*1e-3;
5.18 dz = (5*tonyCellScaleFactor)*1e-3;
5.19
5.20 % ==<boundary conditions>========
5.21 % Here we define the boundary conditions parameters
5.22 % 'pec' : perfect electric conductor
5.23 % 'cpml' : conlvolutional PML
5.24 % if cpml_number_of_cells is less than zero
5.25 % CPML extends inside of the domain rather than outwards
```

```

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 10;
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 10;
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 10;
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 10;
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 10;
boundary.cpml_number_of_cells_zn = 8;

boundary.type_zp = 'cpml';
boundary.air_buffer_number_of_cells_zp = 10;
boundary.cpml_number_of_cells_zp = 8;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1.3;
boundary.cpml_kappa_max = 7;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;

% ===<material types>============
% Here we define and initialize the arrays of material types
% eps_r   : relative permittivity
% mu_r    : relative permeability
% sigma_e : electric conductivity
% sigma_m : magnetic conductivity

% air
material_types(1).eps_r   = 1;
material_types(1).mu_r    = 1;
material_types(1).sigma_e = 0;
material_types(1).sigma_m = 0;
material_types(1).color   = [1 1 1];

% PEC : perfect electric conductor
material_types(2).eps_r   = 1;
material_types(2).mu_r    = 1;
material_types(2).sigma_e = 1e10;
material_types(2).sigma_m = 0;
material_types(2).color   = [1 0 0];

% PMC : perfect magnetic conductor
material_types(3).eps_r   = 1;
material_types(3).mu_r    = 1;
material_types(3).sigma_e = 0;
material_types(3).sigma_m = 1e10;
material_types(3).color   = [0 1 0];

```

```
5.85 % substrate
5.86 material_types(4).eps_r   = 5;
5.87 material_types(4).mu_r    = 1;
5.88 material_types(4).sigma_e = 0;
5.89 material_types(4).sigma_m = 0;
5.90 material_types(4).color   = [0 0 1];
5.91
5.92 % index of material types defining air, PEC, and PMC
5.93 material_type_index_air = 1;
5.94 material_type_index_pec = 2;
5.95 material_type_index_pmc = 3;
```

Finally, slight modifications to the calculate and display far fields code were made. These modifications are shown in Listing 5.5 and make it possible to print out data for every loop iteration of Listing 5.3. Specifically, line 143 of Listing 5.5 saves a .csv file every loop iteration.

Listing 5.5: calculate and display far fields

```
 5.1 % This file calls the routines necessary for calculating
 5.2 % farfield patterns in xy, xz, and yz plane cuts, and displays them.
 5.3 % The display can be modified as desired.
 5.4 % You will find the instructions the formats for
 5.5 % radiation pattern plots can be set by the user.
 5.6
 5.7 if number_of_farfield_frequencies == 0
 5.8     return;
 5.9 end
5.10
5.11 calculate_radiated_power;
5.12 calculate_incident_plane_wave_power;
5.13
5.14 j = sqrt(-1);
5.15 number_of_angles = 360;
5.16
5.17 % parameters used by polar plotting functions
5.18 step_size = 10;       % increment between the rings in the polar grid
5.19 Nrings = 4;           % number of rings in the polar grid
5.20 line_style1 = 'b-';   % line style for theta component
5.21 line_style2 = 'r--';  % line style for phi component
5.22 scale_type = 'dB';    % linear or dB
5.23
5.24 if incident_plane_wave.enabled == false
5.25     plot_type = 'D';
5.26 else
5.27     plot_type = 'RCS';
5.28 end
5.29 % xy plane
5.30 % ===============================================
5.31 farfield_theta = zeros(number_of_angles, 1);
5.32 farfield_phi   = zeros(number_of_angles, 1);
5.33 farfield_theta = farfield_theta + pi/2;
5.34 farfield_phi = (pi/180)*[-180:1:179].';
5.35 const_theta = 90; % used for plot
5.36
5.37 % calculate farfields
5.38 calculate_farfields_per_plane;
```

```

% plotting the farfield data
for mi=1:number_of_farfield_frequencies
 f = figure;
 pat1 = farfield_dataTheta(mi,:).';
 pat2 = farfield_dataPhi(mi,:).';

 %Tony Added
 xy_phi = pat1;
 xy_theta = pat2;

 % if scale_type is db use these, otherwise comment these two lines
 pat1 = 10*log10(pat1);
 pat2 = 10*log10(pat2);

 max_val = max(max([pat1 pat2]));
 max_val = step_size * ceil(max_val/step_size);

 legend_str1 = ...
 [plot_type '_{\theta}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];
 legend_str2 = ...
 [plot_type '_{\phi}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];

 polar_plot_constant_theta(farfield_phi,pat1,pat2,max_val, ...
        step_size, Nrings,line_style1,line_style2,const_theta, ...
        legend_str1,legend_str2,scale_type);
end

% xz plane
% ==========================================
farfield_theta = zeros(number_of_angles, 1);
farfield_phi   = zeros(number_of_angles, 1);
farfield_theta = (pi/180)*[-180:1:179].';
const_phi = 0; % used for plot

% calculate farfields
calculate_farfields_per_plane;

% plotting the farfield data
for mi=1:number_of_farfield_frequencies
 f = figure;
 pat1 = farfield_dataTheta(mi,:).';
 pat2 = farfield_dataPhi(mi,:).';

  %Tony Added
 xz_phi = pat1;
 xz_theta = pat2;

% if scale_type is db use these, otherwise comment these two lines
 pat1 = 10*log10(pat1);
 pat2 = 10*log10(pat2);

 max_val = max(max([pat1 pat2]));
 max_val = step_size * ceil(max_val/step_size);

 legend_str1 = ...
 [plot_type '_{\theta}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];
 legend_str2 = ...
 [plot_type '_{\phi}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];

 polar_plot_constant_phi(farfield_theta,pat1,pat2,max_val, ...
        step_size, Nrings,line_style1,line_style2,const_phi, ...
        legend_str1,legend_str2,scale_type);
end

