ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

Second Edition

DENNIS M. SULLIVAN

IEEE PRESS

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Sullivan, Dennis Michael, 1949-
Electromagnetic simulation using the FDTD method / Dennis M. Sullivan. – Second edition.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-45939-3 (cloth)
1. Electromagnetism–Computer simulation. 2. Finite differences. 3. Time-domain analysis. I. Title.
QC760.S92 2013
537.01–dc23
2012050029

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To
Sully and Jane

CONTENTS

PREFACE

The purpose of the second edition of this book remains the same as that of the first edition: to enable the reader to learn and use the Finite-Difference Time Domain (FDTD) method in a manageable amount of time. For this reason, the first four chapters are fundamentally the same. The goal of these four chapters is to take the reader through one-, two-, and three-dimensional FDTD simulation and, at the same time, present the techniques for dealing with more complicated media. In addition, some basic applications of signal processing theory are explained to enhance the effectiveness of FDTD simulation.

Chapter 5 contains three examples of FDTD simulation. This chapter is by no means comprehensive and does not include most of the applications for FDTD that have been found over the years. They are examples from the author's own research, which are hopefully representative of the flexibility of the FDTD method for electromagnetic applications.

Chapter 6 is a very short introduction to the simulation of the Schrödinger equation using FDTD simulation. The Schrödinger equation is at the heart of quantum mechanics. FDTD is not yet widely used in quantum simulation, but compared with the other methods being used in quantum simulation, it is likely that FDTD will play a substantial role in the future.

GUIDE TO THE BOOK

PURPOSE

This book has one purpose only: it enables the reader or student to learn and do three-dimensional electromagnetic simulation using the finite-difference time domain (FDTD) method. It does not attempt to explain the theory of FDTD simulation in great detail. It is not a survey of all possible approaches to the FDTD method nor is it a "cookbook" of applications. It is aimed at those who would like to learn and do FDTD simulation in a reasonable amount of time.

FORMAT

This book is tutorial in nature. Every chapter attempts to address an additional level of complexity. The text increases in complexity in two major ways:

Dimension of Simulation	Type of Material
One-dimensional	Free space
Two-dimensional	Complex dielectric material
Three-dimensional	Frequency-dependent material

The first section of Chapter 1 is one-dimensional simulation in free space. From there, the chapters progress to more complicated media. In Chapter 2, the simulation of frequency-dependent media is addressed. Chapter 3 introduces two-dimensional simulation, including the simulation of plane waves and how to

implement the perfectly matched layer. Chapter 4 introduces three-dimensional simulation. Chapter 5 consists of three sections, each with a different example of FDTD simulation. Chapter 6 is a short introduction to quantum simulation using the FDTD method.

SPECIFIC CHOICES DEALING WITH SOME TOPICS

There are many ways to handle individual topics having to do with FDTD simulation. This book does not attempt to address all of them. In most cases, a single approach is taken and used throughout the book for the sake of clarity. My philosophy is that when first learning the FDTD method, it is better to learn one specific approach and learn it well rather than to be confused by switching to different approaches. In most cases, the approach being taught is the author's preference. This does not make it the only approach or even the best; it is just the approach that this author has found to be effective. In particular, the following are some of the choices that have been made.

1. *The Use of Normalized Units*. Maxwell's equations have been normalized by substituting

$$\widetilde{E} = \sqrt{\frac{\varepsilon_0}{\mu_0}}\boldsymbol{E}.$$

   This is a system similar to the *Gaussian units*, which is frequently used by physicists. The reason for using it here is the simplicity in the formulation. The E and the H fields have the same order of magnitude. This has an advantage in formulating the perfectly matched layer (PML), which is a crucial part of FDTD simulation.
2. *Maxwell's Equations with the Flux Density*. There is some leeway in forming the time domain Maxwell's equations from which the FDTD formulation is developed. The following is used in Chapter 1:

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_0}\nabla \times \boldsymbol{H} - \frac{\sigma}{\varepsilon_0}\boldsymbol{E} \tag{1}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0}\nabla \times \boldsymbol{E}. \tag{2}$$

   This is a straightforward formulation and among those commonly used. However, by Chapter 2, the following formulation using the flux density is adopted:

$$\frac{\partial \boldsymbol{D}}{\partial t} = \frac{1}{\varepsilon_0}\nabla \times \boldsymbol{H}, \tag{3}$$

$$\boldsymbol{D} = \varepsilon \boldsymbol{E}, \tag{4}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{5}$$

In this formulation, it is assumed that the materials being simulated are nonmagnetic; that is, $\boldsymbol{H} = (1/\mu_0)\boldsymbol{B}$. However, we will be dealing with a broad range of dielectric properties, so Eq. (4) could be a complicated convolution. There is a reason for using this formulation: Eq. (3) and Eq. (5) remain the same regardless of the material; any complicated mathematics stemming from the material lies in Eq. (4). We will see that the solution of Eq. (4) can be looked upon as a digital filtering problem. In fact, the use of signal processing techniques in FDTD simulation will be a recurring theme in this book.

Z TRANSFORMS

As mentioned above, the solution of Eq. (4) for most complicated material can be viewed as a digital filtering problem. This being the case, the most direct approach to solve the problem is to take Eq. (4) into the Z domain. Z transforms are a regular part of electrical engineering education, but not that of physicists, mathematicians, and others. In teaching a class on FDTD simulation, I teach a little Z transform theory so when we reach the sections on complicated dispersive materials, the student are ready to apply Z transforms. This has two distinct advantages: (1) Electrical engineering students have another application of Z transforms to strengthen their understanding of signal processing; and (2) physics students and others now know and can use Z transforms, something that had not usually been part of their formal education. Based on my positive experience, I would encourage anyone using this book when teaching an FDTD course to consider this approach. However, I have left the option open to simulate dispersive methods with other techniques. The sections on Z transforms are optional and may be skipped. Appendix A on Z transforms is provided.

PROGRAMMING EXERCISES

The philosophy behind this book is that the reader will learn by doing. Therefore, the majority of exercises involve programming. Each chapter has one or more FDTD programs written in C. If there is more than one program per chapter, typically only the first will be a complete program listing. The subsequent programs will not be complete, but will only show changes as compared to the first program. For instance, Section 1.1 describes one-dimensional FDTD simulation in free space. The program fd1d_1.1.c at the end of the chapter is a simulation of a pulse in free space. Section 1.3 describes how a simple absorbing boundary condition is implemented. The listing of fd1d_1.2.c is not a complete program, but shows the changes necessary to fd1d_1.1.c to implement the boundary condition. Furthermore, important lines of code are highlighted in boldface.

PROGRAMMING LANGUAGE

The programs at the end of each chapter are written in the C programming language, as this was the case in the first edition of this book. The reason for this is almost universal availability of C compilers on UNIX workstations, and the fact that most potential readers will at least be able to understand C. Presumably, if another language is preferred by the reader, it will not be that difficult to transcribe the given C program to that language. A word of warning: MATLAB is a very attractive for simulation because of its graphics capability. However, I have found it to be too slow for anything except one-dimensional simulation. In contrast, I always use FORTRAN for my own applications. I have found it to be fastest on any computing platform that I have used.

DENNIS M. SULLIVAN

Professor of Electrical and Computer Engineering
University of Idaho

1

ONE-DIMENSIONAL SIMULATION WITH THE FDTD METHOD

This chapter is a step-by-step introduction to the FDTD (finite-difference time domain) method. It begins with the simplest possible problem, the simulation of a pulse propagating in free space in one dimension. This example is used to illustrate the FDTD formulation. Subsequent sections lead to formulations for more complicated media.

1.1 ONE-DIMENSIONAL FREE SPACE SIMULATION

The time-dependent Maxwell's curl equations for free space are

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_0} \nabla \times \boldsymbol{H} \tag{1.1a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.1b}$$

$\boldsymbol{E}$ and $\boldsymbol{H}$ are vectors in three dimensions, so in general, Eq. (1.1a) and (1.1b) represent three equations each. We will start with a simple one-dimensional case using only E_x and H_y, so Eq. (1.1a) and (1.1b) become

$$\frac{\partial E_x}{\partial t} = -\frac{1}{\varepsilon_0} \frac{\partial H_y}{\partial z} \tag{1.2a}$$

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

$$\frac{\partial H_y}{\partial t} = -\frac{1}{\mu_0}\frac{\partial E_x}{\partial z}. \tag{1.2b}$$

These are the equations of a plane wave with the electric field oriented in the x direction, the magnetic field oriented in the y direction, and the wave traveling in the z direction.

Taking the central difference approximations for both the temporal and spatial derivates gives

$$\frac{E_x^{n+1/2}(k) - E_x^{n-1/2}(k)}{\Delta t} = -\frac{1}{\varepsilon_0}\frac{H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)}{\Delta x} \tag{1.3a}$$

$$\frac{H_y^{n+1}\left(k+\frac{1}{2}\right) - H_y^n\left(k+\frac{1}{2}\right)}{\Delta t} = -\frac{1}{\mu_0}\frac{E_x^{n+1.2}(k+1) - E_x^{n+1.2}(k)}{\Delta x}. \tag{1.3b}$$

In these two equations, time is specified by the superscripts, that is, "n" actually means a time step $t = \Delta t \cdot n$. Remember, we have to discretize everything for formulation into the computer. The term $n+1$ means one time step later. The terms in parentheses represent distance, that is, "k" actually means the distance $z = \Delta x \cdot k$. (It might seem more sensible to use Δz as the incremental step, because in this case we are going in the z direction. However, we will use Δx because it is so commonly used for a spatial increment.) The formulation of Eq. (1.3a) and (1.3b) assumes that the E and H fields are interleaved in both space and time. H uses the arguments $k+1/2$ and $k-1/2$ to indicate that the H field values are assumed to be located between the E field values. This is illustrated in Fig. 1.1. Similarly, the $n+1/2$ or $n-1/2$ superscript indicates that it occurs slightly after or before n, respectively. Equation (1.3a) and (1.3b) can be rearranged in an iterative algorithm:

$$E_x^{n+1/2}(k) = E_x^{n-1/2}(k) - \frac{\Delta t}{\varepsilon_0 \cdot \Delta x}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right] \tag{1.4a}$$

$$H_y^{n+1}\left(k+\frac{1}{2}\right) = H_y^n\left(k+\frac{1}{2}\right) - \frac{\Delta t}{\mu_0 \Delta x}[E_x^{n+1.2}(k+1) - E_x^{n+1.2}(k)]. \tag{1.4b}$$

Notice that the calculations are interleaved in space and time. In Eq. (1.4a), for example, the new value of E_x is calculated from the previous value of E_x and the most recent values of H_y. This is the fundamental paradigm of the FDTD method (1).

Equation (1.4a) and Equation (1.4b) are very similar, but because ε_0 and μ_0 differ by several orders of magnitude, E_x and H_y will differ by several orders of magnitude. This is circumvented by making the following change of variables (2):

$$\widetilde{E} = \sqrt{\frac{\varepsilon_0}{\mu_0}}E. \tag{1.5}$$

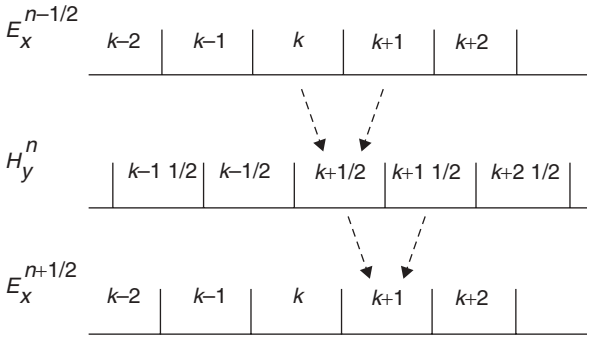

Figure 1.1 Interleaving of the E and H fields in space and time in the FDTD formulation. To calculate H_y, for instance, the neighboring values of E_x at k and $k+1$ are needed. Similarly, to calculate E_x, the values of H_y at $k+1/2$ and $k+1\,1/2$ are needed.

Substituting this into Eq. (1.4a) and (1.4b) gives

$$\widetilde{E}_x^{n+1/2}(k) = \widetilde{E}_x^{n-1/2}(k) - \frac{\Delta t}{\sqrt{\varepsilon_0\mu_0}\Delta x}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right] \tag{1.6a}$$

$$H_y^{n+1}\left(k+\frac{1}{2}\right) = H_y^n\left(k+\frac{1}{2}\right) - \frac{\Delta t}{\sqrt{\varepsilon_0\mu_0}\Delta x}\left[\widetilde{E}_x^{n+1.2}(k+1) - \widetilde{E}_x^{n+1.2}(k)\right]. \tag{1.6b}$$

Once the cell size Δx is chosen, then the time step Δt is determined by

$$\Delta t = \frac{\Delta x}{2 \cdot c_0}, \tag{1.7}$$

where c_0 is the speed of light in free space. (The reason for this is explained later.) Therefore,

$$\frac{\Delta t}{\sqrt{\varepsilon_0\mu_0}\cdot\Delta x} = c_0\frac{\left(\frac{\Delta x}{2}\right)}{\Delta x} = \frac{1}{2}. \tag{1.8}$$

Rewriting Eq. (1.6a) and (1.6b) in C computer code gives the following:

```
ex[k]=ex[k]+0.5*(hy[k-1]-hy[k])          (1.9a)
hy[k]=hy[k]+0.5*(ex[k]-ex[k+1}).         (1.9b)
```

Note that the n or $n+1/2$ or $n-1/2$ in the superscripts is gone. Time is implicit in the FDTD method. In Eq. (1.9a), the `ex` on the right-hand side is the previous value at $n-1/2$ and the `ex` on the left-hand side is the new value, $n+1/2$,

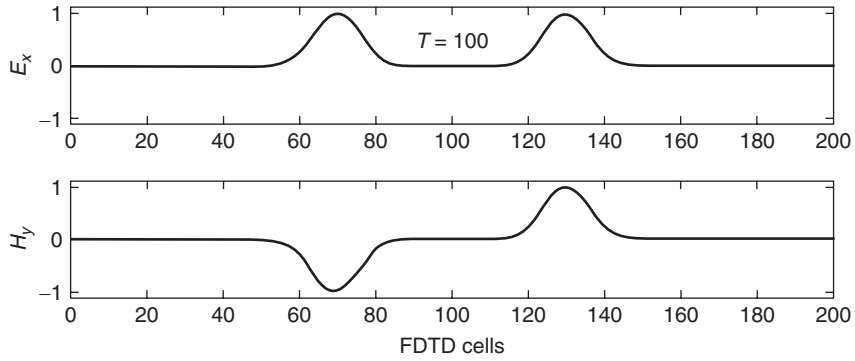

Figure 1.2 FDTD simulation of a pulse in free space after 100 time steps. The pulse originated in the center and travels outward.

which is being calculated. Position, however, is explicit. The only difference is that $k + 1/2$ and $k - 1/2$ are rounded off to k and $k - 1$ in order to specify a position in an array in the program.

The program fd1d_1.1.c at the end of the chapter is a simple one-dimensional FDTD program. It generates a Gaussian pulse in the center of the problem space, and the pulse propagates away in both directions as seen in Fig. 1.2. The E_x field is positive in both directions, but the H_y field is negative in the negative directions. The following points are worth noting about the program:

1. The E_x and H_y values are calculated by separate loops and employ the interleaving described above.
2. After the E_x values are calculated, the source is calculated. This is done by simply specifying a value of E_x at the point `k=kc` and overriding what was previously calculated. This is referred to as a *hard source* because a specific value is imposed on the FDTD grid.

PROBLEM SET 1.1

1. Get the program fd1d_1.1c running. What happens when the pulse hits the end of the array? Why?

2. Modify the program so it has two sources, one at `kc -20` and one at `kc +20`. (Notice that kc is the center of the problem space.) What happens when the pulses meet? Explain this from basic EM (electromagnetic) theory.

3. Instead of E_x as the source, use H_y at `k=kc` as the source. What difference does it make? Try a two-point magnetic source at `kc -1` and `kc` such that `hy[kc-1]=-hy[kc]`. What does this look like? To what does it correspond physically?