% yz plane
% ===============================================
farfield_theta = zeros(number_of_angles, 1);
farfield_phi   = zeros(number_of_angles, 1);
farfield_phi = farfield_phi + pi/2;
farfield_theta = (pi/180)*[-180:1:179].';
const_phi = 90; % used for plot

% calculate farfields
calculate_farfields_per_plane;

% plotting the farfield data
for mi=1:number_of_farfield_frequencies
 f = figure;
 pat1 = farfield_dataTheta(mi,:).';
 pat2 = farfield_dataPhi(mi,:).';

  %Tony Added
 yz_phi = pat1;
 yz_theta = pat2;

% if scale_type is db use these, otherwise comment these two lines
 pat1 = 10*log10(pat1);
 pat2 = 10*log10(pat2);

 max_val = max(max([pat1 pat2]));
 max_val = step_size * ceil(max_val/step_size);

 legend_str1 = ...
 [plot_type '_{\theta}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];
 legend_str2 = ...
 [plot_type '_{\phi}, f=' num2str(farfield.frequencies(mi)*1e-9) ' GHz'];

 polar_plot_constant_phi(farfield_theta,pat1,pat2,max_val, ...
        step_size, Nrings,line_style1,line_style2,const_phi, ...
        legend_str1,legend_str2,scale_type);
end

dataOutput = [xy_phi,xy_theta,xz_phi,xz_theta,yz_phi,yz_theta];
csvwrite(strcat('C:\Users\adesp\Documents\SchoolWork\Thesis\Deliverable Code and
    Results\Chapter 5 Practical Examples\Section 5.3 Radar Cross Section\Section
    5.3.1 RCS of a Cube\dielectric_cube_data_processing\S22_output_cells',num2str(
    tonyCellScaleFactor),'.csv'),dataOutput)
```

Finally, the .csv files saved in Listing 5.5 are read and processed by Listing 5.6.

Listing 5.6: Process RMS Error

```
setfigures;

correct_results = csvread('S24_output_cells1.csv');

```

```
for i = 1:29
    cell_size_sweep(i) = 32/(33-i);
end

cells_per_lambda = 300./(5*cell_size_sweep);

average_array = mean(correct_results);
%average_array(1)

%cell_size_sweep = [4, 6.4, 8];

for i = 1:360
    correct_results2(i,1) = sqrt(correct_results(i,1)^2+correct_results(i,2)^2);
    correct_results2(i,2) = sqrt(correct_results(i,3)^2+correct_results(i,4)^2);
    correct_results2(i,3) = sqrt(correct_results(i,5)^2+correct_results(i,6)^2);
end

average_array2 = mean(correct_results2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%S22 data analysis
RMS_counter = 0;
for i = 1:length(cell_size_sweep)
    S22_test_results = csvread(strcat('S22_output_cells',num2str(cell_size_sweep(i))
        ,'.csv'));
    for j = 1:6
        for k = 1:360
            RMS_counter = RMS_counter + (S22_test_results(k,j)-correct_results(k,j))
                ^2;
        end
        total_error(i,j) = sqrt(RMS_counter/360);
        RMS_counter = 0;
    end
end


% figure;
% for i = 1:6
%     plot(cells_per_lambda,(squeeze(total_error(:,i))/average_array(i))*100)
%     hold on;
% end

figure;
plot(cells_per_lambda,(squeeze(total_error(:,1))/average_array(1))*100,'r-','
    LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error(:,2))/average_array(2))*100,'mo-','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,3))/average_array(3))*100,'g:','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,4))/average_array(4))*100,'c*-','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,5))/average_array(5))*100,'b--','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,6))/average_array(6))*100,'k*--','
    LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)


```

```
legend('xy\_phi','xy\_theta','xz\_phi','xz\_theta','yz\_phi','yz\_theta','Error
    Threshold','Location','east')
title('Second Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 50])
grid on;
grid minor;
yticks(0:5:50)

RMS_counter = 0;
for i = 1:length(cell_size_sweep)
    S22_test_results = csvread(strcat('S22_output_cells',num2str(cell_size_sweep(i))
        ,'.csv'));
    for jk = 1:360
        S22_test_results2(jk,1) = sqrt(S22_test_results(jk,1)^2+S22_test_results(jk
            ,2)^2);
        S22_test_results2(jk,2) = sqrt(S22_test_results(jk,3)^2+S22_test_results(jk
            ,4)^2);
        S22_test_results2(jk,3) = sqrt(S22_test_results(jk,5)^2+S22_test_results(jk
            ,6)^2);
    end
    for j = 1:3
        for k = 1:360
            RMS_counter = RMS_counter + (S22_test_results2(k,j)-correct_results2(k,j
                ))^2;
        end
        total_error2(i,j) = sqrt(RMS_counter/360);
        RMS_counter = 0;
    end
end

% figure;
% for i = 1:3
%     plot(cells_per_lambda,(squeeze(total_error2(:,i))/average_array2(i))*100)
%     hold on;
% end

figure;
plot(cells_per_lambda,(squeeze(total_error2(:,1))/average_array2(1))*100,'r-','
    LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error2(:,2))/average_array2(2))*100,'g:','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error2(:,3))/average_array2(3))*100,'b--','
    LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)

legend('xy','xz','yz','Error Threshold','Location','east')
title('Second Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 50])
grid on;
grid minor;
yticks(0:5:50)

%zoomed in plot for S22
figure;
```

```
plot(cells_per_lambda,(squeeze(total_error2(:,1))/average_array2(1))*100,'r-','LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error2(:,2))/average_array2(2))*100,'g:','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error2(:,3))/average_array2(3))*100,'b--','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[2,2],'k-.','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[10,10],'k-.','LineWidth',1.5)

legend('xy','xz','yz','Error Thresholds','Location','northeast')
title('Second Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 20])
grid on;
grid minor;
yticks(0:5:20)
xlim([7 40])
xticks(0:5:40)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%S24 data analysis
RMS_counter = 0;
for i = 1:length(cell_size_sweep)
    S24_test_results = csvread(strcat('S24_output_cells',num2str(cell_size_sweep(i)),'.csv'));
    for j = 1:6
        for k = 1:360
            RMS_counter = RMS_counter + (S24_test_results(k,j)-correct_results(k,j))^2;
        end
        total_error(i,j) = sqrt(RMS_counter/360);
        RMS_counter = 0;
    end
end