1.2 STABILITY AND THE FDTD METHOD

Let us return to the discussion of how we determine the time step. An EM wave propagating in free space cannot go faster than the speed of light. To propagate a distance of one cell requires a minimum time of $\Delta t = \Delta x/c_0$. With a two-dimensional simulation, we must allow for the propagation in the diagonal direction, which brings the requirement to $\Delta t = \Delta x/(\sqrt{2}c_0)$. Obviously, a three-dimensional simulation requires $\Delta t = \Delta x/(\sqrt{3}c_0)$. This is summarized by the well-known *Courant Condition* (3, 4):

$$\Delta t = \frac{\Delta x}{\sqrt{n}c_0}, \tag{1.10}$$

where n is the dimension of the simulation. Unless otherwise specified, throughout this book, we will determine Δt by

$$\Delta t = \frac{\Delta x}{2c_0}. \tag{1.11}$$

This is not necessarily the best formula; we will use it for simplicity.

PROBLEM SET 1.2

1. In fd1d_1.1.c, go to the governing equation, Eq. (1.9a) and (1.9b), and change the factor 0.5 to 1.0. What happens? Change it to 1.1. Now what happens? Change it to 0.25 and see what happens.

1.3 THE ABSORBING BOUNDARY CONDITION IN ONE DIMENSION

Absorbing boundary conditions are necessary to keep outgoing E and H fields from being reflected back into the problem space. Normally, in calculating the E field, we need to know the surrounding H values; this is a fundamental assumption of the FDTD method. At the edge of the problem space, we will not have the value of one side. However, we have an advantage because we know that the fields at the edge must be propagating outward. We will use this fact to estimate the value at the end by using the value next to it (5).

Suppose we are looking for a boundary condition at the end where $k = 0$. If a wave is going toward a boundary in free space, it is traveling at c_0, the speed of light. So in one time step of the FDTD algorithm, it travels

$$\text{Distance} = c_0 \cdot \Delta t = c_0 \cdot \frac{\Delta x}{2 \cdot c_0} = \frac{\Delta x}{2}.$$

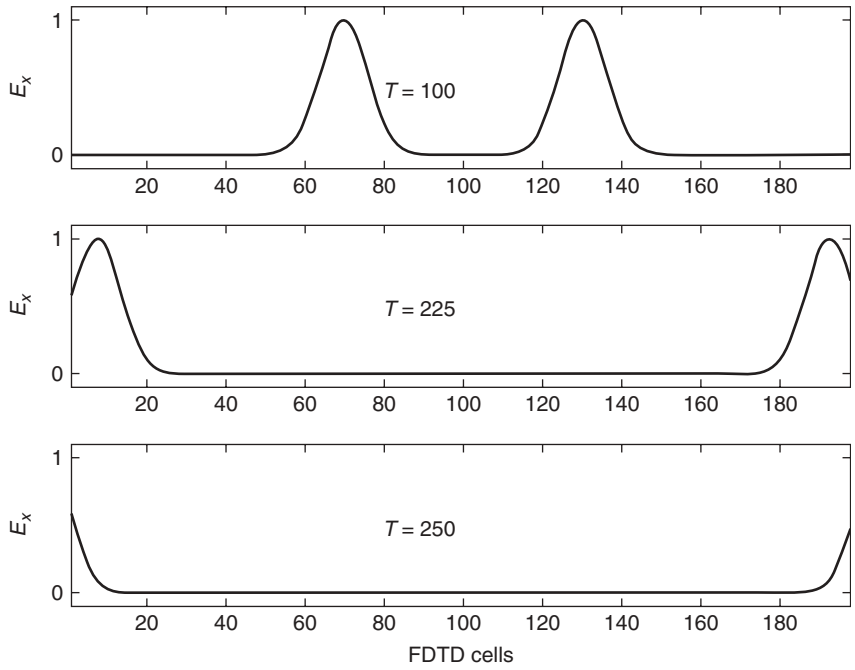

Figure 1.3 Simulation of an FDTD program with absorbing boundary conditions. Notice that the pulse is absorbed at the edges without reflecting anything back.

This equation basically shows that it takes two time steps for the field to cross one cell. A commonsense approach tells us that an acceptable boundary condition might be

$$E_x^n(0) = E_x^{n-2}(1). \tag{1.12}$$

It is relatively easy to implement this. Simply store a value of $E_x(1)$ for two time steps and then put it in $E_x(0)$. Boundary conditions such as these have been implemented at both ends of the E_x array in the program fd1d_1.2.c. (In the program fd1d_1.2.c at the end of the chapter, not the entire program but only those parts that are different from fd1d_1.1.c have been reproduced. Furthermore, key points are highlighted in boldface.) Figure 1.3 shows the results of a simulation using fd1d_1.2.c. A pulse that originates in the center propagates outward and is absorbed without reflecting anything back into the problem space.

PROBLEM SET 1.3

1. The program fd1d_1.2.c has absorbing boundary conditions at both ends. Get this program running and test it to ensure that the boundary conditions completely absorb the pulse.

1.4 PROPAGATION IN A DIELECTRIC MEDIUM

In order to simulate a medium with a dielectric constant other than 1, which corresponds to free space, we have to add the relative dielectric constant ε_r to Maxwell's equations:

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_r \varepsilon_0} \nabla \times \boldsymbol{H} \tag{1.13a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.13b}$$

We will stay with our one-dimensional example and make the change of variables in Eq. (1.5),

$$\frac{\partial E_x}{\partial t} = -\frac{1}{\varepsilon_r \varepsilon_0} \frac{\partial H_y}{\partial z} \tag{1.14a}$$

$$\frac{\partial H_y}{\partial t} = -\frac{1}{\mu_0} \frac{\partial E_x}{\partial z}, \tag{1.14b}$$

and then go to the finite-difference approximations

$$\widetilde{E}_x^{n+1/2}(k) = \widetilde{E}_x^{n-1/2}(k) - \frac{(1/2)}{\varepsilon_r} \left[H_y^n \left(k + \frac{1}{2} \right) - H_y^n \left(k - \frac{1}{2} \right) \right] \tag{1.15a}$$

$$H_y^{n+1} \left(k + \frac{1}{2} \right) = H_y^n \left(k + \frac{1}{2} \right) - \frac{1}{2} [\widetilde{E}_x^{n+1.2}(k+1) - \widetilde{E}_x^{n+1.2}(k)]. \tag{1.15b}$$

From this we can get the computer equations

```
ex[k]=ex[k]+cb[k]*(hy[k-1]-hy[k])
```
(1.16a)

```
hy[k]=hy[k]+0.5*(ex[k]-ex[k+1}).
```
(1.16b)

where

```
cb[k] = 0.5/epsilon
```
(1.17)

over those values of k that specify the dielectric material.

The program fd1d_1.3.c simulates the interaction of a pulse traveling in free space until it strikes a dielectric medium. The medium is specified by the parameter `cb` in Eq. (1.17). Figure 1.4 shows the result of a simulation with a dielectric medium having a relative dielectric constant of 4. Note that one portion of the pulse propagates into the medium and the other is reflected, in keeping with basic EM theory (6).

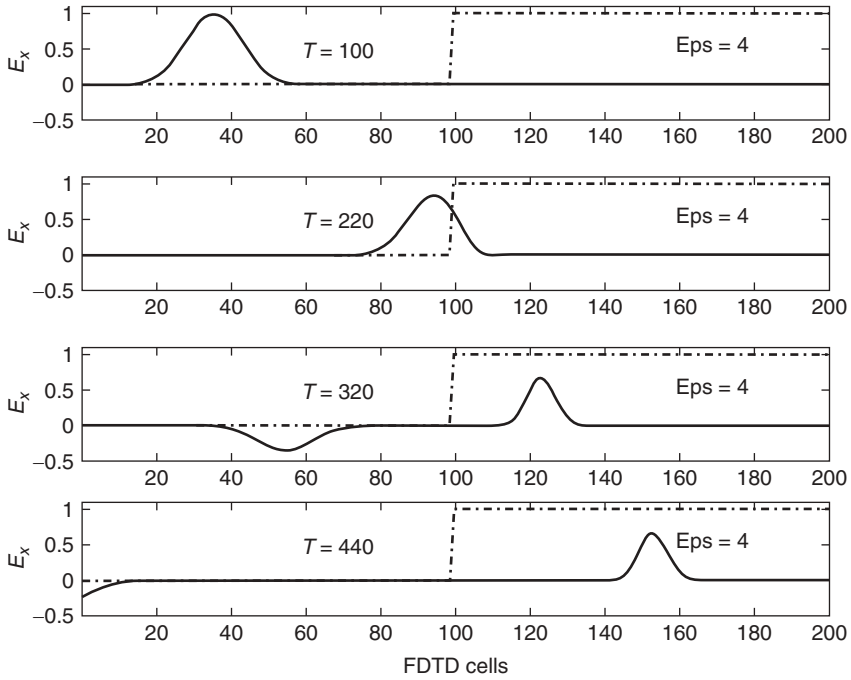

Figure 1.4 Simulation of a pulse striking a dielectric material with a dielectric constant of 4. The source originates at cell number 5.