% figure;
% for i = 1:6
%     plot(cells_per_lambda,(squeeze(total_error(:,i))/average_array(i))*100)
%     hold on;
% end
figure;
plot(cells_per_lambda,(squeeze(total_error(:,1))/average_array(1))*100,'r-','LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error(:,2))/average_array(2))*100,'mo-','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,3))/average_array(3))*100,'g:','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,4))/average_array(4))*100,'c*-','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,5))/average_array(5))*100,'b--','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error(:,6))/average_array(6))*100,'k*--','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)

```

```

legend('xy\_phi','xy\_theta','xz\_phi','xz\_theta','yz\_phi','yz\_theta','Error
    Threshold','Location','east')
title('Fourth Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 50])
grid on;
grid minor;
yticks(0:5:50)

RMS_counter = 0;
for i = 1:length(cell_size_sweep)
    S24_test_results = csvread(strcat('S24_output_cells',num2str(cell_size_sweep(i))
        ,'.csv'));
    for jk = 1:360
        S24_test_results2(jk,1) = sqrt(S24_test_results(jk,1)^2+S24_test_results(jk
            ,2)^2);
        S24_test_results2(jk,2) = sqrt(S24_test_results(jk,3)^2+S24_test_results(jk
            ,4)^2);
        S24_test_results2(jk,3) = sqrt(S24_test_results(jk,5)^2+S24_test_results(jk
            ,6)^2);
    end
    for j = 1:3
        for k = 1:360
            RMS_counter = RMS_counter + (S24_test_results2(k,j)-correct_results2(k,j
                ))^2;
        end
        total_error2(i,j) = sqrt(RMS_counter/360);
        RMS_counter = 0;
    end
end

% figure;
% for i = 1:3
%     plot(cells_per_lambda,(squeeze(total_error2(:,i))/average_array2(i))*100)
%     hold on;
% end
figure;
plot(cells_per_lambda,(squeeze(total_error2(:,1))/average_array2(1))*100,'r-','
    LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error2(:,2))/average_array2(2))*100,'g:','
    LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error2(:,3))/average_array2(3))*100,'b--','
    LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)

legend('xy','xz','yz','Error Threshold','Location','east')
title('Fourth Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 50])
grid on;
grid minor;
yticks(0:5:50)

%zoomed in figure for S24
figure;
```

```
plot(cells_per_lambda,(squeeze(total_error2(:,1))/average_array2(1))*100,'r-','LineWidth',1.5)
hold on;
plot(cells_per_lambda,(squeeze(total_error2(:,2))/average_array2(2))*100,'g:','LineWidth',1.5)
plot(cells_per_lambda,(squeeze(total_error2(:,3))/average_array2(3))*100,'b--','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[2,2],'k-.','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[5,5],'k-.','LineWidth',1.5)
plot([0,max(cells_per_lambda)],[10,10],'k-.','LineWidth',1.5)

legend('xy','xz','yz','Error Threshold','Location','northeast')
title('Fourth Order RMS Relative Error')
ylabel('RMS Error (%)')
xlabel('Free Space Cells per \lambda')
ylim([0 20])
grid on;
grid minor;
yticks(0:5:20)
xlim([7 40])
xticks(0:5:40)
```

The results of Listing 5.6 are shown in Figure 5.30, Figure 5.31, Figure 5.32, and Figure 5.33.

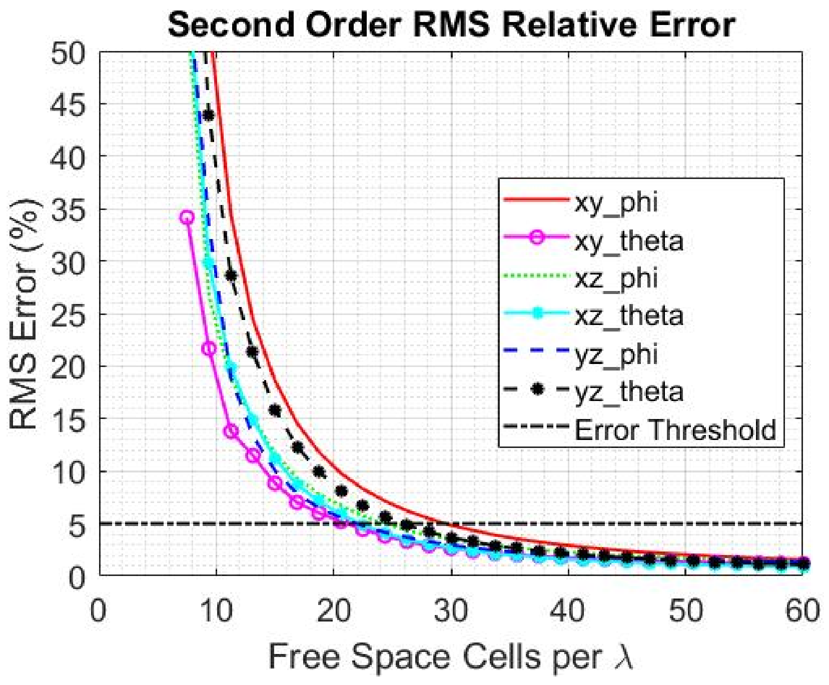


Figure 5.30 Second order theta and phi component RMS error as a function of cells per λ.

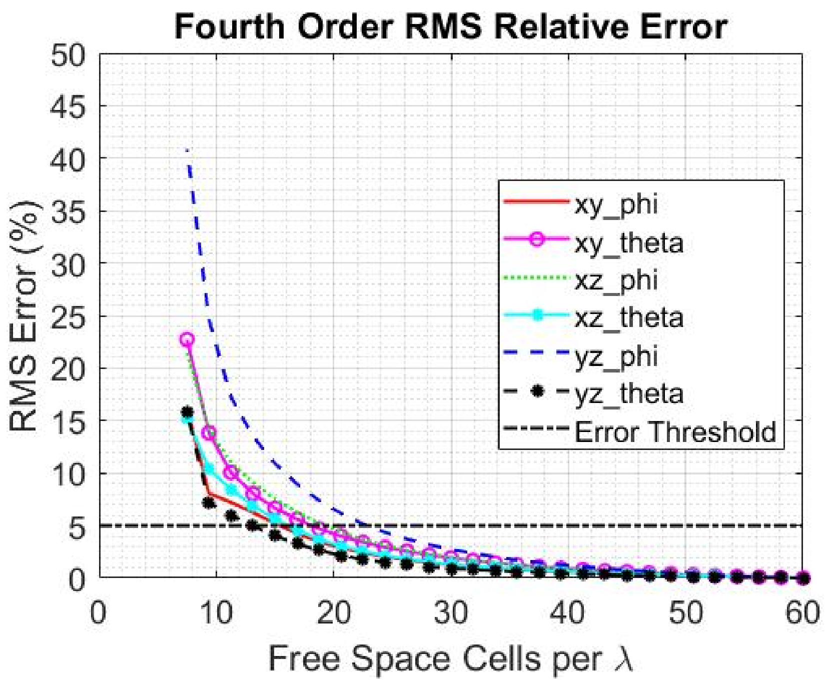


Figure 5.31 Fourth order theta and phi component RMS error as a function of cells per λ.

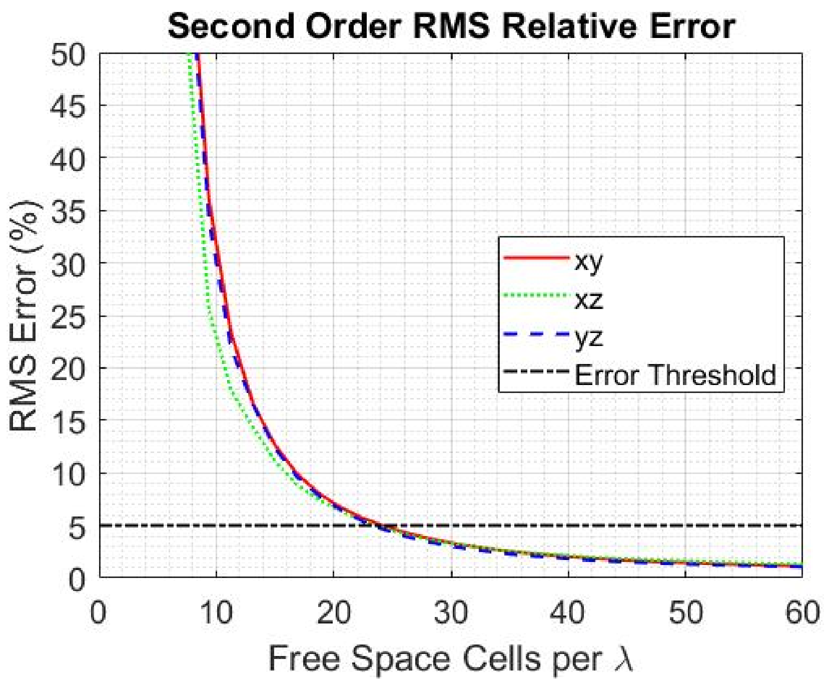


Figure 5.32 Second order field magnitude RMS error as a function of cells per λ.

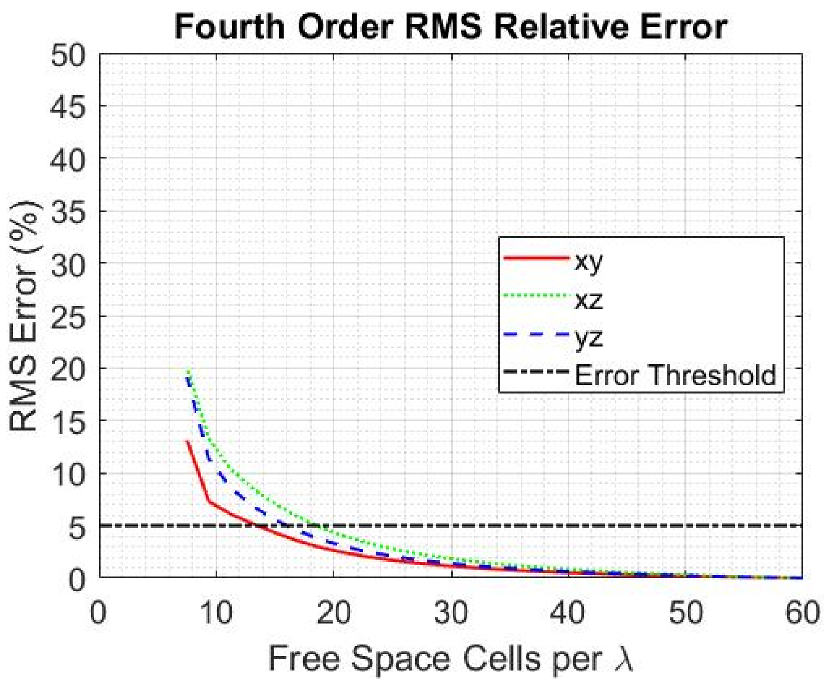


Figure 5.33 Fourth order field magnitude RMS error as a function of cells per λ.

Figure 5.34 compares the second and fourth order simulation's cell size at multiple error thresholds.

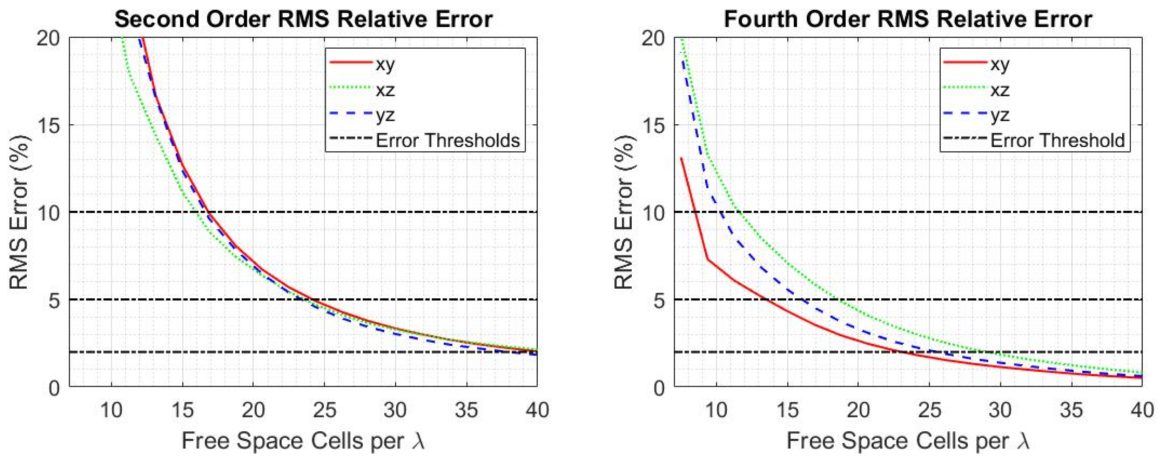


Figure 5.34 Second order and fourth order compared for multiple error thresholds.

Table 5.3 is a summary of Figure 5.34 and makes it clear fourth order FDTD is advantageous over second order FDTD in the RCS calculation of a dielectric cube. Note that since there are the same material property and field component arrays in the fourth order FDTD formulation, as there are in the second order FDTD formulation, the fourth order memory reduction compared to second order is simply the cell size ratio cubed.

Table 5.3 A summary showing cell size increase and memory reduction achieved in the RCS cube simulation by using fourth order FDTD rather than second order FDTD

RMS Error Threshold (%)	2nd Order Cells/λ	4th Order Cells/λ	Cell Size Increase Factor	3D Space Relative Memory Reduction
2	38.5	23.0	1.67	4.66
5	23.0	13.0	1.77	5.55
10	16	8.5	1.88	6.67

5.4 Computation Time for Second and Fourth Order Simulations

Sections 5.3.1 and 5.1.3 show that the fourth order FDTD formulation presented in this thesis can maintain accuracy at larger cell sizes better than the second order FDTD formulation can. The results in sections 5.3.1 and 5.1.3 show the fourth order simulation with a larger cell size requires less computational memory than second order simulations. The goal of this section is to evaluate the time it takes for a simulation to complete. The computer used to produce these results utilizes an Intel(R) Core(TM) i7-8700K CPU at 3.70GHz. The computer has a graphics card, but the FDTD MATLAB codes used in this thesis do not utilize GPUs.

Figure 5.35 shows the effect of varying the number of cells in a 3D problem space on simulation time. The problem space is the same as the dielectric cube problem presented in section 5.3.1.

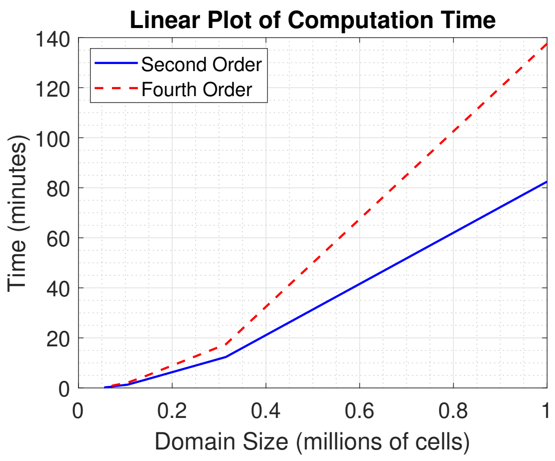


Figure 5.35 A linear plot showing the relationships between second order FDTD computation time, fourth order FDTD computation time, and domain size.

Since Figure 5.35 is hard to read due to the large range of domain size and simulation time, Figure 5.36 shows the information in a log-log plot.

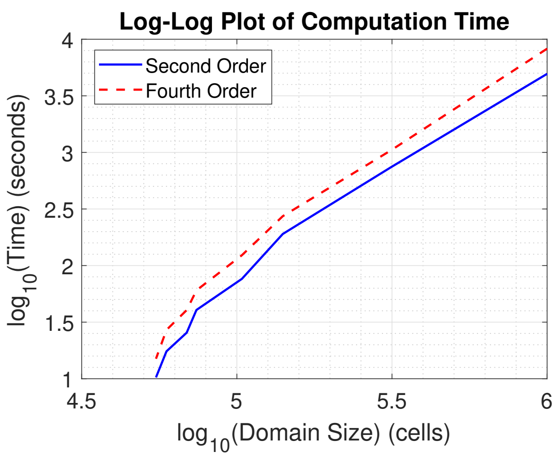


Figure 5.36 A log-log plot showing the relationships between second order FDTD computation time, fourth order FDTD computation time, and domain size.

As shown by figure Figure 5.36, the fourth order simulation consistently takes longer to simulate at any given domain size. This is due to the fact that the fourth order updating equations are more complicated than the second order updating equations. There are only two terms in the derivative approximations for the second order FDTD formulation. The fourth order FDTD formulation requires four terms in the derivative approximations. These extra terms mean the computer must complete more computations in a fourth order simulation that a second order simulation, even if the domains are the same size. Table 5.4 again shows the data presented in figure Figure 5.35 and highlights the ratio between the simulation time of the fourth order simulations and the second order simulations.

Table 5.4 A summary of the computation time needed for second and fourth order FDTD simulations with a varying domain size

Domain Size (N_x, N_y, N_z)	Domain Size Total / 1000	2^{nd} Order Simulation Time (s)	4^{th} Order Simulation Time (s)	4^{th} Order / 2^{nd} Order Time Ratio
(100,100,100)	1000	4949	8258	1.67
(68,68,68)	314	742	1047	1.41
(52,52,52)	141	190	273	1.44
(47,47,47)	104	76	123	1.62