PROBLEM SET 1.4

1. The program fd1d_1.3.c simulates a problem partly containing free space and partly dielectric material. Run this program and duplicate the results of Fig. 1.4.

2. Look at the relative amplitudes of the reflected and transmitted pulses. Are they correct? Check them by calculating the reflection and transmission coefficients (Appendix 1.A).

3. Still using a dielectric constant of 4, let the transmitted pulse propagate until it hits the far right wall. What happens? What could you do to correct this?

1.5 SIMULATING DIFFERENT SOURCES

In the first two programs, a source is assigned as values to E_x, this is referred to as a *hard source*. In fd1d_1.3.c, however, a value is added to E_x at a certain point; this is called a *soft source*. The reason is that with a hard source, a propagating pulse will see that value and be reflected, because a hard value of E_x looks like

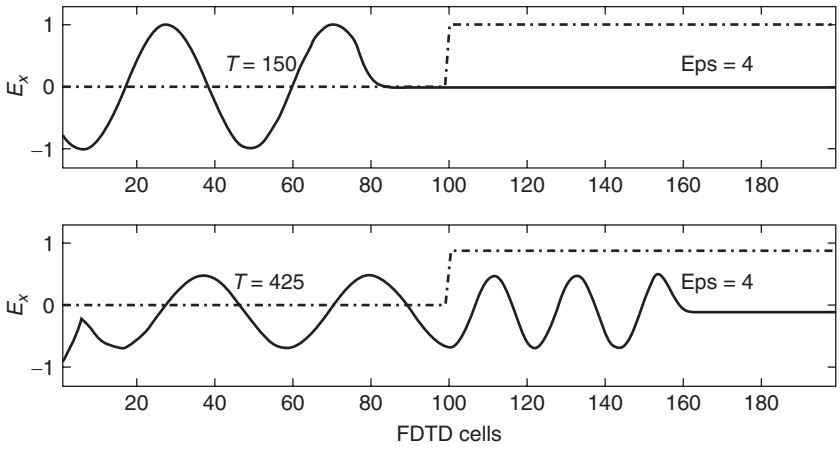

Figure 1.5 Simulation of a propagating sinusoidal wave of 700 MHz striking a medium with a relative dielectric constant of 4.

a metal wall to FDTD. With the soft source, a propagating pulse will just pass through.

Until now, we have been using a Gaussian pulse as the source. It is very easy to switch to a sinusoidal source. Just replace the parameter `pulse` with the following:

```
Pulse = sin[2*pi*freq_in*dt*T]
ex[5] = ex[5] + pulse.
```

The parameter `freq_in` determines the frequency of the wave. The source is used in the program fd1d_1.4.c. Figure 1.5 shows the same dielectric medium problem with a sinusoidal source. A frequency of 700 MHz is used. Notice that the simulation was stopped before the wave reached the far right side. Remember that we have an absorbing boundary condition, but it was only for free space.

In fd1d_1.4.c, the cell size `ddx` and the time step `dt` are specified explicitly. We do this because we need `dt` in the calculation of pulse. The cell size `ddx` is only specified because it is needed to calculate `dt` from Eq. (1.7).

PROBLEM SET 1.5

1. Modify your program fd1d_1.3.c to simulate the sinusoidal source (see fid1d_1.4.c).

2. Keep increasing your incident frequency from 700 MHz upward at intervals of 300 MHz. What happens?

3. *Wave packet*, a sinusoidal function in a Gaussian envelope, is a type of propagating wave function that is of great interest in areas such as optics. Modify your program to simulate a wave packet.

1.6 DETERMINING CELL SIZE

Choosing the cell size to be used in an FDTD formulation is similar to any approximation procedure: enough sampling points must be taken to ensure that an adequate representation is made. The number of points per wavelength is dependent on many factors (3, 4). However, a good rule of thumb is 10 points per wavelength. Experience has shown this to be adequate, with inaccuracies appearing as soon as the sampling drops below this rate.

Naturally, we must use a worst-case scenario. In general, this will involve looking at the highest frequencies we are simulating and determining the corresponding wavelength. For instance, suppose we are running simulations with 400 MHz. In free space, EM energy will propagate at the wavelength

$$\lambda_0 = \frac{c_0}{400\ \text{MHz}} = \frac{3 \times 10^8\ \text{m/s}}{4 \times 10^8\ \text{s}^{-2}} = 0.75\ \text{m}. \tag{1.18}$$

If we were only simulating free space, we would choose

$$\Delta x = \frac{\lambda_0}{10} = 7.5\ \text{cm}.$$

However, if we are simulating EM propagation in biological tissues, for instance, we must look at the wavelength in the tissue with the highest dielectric constant, because this will have the corresponding shortest wavelength. For instance, muscle has a relative dielectric constant of about 50 at 400 MHz, so

$$\lambda_0 = \frac{\left(\frac{c_0}{\sqrt{50}}\right)}{400\ \text{MHz}} = \frac{0.424 \times 10^8\ \text{m/s}}{4 \times 10^8\ \text{s}^{-2}} = 10.6\ \text{cm}.$$

So we would probably select a cell size of 1 cm.

PROBLEM SET 1.6

1. Simulate a 3-GHz sine wave impinging on a material with a dielectric constant of $\varepsilon_r = 20$.

1.7 PROPAGATION IN A LOSSY DIELECTRIC MEDIUM

So far, we have simulated EM propagation in free space or in simple media that are specified by the relative dielectric constant ε_r. However, there are many media that also have a loss term specified by the conductivity. This loss term results in the attenuation of the propagating energy.

Once more we will start with the time-dependent Maxwell's curl equations, but we will write them in a more general form, which will allow us to simulate propagation in media that have conductivity:

$$\varepsilon \frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_0} \nabla \times \boldsymbol{H} - \boldsymbol{J} \tag{1.19a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.19b}$$

$\boldsymbol{J}$ is the current density, which can also be written as

$$\boldsymbol{J} = \sigma \boldsymbol{E},$$

where σ is the conductivity. Putting this into Eq. (1.19a) and dividing by the dielectric constant we get,

$$\frac{\partial \boldsymbol{E}}{\partial t} = -\frac{1}{\varepsilon_0 \varepsilon_\mathrm{r}} \nabla \times \boldsymbol{H} - \frac{\sigma}{\varepsilon_\mathrm{r} \varepsilon_0} \boldsymbol{E}.$$

We now revert to our simple one-dimensional equation:

$$\frac{\partial E_x(t)}{\partial t} = -\frac{1}{\varepsilon_\mathrm{r} \varepsilon_0} \frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\varepsilon_\mathrm{r} \varepsilon_0} E_x(t)$$

and make the change of variable in Eq. (1.5), which gives

$$\frac{\partial \widetilde{E}_x(t)}{\partial t} = -\frac{1}{\varepsilon_\mathrm{r} \sqrt{\mu_0 \varepsilon_0}} \frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\varepsilon_\mathrm{r} \varepsilon_0} \widetilde{E}_x(t) \tag{1.20a}$$

$$\frac{\partial H_y(t)}{\partial t} = -\frac{1}{\sqrt{\mu_0 \varepsilon_0}} \frac{\partial \widetilde{E}_x(t)}{\partial z}. \tag{1.20b}$$

Next take the finite-difference approximation for both the temporal and spatial derivatives similar to Eq. (1.3a)

$$\begin{aligned} \frac{E_x^{n+1/2}(k) - E_x^{n-1/2}(k)}{\Delta t} = &-\frac{1}{\varepsilon_\mathrm{r} \sqrt{\varepsilon_0 \mu_0}} \frac{H_y^n\left(k + \frac{1}{2}\right) - H_y^n\left(k - \frac{1}{2}\right)}{\Delta x} \\ &- \frac{\sigma}{\varepsilon_\mathrm{r} \varepsilon_0} \frac{E_x^{n+1/2}(k) + E_x^{n-1/2}(k)}{2}. \end{aligned} \tag{1.21}$$