(42,42,42)	74	41	60	1.49
(41,41,41)	69	26	40	1.58
(39,39,39)	59	17	27	1.55
(38,38,38)	55	10	15	1.46
				Average: 1.53

As is clear after looking at Table 5.4, the fourth order simulations take roughly 1.53 times longer to simulate than the second order simulations. From sections 5.1.3 and 5.3.1, it is clear the fourth order FDTD formulation can still produce accurate results with a cell size increase ratio of up to 1.91. 1.91

cubed is 6.97, which is the computational memory reduction. The computational time reduction is a little more complicated to calculate. With the increase in cells size there is also an increase in the Δt of the simulation. The increase in Δt is proportional to the cells size increase ratio and thus the number of time steps in the simulation is also reduced by the cell size increase factor. Finally, there is a computational slow down factor going from second to fourth order due to the fact fourth order FDTD has more terms in the updating equations and takes longer to simulate when the domain size is the same as a second order simulation. Equation 5.8 shows the final computational speed up ratio achieved going from second order FDTD to fourth order FDTD.

$$k = \frac{CSR^3 * DTI}{FOC} \tag{5.8}$$

Where k is the computation speed up factor achived by using fourth order FDTD, CSR is the cell size ratio allowable in fourth order FDTD compared to second order FDTD, DTI is the increase in Δt due to the cell size increase, and FOC is the extra time the fourth order FDTD simulation needs compared to a second order FDTD simulation with the same domain size. Note $DTI = CSR$ and equation 5.8 can be reduced to:

$$k = \frac{CSR^4}{FOC} \tag{5.9}$$

Substituting in the values for $CSR = 1.91$ and $FOC = 1.53$, the final k value is shown in equation 5.10.

$$k = \frac{1.91^4}{1.53} = 8.70 \tag{5.10}$$

Equation 5.10 shows that the total computation time of a fourth order simulation can be up to 8.70 times faster than a second order simulation for the same accuracy level. As multiple cell size ratios are shown in Table 5.3, each is associated with a different k value. Table 5.5 shows the calculated k values resulting from the cell size ratios shown in Table 5.3 and section 5.1.3.

Table 5.5 A summary of the computation speed up achieved by using fouth order FDTD rather than second order FDTD

Simulation Geometry	Error Threshold (%)	CSR Value	FOC Value	k Value
RCS of a Dielectric Cube	2	1.67	1.53	5.08
RCS of a Dielectric Cube	5	1.77	1.53	6.42
RCS of a Dielectric Cube	10	1.88	1.53	8.16
Long Dipole Antenna	5	1.91	1.53	8.70

Table 5.5 shows that the computational speed up achieved by using fourth order FDTD varies with error threshold and simulation geometry. However, in all cases analyzed, the fourth order FDTD code allows for significant computational speed up compared to second order FDTD.

5.5 Discussions

After reading the previous sections in this chapter it should be clear that the fourth order FDTD is computationally advantageous over second order FDTD. Section 5.1.3 shows that a fourth order simulation can use 6.97 times less memory than a second order simulation and still maintain good simulation accuracy. Section 5.3.1 shows that a fourth order simulation can use 4.66-6.67 times less memory than a second order simulation and still maintain good simulation accuracy. Section 5.4 shows that with a cell size increase factor of 1.91, a fourth order FDTD simulation will be 8.70 times faster than a second order FDTD simulation. In summary, a fourth order simulation can use 6.97 times less memory than a comparable second order simulation, and a fourth order simulation can complete an FDTD simulation 8.70 times faster than a comparable second order simulation.

CHAPTER 6
CONCLUSIONS AND FUTURE WORK

This thesis has accomplished many important aspects of achieving fully functioning fourth order accurate FDTD. However, there are still a few areas where future work is needed.

6.1 Accomplishments

This thesis demonstrates a straightforward, advantageous, and practical implementation of a fourth order FDTD formulation. The formulation allows for the integration of thin wire, active and passive circuit elements, CPML, and other special formulations that have been achieved in second order FDTD formulations [1].

In order to confirm the validity of the developed fourth order formulation, the simulations of Gaussian wave propagating in free space, a cavity resonator, the radiation from a single element dipole antenna, dipole arrays, and the RCS of a dielectric cube are performed and shown to produce the expected results. The Gaussian wave propagation in free space and the cavity resonator simulations did not require absorbing boundaries. The domain of both simulations were terminated with PEC so as to assess the validity of the updating equations with no CPML effects. Both of these fourth order simulations produced the expected results showing evidence that the fourth order updating equations are accurately derived.

In order to simulate more realistic problems such as the radiation from a dipole antenna and the RCS of a dielectric cube, fourth order CPML boundaries were formulated and implemented. The fourth order CPML implementation was first tested with a Gaussian source, and the reflections off the boundary of the fourth order CPML were compared to the second order reflections. After tuning the fourth order CPML optimization parameters, σ_f, κ_{max}, and α_{max}, the reflections off the fourth order CPML were as small as the second order CPML. This analysis shows that the implementation of fourth order CPML presented in this thesis is working correctly. Once the fourth order CPML was correctly implemented, the formulation for fourth order thin wires was implemented and shown to be accurate through the analysis of radiation from a thin wire dipole antenna. The results of the fourth order dipole antenna simulations matched the second order simulations, and both simulations matched the analytical expected results. By showing correct results for the radiation from a thin wire dipole, evidence is provided that the formulations for fourth order thin wires, fourth order CPML, fourth order voltage sources, and fourth order near to far field formulations are derived and implemented correctly.

Finally, the RCS of a dielectric cube is simulated with fourth order FDTD. In order to perform this simulation, a fourth order plane wave formulation and implementation is completed. Ultimately, the results of the fourth order RCS calculation match the second order results presented in [1]. These results not only show the plane wave is working correctly, but they also show the fourth order FDTD code can correctly handle some dielectric interfaces.

In summary, the fourth order formulation presented in this thesis is shown to correctly simulate free space propagation, CPML, resistive voltage sources, thin wires, near to far field conversions, some dielectric materials, and plane waves.

Once the fourth order formulation was shown to work correctly for many different applications, the accuracy of the fourth order simulations is analyzed against the standard second order FDTD formulation. For this analysis multiple practical electromagnetic simulations are simulated with both second order and fourth order FDTD, and the results are compared. These simulations include a thin wire dipole of varying electrical lengths, different sizes of linear and planar arrays, and the radar cross section of a cube. In order to see the increased accuracy of the fourth order simulations over the second order simulations, the cell sizes are varied for both simulations. The results are then compared for these various cell sizes. Ultimately, it is shown in the case of a long dipole antenna and the RCS calculation of a dielectric cube, that the fourth order simulation maintains better accuracy with a larger cell size than the second order simulation does. This increase in cell size is then translated to a reduction in computational resources, highlighting the benefit of fourth order FDTD. The results of this accuracy analysis are that the presented fourth order FDTD formulation can use up to 6.91 times less memory and complete an FDTD simulation up to 8.70 times faster than the standard second order FDTD formulation.

To conclude, the accomplishment of this thesis is formulating and implementing a fourth order FDTD simulator that can correctly simulate free space propagation, CPML, resistive voltage sources, thin wires, near to far field conversions, some dielectric materials, and plane waves, all the while using 6.91 times less computational memory and completing an FDTD simulation up to 8.70 times faster than the standard second order FDTD formulation.

6.2 Future Work