Notice that the last term in Eq. (1.20a) is approximated as the average across two time steps in Eq. (1.21). From the previous section,

$$\frac{1}{\sqrt{\mu_0 \varepsilon_0}} \frac{\Delta t}{\Delta x} = \frac{1}{2},$$

so Eq. (1.21) becomes

$$E_x^{n+1/2}(k)\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)=\left(1-\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)E_x^{n-1/2}(k) - \frac{\left(\frac{1}{2}\right)}{\varepsilon_r}\left[H_y^n\left(k+\frac{1}{2}\right)-H_y^n\left(k-\frac{1}{2}\right)\right]$$

or

$$E_x^{n+1/2}(k)=\frac{\left(1-\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}{\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}E_x^{n-1/2}(k) - \frac{\left(\frac{1}{2}\right)}{\varepsilon_r\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}\left[H_y^n\left(k+\frac{1}{2}\right)-H_y^n\left(k-\frac{1}{2}\right)\right].$$

From these we can get the computer equations

```
ex[k]=ca[k]*ex[k]+cb[k]*(hy[k-1]-hy[k])        (1.22a)
hy[k]=hy[k]+0.5*(ex[k]-ex[k+1]),               (1.22b)
```

where

```
  eaf=dt*sigma/(2*epsz*epsilon),               (1.23a)
ca[k]=(1-eaf)/(1+eaf),                         (1.23b)
cb[k]=0.5/(epsilon*(1+eaf)).                   (1.23c)
```

The program fd1d_1.5.c simulates a sinusoidal wave hitting a lossy medium that has a dielectric constant of 4 and a conductivity of 0.04. The pulse is generated at the far left side and propagates to the right (Fig. 1.6). Notice that the waveform

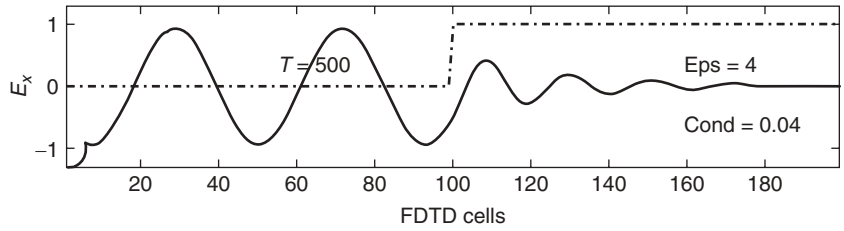

Figure 1.6 Simulation of a propagating sinusoidal wave striking a lossy dielectric material with a dielectric constant of 4 and a conductivity of 0.04 (S/m). The source is 700 MHz and originates at cell number 5.

in the medium is absorbed before it hits the boundary, so we do not have to worry about absorbing boundary conditions.

PROBLEM SET 1.7

1. Run program fd1d_1.5.c to simulate a complex dielectric material. Duplicate the results of Fig. 1.6.
2. Verify that your calculation of the sine wave in the lossy dielectric is correct, that is, it is the correct amplitude going into the slab, and then it attenuates at the proper rate (Appendix 1.A).
3. How would you write an absorbing boundary condition for a lossy material?
4. Simulate a pulse hitting a metal wall. This is very easy to do, if you remember that metal has very high conductivity. For the complex dielectric, just use $\sigma = 1.e6$, or any large number. (It does not have to be the correct conductivity of the metal, just very large.) What does this do to the FDTD parameters *ca* and *cb*? What result does this have for the field parameters E_x and H_y? If you did not want to specify dielectric parameters, how else would you simulate metal in an FDTD program?

1.A APPENDIX

When a plane wave traveling in medium 1 strikes medium 2, the fraction that is reflected is given by the reflection coefficient Γ and the fraction that is transmitted into medium 2 is given by the transmission coefficient τ. These are determined by the intrinsic impedances η_1 and η_2 of the respective media (6):

$$\Gamma = \frac{E_{\text{ref}}}{E_{\text{inc}}} = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1} \tag{1.A.1}$$

$$\tau = \frac{E_{\text{trans}}}{E_{\text{inc}}} = \frac{2\eta_2}{\eta_2 + \eta_1}. \tag{1.A.2}$$

The impedances are given by

$$\eta = \sqrt{\frac{\mu}{\varepsilon_0 \varepsilon_{\text{r}}}}. \tag{1.A.3}$$

The complex relative dielectric constant ε_{r}^* is given by

$$\varepsilon_{\text{r}}^* = \varepsilon_{\text{r}} + \frac{\sigma}{j\omega\varepsilon_0}.$$

For the case where $\mu = \mu_0$, Eq. (1.A.1) and Eq. (1.A.2) become

$$\Gamma = \frac{\frac{1}{\sqrt{\varepsilon_2^*}} - \frac{1}{\sqrt{\varepsilon_1^*}}}{\frac{1}{\sqrt{\varepsilon_2^*}} + \frac{1}{\sqrt{\varepsilon_1^*}}} = \frac{\sqrt{\varepsilon_1^*} - \sqrt{\varepsilon_2^*}}{\sqrt{\varepsilon_1^*} + \sqrt{\varepsilon_2^*}} \tag{1.A.4}$$

$$\tau = \frac{\frac{2}{\sqrt{\varepsilon_1^*}}}{\frac{1}{\sqrt{\varepsilon_2^*}} + \frac{1}{\sqrt{\varepsilon_1^*}}} = \frac{2\sqrt{\varepsilon_1^*}}{\sqrt{\varepsilon_1^*} + \sqrt{\varepsilon_2^*}}. \tag{1.A.5}$$

The amplitude of an electric field propagating in the positive z direction in a lossy dielectric medium is given by

$$E_x(z) = E_0 \mathrm{e}^{-kz} = \mathrm{e}^{-\mathrm{Re}\{k\}z} \mathrm{e}^{-j\mathrm{Im}\{k\}z},$$

where E_0 is the amplitude at $z = 0$. The wave number k is determined by

$$k = \frac{\omega}{c_0}\sqrt{\varepsilon_\mathrm{r}}. \tag{1.A.6}$$

REFERENCES

1. K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antenn. Propag.*, vol. 17, 1966, pp. 585–589.
2. A. Taflove and M. Brodwin, Numerical solution of steady state electromagnetic scattering problems using the time-dependent Maxwell's equations, *IEEE Trans. Microw. Theor. Tech.*, vol. 23, 1975, pp. 623–730.
3. A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd Edition, Boston, MA: Artech House, 1995.
4. K. S. Kunz and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*, Boca Raton, FL: CRC Press, 1993.
5. G. Mur, Absorbing boundary conditions for the finite-difference approximation of the time domain electromagnetic field equations, *IEEE Trans. Electromagn. C.*, vol. 23, 1981, pp. 377–384.
6. D. K. Cheng, *Field and Wave Electromagnetics*, Menlo Park, CA: Addison-Wesley, 1992.