Though many aspects of fourth order FDTD were accomplished in this thesis, there are still a few areas where future work is needed. Fourth order CPML was fully implemented and analyzed in this thesis, but its increase in performance compared to second order CPML was not investigated. It is likely that due to the increased accuracy of the fourth order CPML, less CPML and air buffer cells could be used to achieve the same performance as a second order simulation. Using less cells for the air buffer and the CPML itself would again be a way to reduce computational memory and time.

Additionally, many techniques for the handling of PEC objects in fourth order FDTD were explored, but none were able to fully remove the low frequency errors caused by the PEC interfaces. Future work should include more analysis on the special treatment of the PEC interfaces in the fourth order FDTD code. This thesis mostly relied on mixed second/fourth order updating equations near the PEC objects to ensure the larger mask of the pure fourth order updating equations didn't cross a PEC interface. Other formulations that utilized one-sided fourth order updating equations also show promise in being able to handle these PEC interfaces [5].

This thesis presents the formulation for simulating lumped circuit elements, such as capacitors, inductors, and diodes, with fourth order FDTD. The updating equations for these circuit elements are shown to be easily derived and take the same form as the second order updating equations. However, a fourth order implementation for these components is not investigated. Future work should include integrating the formulation for these devices into the code, and then analyzing the accuracy of the fourth order FDTD simulations by comparing them to the second order simulations while varying cell size.

Finally, a FDTD formulation that incorporates subgridding with fourth order would be one of the most computationally efficient configurations for FDTD in general. Subgridding would allow the cell size to drop significantly to discretize an electromagnetic geometry [31]. In general, the subgrid would be updated with second order FDTD and thus be able to integrate PEC boundaries without the challenges of the fourth order implementation [32]. Additionally, the use of a subgrid will ensure the accurate representation of the geometry and minimize "staircasing" errors [33]. Of course, outside this subgrid, fourth order FDTD would be implemented. Fourth order FDTD would be deployed for free space regions between subgrids as well as the CPML boundaries themselves. By doing this, a small cell size accurately discretizes the geometry of the problem while a large cell size accurately simulates the free space propagation using fourth order FDTD.

REFERENCES

[1] A. Z. Elsherbeni and V. Demir. *The Finite-Difference Time-Domain Method for Electromagnetics with MATLAB® Simulation, 2nd edition.* SciTech Publishing, Edison, New Jersey, 2015.

[2] J. Fang. *Time-domain finite-difference computations for Maxwell's equations.* PhD thesis, EECS Dept., Univ. California, Berkeley, 1989.

[3] K. Hwang and A. C. Cangellaris. Computational Efficiency of Fang's Fourth-Order FDTD Schemes. *Electromagnetics*, 23(1):14, 2003.

[4] M. Hadi and M.Piket-May. A Modified FDTD (2, 4) Scheme for Modeling Electrically Large Structures with High-Phase Accuracy. *IEEE Trans. Antennas Propagat.*, 45(2):11, 1997.

[5] N. V. Kantartzis and T. D. Tsiboukis. *Higher Order FDTD Schemes for Waveguide and Antenna Structures, 1st edition.* Morgan & Claypool Publishers, San Rafael, California, 2006.

[6] M. F. Hadi and R. K. Dib. Phase-Matching the Hybrid FV24/S22 FDTD Algorithm. *Progress In Electromagnetics Research*, 72:307–323, 2007.

[7] Selçuk Paker and Levent Sevgi. FDTD Evaluation of the SAR Distribution in a Human Head Near a Mobile Cellular Phone. *Elektrik*, 6(3):227–242, 1998.

[8] Stavros V. Georgakopoulos, Craig R. Birtcher, Constantine A. Balanis, and Rosemary A. Renaut. Higher-Order Finite-Difference Schemes for Electromagnetic Radiation, Scattering, and Penetration, Part I: Theory. *IEEE Antenna and Propogation Magazine*, 44(1):9, 2002.

[9] Stavros V. Georgakopoulos, Craig R. Birtcher, Constantine A. Balanis, and Rosemary A. Renaut. Higher-Order Finite-Difference Schemes for Electromagnetic Radiation, Scattering, and Penetration, Part II: Applications. *IEEE Antenna and Propogation Magazine*, 44(2):10, 2002.

[10] Stavros V. Georgakopoulos, Craig R. Birtcher, Constantine A. Balanis, and Rosemary A. Renaut. HIRF Penetration and PED Coupling Analysis for Scaled Fuselage Models Using a Hybrid Subgrid FDTD(2,2)/FDTD(2,4) Method. *IEEE Transactions on Electromagnetic Compatibility*, 45(2):293–305, 2003.

[11] Adel M. Abdin. *Fourth Order Accurate FDTD technique and its Applications in Electromagnetics and Microwave Circuits.* PhD thesis, University of Mississippi, 1998.

[12] Hany E. Abd El-Raouf, Esam A. El-Diwani, Abd El-Hadi Ammar, and Fatma M. El-Hefnawi. A Low-Dispersion 3-D Second-Order in Time Fourth-Order in Space FDTD Scheme (M3d24). *IEEE Transactions on Electromagnetic Compatibility*, 52(7):1638–1646, 2004.

[13] Aly Fathy, Cheng Wang, Joshua Wilson, and Songnan Yang. A Fourth Order Difference Scheme for the Maxwell Equations on Yee Grid. *Journal of Hyperbolic Differential Equations*, 5(3):613–642, 2008.

[14] B.A. Al-Zohouri and M.F. Hadi. Conformal modeling of perfect conductors in the high order M24 finite-difference time-domain algorithm. *IET Microwaves, Antennas & Propagation*, 5(5):583–587, 2011.

[15] Jeffrey L. Young, Datta Gaitonde, and Joseph J. S. Shang. Toward the Construction of a Fourth-Order Difference Scheme for Transient EM Wave Simulation: Staggered Grid Approach. *IEEE Transactions on Antennas and Propagation*, 45(11):1573–1580, 1997.

[16] Konstantinos P. Prokopidis and Theodoros D. Tsiboukis. FDTD Algorithm for Microstrip Antennas with Lossy Substrates Using Higher Order Schemes. *Electromagnetics*, 24:301–315, 2004.

[17] K. P. Prokopidis and T. D. Tsiboukis. Higher-order FDTD(2,4) Scheme for Accurate Simulations in Lossy Dielectrics. *Electronics Letters*, 39(11):835–836, 2003.

[18] Amir Yefet and Peter G. Petropoulos. A Staggered Fourth-Order Accurate Explicit Finite Difference Scheme for the Time-Domain Maxwell's Equations. *Journal of Computational Physics*, 168:286–315, 2001.

[19] Kane S. Yee. Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media. *IEEE Trans on Antennas and Propagation*, 14(8):6, 1966.

[20] H.Z. Hassana, A.A. Mohamada, and G.E. Atteia. An algorithm for the finite difference approximation of derivatives with arbitrary degree and order of accuracy. *Journal of Computational and Applied Mathematics*, 236:2622–2631, 2012.

[21] Konstantinos P. Prokopidis. A higher-order spatial FDTD scheme with CFS PML for 3D numerical simulation of wave propagation in cold plasma. *Cornell University*, pages 1–7, 2013.

[22] K. P. Prokopidis, E. P. Kosmidou, and T. D. Tsiboukis. An FDTD Algorithm for Wave Propagation in Dispersive Media Using Higher-Order Schemes. *J. of Electromagn. Waves and Appl.*, 18(9): 1171–1194, 2004.

[23] Mohammed F. Hadi. *A Modified FDTD (2,4) Scheme for Modeling Electrically Large Structures with High Phase Accuracy.* PhD thesis, University of Colorado at Boulder, 1992.

[24] Mohammed F. Hadi. A Finite Volumes-Based 3-D Low Dispersion FDTD Algorithm. *IEEE Trans on Antennas and Propagation*, 55(8):7, 2007.

[25] J. Roden and S. Gedney. Convolution PML (CPML): an efficient FDTD implementation of the CFS-PML for arbitrary media. *Microwave and Optical Technology Letters*, 27(5):6, 2000.

[26] Stephen D. Gedney. An Anisotropic Perfectly Matched Layer-Absorbing Medium for the Truncation of FDTD Lattices. *IEEE Trans on Antennas and Propagation*, 44(12):1630–1639, 1996.

[27] Wei-Yang and Chih-Wen Kuo. Handling of the Perfect Electric Conductor in Higher-Order FDTD Method. *Proceedings of APMC2001*, pages 203–207, 2001.

[28] C. A. Balanis. *Antenna Theory Analysis and Design, 3rd edition.* A John Wiley & Sons, Inc., Hoboken, New Jersey, 2005.

[29] Mohammed F. Hadi and Rabie K. Dib. Eliminating Interface Reflections in Hybrid Low-Dispersion FDTD Algorithms. *ACES Journal*, 22(3):9, 2004.

[30] Kyu-Pyung Hwang and Andreas C. Cangellaris. Effective Permittivities for Second-Order Accurate FDTD Equations at Dielectric Interfaces. *IEEE Microwave and Wireless Components Letters*, 11(4): 158–160, 2001.

[31] Madison Le, Mohammed Hadi, and Atef Elsherbeni. Quantifying Subgridding Errors when Modeling Multiscale Structures with FDTD. *ACES*, S23:2, 2019.

[32] Stavros V. Georgakopoulos, Rosemary A. Renaut, Constantine A. Balanis, and Craig R. Birtcher. A Hybrid Fourth-Order FDTD Utilizing a Second-Order FDTD Subgrid. *IEEE Microwave and Wireless Components Letters*, 11(11):462–464, 2001.

[33] A. Akyurtlu, D. H. Werner, V. Veremey, D. J. Steich, and K. Aydin. Staircasing Errors in FDTD at an Air–Dielectric Interface. *IEEE Microwave and Guided Wave Letters*, 9(11):444–446, 1999.