C PROGRAMS USED TO GENERATE THE FIGURES IN THE CHAPTER

```
/* FD1D_1.1.C.  1D FDTD simulation in free space */

# include <math.h>
# include <stdlib.h>
# include <stdio.h>
```

```
#define KE  200

main ()
{
    float ex[KE],hy[KE];
    int n,k,kc,ke,NSTEPS;
    float ddx,dt,T;
    float t0,spread,pulse;
    FILE *fp, *fopen();

    /* Initialize */
       for ( k=1; k < KE; k++ )
       { ex[k] = 0.;
       hy[k] = 0.;
       }

    kc = KE/2;
    t0 = 40.0;
    spread = 12;
    T = 0;
    NSTEPS = 1;

while ( NSTEPS > 0 ) {
       printf( "NSTEPS --> ");
       scanf("%d", &NSTEPS);
       printf("%d \n", NSTEPS);
       n= 0;

    for ( n=1; n <=NSTEPS ; n++)
    {
       T = T + 1;

    /*  Main FDTD Loop */

       /* Calculate the Ex field */
       for ( k=1; k < KE; k++ )
       { ex[k] = ex[k] + .5*( hy[k-1] - hy[k] ) ; }

       /* Put a Gaussian pulse in the middle */

       pulse =  exp(-.5*(pow( (t0-T)/spread,2.0) ));
       ex[kc] =  pulse;
       printf( "%5.1f  %6.2f\n",t0-T,ex[kc]);

       /* Calculate the Hy field */
       for ( k=0; k < KE-1; k++ )
       { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }
```

```
       }

   /* End of the Main FDTD Loop */

       /* At the end of the calculation, print out
             the Ex and Hy fields */
       for ( k=1; k <= KE; k++ )
       { printf( "%3d   %6.2f  %6.2f\n",k,ex[k],hy[k]); }

       /* Write the E field out to a file "Ex" */
       fp = fopen( "Ex","w");
       for ( k=1; k <= KE; k++ )
       { fprintf( fp,"  %6.2f \n",ex[k]); }
       fclose(fp);

       /* Write the H field out to a file "Hy" */
       fp = fopen( "Hy","w");
       for ( k=1; k <= KE; k++ )
       { fprintf( fp,"  %6.2f \n",hy[k]); }
       fclose(fp);

       printf( "T = %5.0f\n",T);
}

}

/* FD1D_1.2.C.  1D FDTD simulation in free space */
/* Absorbing Boundary Condition added */

main ()
{
    float ex[KE],hy[KE];
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;

    for ( n=1; n <=NSTEPS ; n++)
    {
       T = T + 1;

    /*  Main FDTD Loop */

       /* Calculate the Ex field */
       for ( k=1; k < KE; k++ )
       { ex[k] = ex[k] + .5*( hy[k-1] - hy[k] ) ; }

       /* Put a Gaussian pulse in the middle */

       pulse =  exp(-.5*(pow( (t0-T)/spread,2.0) ));
       ex[kc] = ex[kc] + pulse;
       printf( "%5.1f %6.2f %6.2f\n",t0-T,arg,ex[kc]);
```

```
       /* Absorbing Boundary Conditions */

       ex[0]      = ex_low_m2;
       ex_low_m2  = ex_low_m1;
       ex_low_m1  = ex[1];

       ex[KE-1]      = ex_high_m2;
       ex_high_m2  = ex_high_m1;
       ex_high_m1  = ex[KE-2];

       /* Calculate the Hy field */
       for ( k=0; k < KE-1; k++ )
       { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }

       }

   /* End of the Main FDTD Loop */

/* FD1D_1.3.c.
/* Simulation of a pulse hitting a dielectric medium */

main ()
{
    float ex[KE],hy[KE];
    float cb[KE];
    int n,k,kc,ke,kstart,nsteps;
    float ddx,dt,T,epsz,epsilon,sigma,eaf;
    float t0,spread,pi,pulse;
    FILE *fp, *fopen();
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;

    for ( k=1; k <= KE; k++ ) {  /* Initialize to free space */
      cb[k] = .5;
    }
       printf( "Dielectric starts at --> ");
       scanf("%d", &kstart);
       printf( "Epsilon --> ");
       scanf("%f", &epsilon);
       printf("%d %6.2f   \n", kstart,epsilon);

    for ( k=kstart; k <= KE; k++ ) {
      cb[k] = .5/epsilon;
    }

       for ( k=1; k <= KE; k++ )
       { printf( "%2d   %4.2f\n",k,cb[k]); }

    /* Main part of the program */
```

```
while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;

       /* Calculate the Ex field */
       for ( k=0; k < KE; k++ )
       { ex[k] = ex[k] + cb[k]*( hy[k-1] - hy[k] ) ; }

       /* Put a Gaussian pulse at the low end */

       pulse =  exp(-.5*(pow((t0-T)/spread,2.0)));
       ex[5] =  ex[5] + pulse;
       printf( "%5.1f %6.2f %6.2f\n",T,pulse,ex[5]);

/* FD1D_1.4.c.
/* Simulation of a sinusoidal wave hitting a dielectric
 medium */

# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE  200

main ()
{
    float ddx,dt;
    float freq_in;

    ddx = .01;                  /* Cells size */
    dt =ddx/6e8;                /* Time steps */

    /* These parameters specify the input */
       printf( "Input freq (MHz)--> ");
       scanf("%f", &freq_in);
       freq_in = freq_in*1e6;
       printf(" %8.0f   \n", freq_in);

    T = 0;
    nsteps = 1;

    /* Main part of the program */
```

```
while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;

       /* Calculate the Ex field */
       for ( k=0; k < KE; k++ )
       { ex[k] = ex[k] + cb[k]*( hy[k-1] - hy[k] ) ; }

       /* Put a Gaussian pulse at the low end */

       pulse = sin(2*pi*freq_in*dt*T);
       ex[5] =  ex[5] + pulse;
       printf( "%5.1f %6.2f %6.2f\n",T,pulse,ex[5]);

/* FD1D_1.5.c. Simulation of a sinusoid hitting a lossy
 dielectric medium */
{
    float ca[KE],cb[KE];

     /* Initialize to free space */
    for ( k=0; k <= KE; k++ ) {
      ca[k] = 1.;
      cb[k] = .5;
    }
       printf( "Dielectric starts at --> ");
       scanf("%d", &kstart);
       printf( "Epsilon --> ");
       scanf("%f", &epsilon);
       printf( "Conductivity --> ");
       scanf("%f", &sigma);
       printf("%d %6.2f  %6.2f  \n", kstart,epsilon, sigma);

    eaf = dt*sigma/(2*epsz*epsilon);
       printf(" %6.4f  \n", eaf);
    for ( k=kstart; k >= KE; k++ ) {
      ca[k] = (1. - eaf)/(1 + eaf) ;
      cb[k] = .5/(epsilon*(1 + eaf) );
    }

     /* Main part of the program */

       /* Calculate the Ex field */
       for ( k=0; k < KE; k++ )
       { ex[k] = ca[k]*ex[k] + cb[k]*( hy[k-1] - hy[k] ) ; }
```

2

MORE ON ONE-DIMENSIONAL SIMULATION

Before moving on to two- and three-dimensional problems, we will stay with one-dimensional simulation to introduce some advanced concepts. First we will change the formulation slightly and introduce the use of the flux density into the simulation. This may initially seem like an unnecessary complication. However, as we get to frequency-dependent materials in Section 2.3, the advantages will become apparent. Then, in Section 2.2, we introduce the use of the discrete Fourier transform in FDTD simulation. This is an extremely powerful method to quantify the output of the simulation.

It should become apparent from this chapter how closely signal processing is linked to time domain EM simulation. This will be obvious in Section 2.4, where we use Z transforms to simulate complicated media.

2.1 REFORMULATION USING THE FLUX DENSITY

Until now, we have been using the form of Maxwell's equations given in Eq. (1.1), which uses only the E and H fields. However, a more general form is

$$\frac{\partial \boldsymbol{D}}{\partial t} = \nabla \times \boldsymbol{H}, \tag{2.1a}$$

$$\boldsymbol{D}(\omega) = \varepsilon_0 \cdot \varepsilon_r^*(\omega) \cdot \boldsymbol{E}(\omega), \tag{2.1b}$$

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0}\nabla \times \boldsymbol{E}, \tag{2.1c}$$

where $\boldsymbol{D}$ is the electric flux density. Notice that Eq. (2.1b) is written in the frequency domain. The reason for this will be explained later. We will begin by normalizing these equations, using

$$\widetilde{E} = \sqrt{\frac{\varepsilon_0}{\mu_0}} \cdot E \tag{2.2a}$$

$$\widetilde{D} = \sqrt{\frac{1}{\varepsilon_0 \mu_0}} \cdot D, \tag{2.2b}$$

which leads to

$$\frac{\partial \widetilde{\boldsymbol{D}}}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}}\nabla \times \widetilde{\boldsymbol{H}}, \tag{2.3a}$$

$$\widetilde{\boldsymbol{D}}(\omega) = \varepsilon_{\mathrm{r}}^*(\omega) \cdot \widetilde{\boldsymbol{E}}(\omega), \tag{2.3b}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}}\nabla \times \widetilde{\boldsymbol{E}}. \tag{2.3c}$$

We saw in Chapter 1 that the forms of Eq. (2.3a) and (2.3c) lead to the very simple finite-difference equations, Eq. (1.4a) and (1.4b), respectively. The only change is the use of D instead of E. However, we still have to get Eq. (2.3b) into a time domain difference equation for implementation into FDTD. The first task is to shift from the frequency domain to the time domain. We will assume we are dealing with a lossy dielectric medium of the form

$$\varepsilon_{\mathrm{r}}^*(\omega) = \varepsilon_{\mathrm{r}} + \frac{\sigma}{j\omega\varepsilon_0} \tag{2.4}$$

and substitute Eq. (2.4) into Eq. (2.3b):

$$D(\omega) = \varepsilon_{\mathrm{r}} E(\omega) + \frac{\sigma}{j\omega\varepsilon_0} E(\omega). \tag{2.5}$$

Taking the first term into the time domain is not a problem because it is a simple multiplication. In the second term, Fourier theory tells us that $1/j\omega$ in the frequency domain is integration in the time domain, so Eq. (2.5) becomes

$$D(t) = \varepsilon_{\mathrm{r}} E(t) + \frac{\sigma}{j\omega\varepsilon_0}\int_0^t E(t')\mathrm{d}t'.$$

We will want to go to the sampled time domain, so the integral will be approximated as a summation over the time step Δt:

$$D^n = \varepsilon_r E^n + \frac{\sigma \cdot \Delta t}{\varepsilon_0} \sum_{i=0}^{n} E^i. \tag{2.6}$$

Note that E and D are specified at time $t = n \cdot \Delta t$. There is still one problem: looking back at Eq. (2.3b), we see that we have to solve for E^n given the value D^n. But the value E^n is needed in the calculation of the summation. We will circumvent this by separating the E^n term from the rest of the summation:

$$D^n = \varepsilon_r E^n + \frac{\sigma \cdot \Delta t}{\varepsilon_0} E^n + \frac{\sigma \cdot \Delta t}{\varepsilon_0} \sum_{i=0}^{n-1} E^i.$$

Now we can calculate E^n from

$$E^n = \frac{D^n - \dfrac{\sigma \cdot \Delta t}{\varepsilon_0} \displaystyle\sum_{i=0}^{n-1} E^i}{\varepsilon_r + \dfrac{\sigma \cdot \Delta t}{\varepsilon_0}}. \tag{2.7}$$

We can calculate E^n, the *current* value of E, from the current value of D and *previous* values of E. It will prove advantageous to define a new parameter for the summation

$$I^n = \frac{\sigma \cdot \Delta t}{\varepsilon_0} \sum_{i=0}^{n-1} E^i,$$

so Eq. (2.7) can be reformulated with the following equations:

$$E^n = \frac{D^n - I^{n-1}}{\varepsilon_r + \dfrac{\sigma \cdot \Delta t}{\varepsilon_0}} \tag{2.8a}$$

$$I^n = I^{n-1} + \frac{\sigma \cdot \Delta t}{\varepsilon_0} \sum_{i=0}^{n-1} E^i. \tag{2.8b}$$

Note that the summation is calculated by Eq. (2.8b), which at every time step n, simply adds the value E^n times the constant term to the previous values of the summation at $n - 1$. It is not necessary to store all the values of E^n from 0 to n. Now the entire FDTD formulation is

```
dx[k]=dx[k]+0.5*([hyk-1]-hy[k]),        (2.9a)
dx[k]=gax[k]*(dx[k]-ix[k]),             (2.9b)
```

```
ix[k]=ix[k]+gbx[k]*ex[k],                    (2.9c)
hy[k]=hy[k]+0.5*(ex[k]-ex[k+1]),             (2.9d)
```

where

```
gax[k]=1/(epsilon+(sigma*dt/epsz))           (2.10a)
gbx[k]=sigma*dt/epsz.                        (2.10b)
```

The important point is that all of the information regarding the media is contained in Eq. (2.9b) and (2.9c). For free space, `gax = 1` and `gbx = 0` and for lossy material, `gax` and `gbx` are calculated according to Eq. (2.10a) and (2.10b). In calculating `ex[k]` at the point k, it uses only values of `dx[k]` and the previous values of `ex[k]` in the time domain. Equation (2.9a) and (2.9d), which contain the spatial derivatives, do not change regardless of the media.

It may seem as though we have paid a high price for this fancy formulation compared to the formulation of Chapter 1. We now need D_x as well as E_x and an auxiliary parameter *Ix*. The real advantage comes when we deal with more complicated materials, as we will see in the following sections.

PROBLEM SET 2.1

1. The program fd1d_2.1.c implements the reformulation using the flux density. Get this program running and repeat the results of problem 1.7.1.

2.2 CALCULATING THE FREQUENCY DOMAIN OUTPUT

Until now, the output of our FDTD programs has been the E field itself, and we have been content to simply watch a pulse or sine wave propagate through various media. Needless to say, before any such practical applications can be implemented, it will be necessary to quantify the results. Suppose now that we are asked to calculate the E field distribution at every point in a dielectric medium subject to illumination at various frequencies. One approach would be to use a sinusoidal source and iterate the FDTD program until a steady state is reached and determine the resulting amplitude and phase at every point of interest in the medium. This would work, but then we must repeat the process for every frequency of interest. According to system theory, we can get the response to every frequency if we use an impulse as the source. We could go back to using the Gaussian pulse, which, if it is narrow enough, is a good approximation to an impulse. We then iterate the FDTD program until the pulse has died out, and take the Fourier transform of the E fields in the medium. If we have the Fourier transform of the E field at a point, then we know the amplitude and phase of the E field that would result from illumination by any sinusoidal source. This, too,

has a serious drawback: the E field for all the time domain data at every point of interest would have to be stored until the FDTD program is through iterating so that the Fourier transform of the data could be taken, presumably using a fast Fourier transform algorithm. This presents a logistical difficulty.

Here is an alternative. Suppose we want to calculate the Fourier transform of the E field $E(t)$ at a frequency f_1. This can be done by the equation

$$E(f_1) = \int_0^{t_T} E(t) \cdot \mathrm{e}^{-j2\pi f_1 t} \mathrm{d}t. \tag{2.11}$$

Notice that the lower limit of the integral is at 0 because the FDTD program assumes all causal functions. The upper limit is t_T, the time at which the FDTD iteration is halted. Rewrite Eq. (2.11) in a finite-difference form,

$$E(f_1) = \sum_{n=0}^{T} E(n \cdot \Delta t) \cdot \mathrm{e}^{j2\pi f_1 (n \cdot \Delta t)}, \tag{2.12}$$

where T is the number of iterations and Δt is the time step, so $t_T = T \cdot \Delta t$. Equation (2.12) can be divided into its real and imaginary parts

$$\begin{aligned} E(f_1) = &\sum_{n=0}^{T} E(n \cdot \Delta t) \cdot \cos(2\pi f_1 \cdot \Delta t \cdot n) \\ &- j \sum_{n=0}^{T} E(n \cdot \Delta t) \cdot \sin(2\pi f_1 \cdot \Delta t \cdot n), \end{aligned} \tag{2.13}$$

which can be implemented in computer code by

```
real_pt[m,k]=real_pt[m,k]+ex[k]*cos(2*pi*freq(m)*dt*n)   (2.14a)
```

```
imag_pt[m,k]=imag_pt[m,k]+ex[k]*sin(2*pi*freq(m)*dt*n).   (2.14b)
```

For every point k in the region of interest, we require only two computer words of memory for every frequency of interest f_m. At any point k, from the real part, `real_pt[m,k]`, and the imaginary part, `imag_pt[m,k]`, of $E(f_m)$, we can determine the amplitude and phase at the frequency f_m:

```
amp[m,k]=sqrt(pow(real_pt[m,k],2.)+pow(imag_pt[m,k],2.))   (2.15a)
```

```
phase[m,k]= atanw(imag_pt[m,k],real_pt[m,k]).   (2.15b)
```

Note that there is an amplitude and phase associated with every frequency at each cell (1, 2). The program fd1d_2.2c at the end of the chapter calculates the

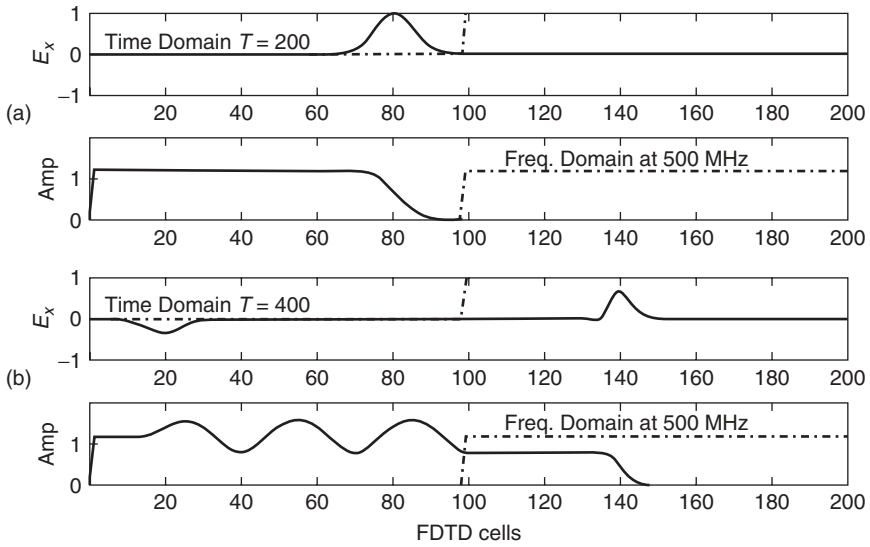

Figure 2.1 Simulation of a pulse striking a dielectric medium with $\varepsilon_r = 4$. (a) The pulse after 200 time steps. Notice that the Fourier amplitude is 1 in that part of the space where the pulse has traveled, but 0 elsewhere. (b) After 400 time steps, the pulse has struck the medium, and part of it has been transmitted and the other part reflected. The Fourier amplitude in the medium is 0.667, which is the percentage that has been transmitted.

frequency response at three frequencies throughout the problem space. Figure 2.1 is a simulation of a pulse hitting a dielectric medium with a dielectric constant of 4, similar to Fig. 1.4. The frequency response at 500 MHz is also shown. At $T = 200$, before the pulse has hit the medium, the frequency response is 1 through the part of the space where the pulse has traveled. After 400 time steps, the pulse has hit the medium; some of the pulse has penetrated into the medium and some of it has been reflected. The amplitude of the transmitted pulse is determined by Eq. (1.A.2)

$$\tau = \frac{\sqrt{4} \cdot 1}{1 + \sqrt{4}} = 0.667,$$

which is the Fourier amplitude in the medium. The Fourier amplitude outside the medium varies between $1 - 0.333$ and $1 + 0.333$. This is in keeping with the pattern formed by the standing wave that is created from a sinusoidal signal, the reflected wave of which interacts with the original incident wave.

PROBLEM SET 2.2

1. The program fd1d_2.2.c implements the discrete Fourier transform with a Gaussian pulse as its source. Get this program running. Duplicate the results in Fig. 2.1.

2.3 FREQUENCY-DEPENDENT MEDIA

The dielectric constant and conductivity of most media vary at different frequencies. The pulses we have been using as a source in Chapters 1 and 2 contain a spectrum of frequencies. In order to simulate frequency-dependent material, we will need a way to account for this. One of the most significant developments in the FDTD method is a means to simulate frequency-dependent materials (3).

We will start with a very simple example to illustrate the ideas. Suppose we have a medium whose dielectric constant and conductivity vary over the frequency range 10–1000 MHz (Fig. 2.2). A material like this can be adequately represented by the following formulation:

$$\varepsilon_{\mathrm{r}}^{*}(\omega) = \varepsilon_{\mathrm{r}} + \frac{\sigma}{j\omega\varepsilon_0} + \frac{\chi_1}{1 + j\omega t_0}. \tag{2.16}$$

This is referred to as the Debye formulation. In this formulation, there is a dielectric constant ε_{r} and a conductivity σ, but there is also a frequency-dependent term. The following parameters represent the medium of Fig. 2.2:

$$\varepsilon_{\mathrm{r}} = 2, \quad \sigma = 0.01, \quad \chi_1 = 2, \quad t_0 = 0.001\ \mu\mathrm{s}.$$

In order to simulate this medium in FDTD, Eq. (2.16) must be put into the sampled time domain. Let us define the last term times the E field as

$$S(\omega) = \frac{\chi_1}{1 + j\omega t_0} E(\omega). \tag{2.17}$$

The inverse Fourier transform of the Debye term is $(\chi_1/t_0)\mathrm{e}^{-(t/t_0)}u(t)$, where $u(t)$ is the rectangular function, which is 0 for $t < 0$ and 1 thereafter. (Recall that implicit to FDTD is the fact that all functions are causal, i.e., 0 for $t < 0$,

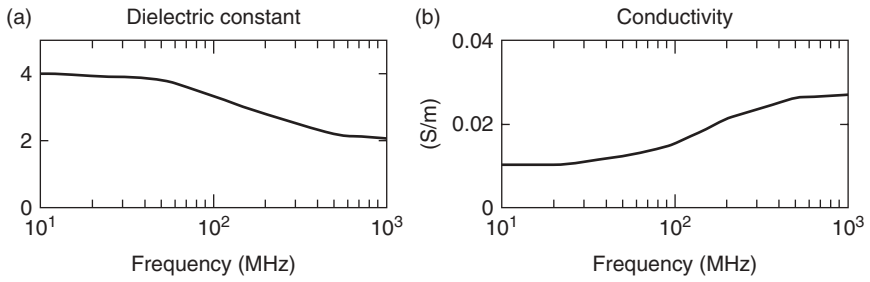

Figure 2.2 (a) Relative dielectric constant and (b) conductivity as functions of frequency for a Debye medium with the following properties: $\varepsilon_{\mathrm{r}} = 2, \sigma = 0.1, \chi = 1,$ and $\tau = 0.001\ \mu\mathrm{s}$.

because the computer programs initialize the field values to 0.) Equation (2.17) in the frequency domain becomes the convolution

$$S(t) = \frac{\chi_1}{t_0}\int_0^t \mathrm{e}^{-(t'-t)/t_0}\cdot E(t')\mathrm{d}t'$$

in the time domain. We now have to approximate this as a summation in the sampled time domain:

$$\begin{aligned} S^n &= \chi_1 \cdot \frac{\Delta t}{t_0}\sum_{i=0}^{n} \mathrm{e}^{-\Delta t(n-i)/t_0}\cdot E^i \\ &= \chi_1 \cdot \frac{\Delta t}{t_0}\left[E^n + \sum_{i=0}^{n-1} \mathrm{e}^{-\Delta t(n-i)/t_0}\cdot E^i\right]. \end{aligned} \tag{2.18}$$

Notice that

$$\begin{aligned} S^{n-1} &= \chi_1 \cdot \frac{\Delta t}{t_0}\sum_{i=0}^{n-1} \mathrm{e}^{-\Delta t(n-1-i)/t_0}\cdot E^i \\ &= \chi_1 \cdot \mathrm{e}^{-\Delta/t_0}\frac{\Delta t}{t_0}\sum_{i=0}^{n-1} \mathrm{e}^{-\Delta t(n-i)/t_0}\cdot E^i. \end{aligned}$$

Substituting this value into Eq. (2.18) gives

$$S^n = \mathrm{e}^{-\Delta/t_0}S^{n-1} + \chi_1 \cdot \frac{\Delta t}{t_0}\cdot E^n. \tag{2.19}$$

Similar to the way we handled the lossy dielectric, we can write

$$\begin{aligned} D^n &= \varepsilon_\mathrm{r}E^n + I^n + S^n \\ &= \varepsilon_\mathrm{r}E^n + \left[\frac{\sigma\cdot\Delta t}{\varepsilon_0}E^n + I^n\right] + \left[\chi_1\cdot\frac{\Delta t}{t_0}E^n + \mathrm{e}^{-\Delta t/t_0}\cdot S^{n-1}\right], \end{aligned} \tag{2.20}$$

and solving for E^n

$$E^n = \frac{D^n - I^{n-1} - \mathrm{e}^{-\Delta t/t_0}S^{n-1}}{\varepsilon_\mathrm{r} + \frac{\sigma\cdot\Delta t}{\varepsilon_0} + \chi_1\cdot\frac{\Delta t}{t_0}}, \tag{2.21a}$$

$$I^n = I^{n-1} + \frac{\sigma\cdot\Delta t}{\varepsilon_0}E^n, \tag{2.21b}$$

$$S^n = \mathrm{e}^{-\Delta t/t_0}S^{n-1} + \chi_1\cdot\frac{\Delta t}{t_0}E^n. \tag{2.21c}$$

This formulation is implemented by the following computer codes:

```
dx[k]=dx[k]+.5*(hy[k-1]-hy[k]),                    (2.22a)
ex[k]=gax[k]+ (dx[k]-ix[k]-del_exp*sx[k]),         (2.22b)
ix[k]=ix[k]+gbx[k]*ex[k],                          (2.22c)
sx[k]=del_exp*sx[k]+gcx[k]*ex[k],                  (2.22d)
hy[k]=hy[k]+.5*(ex[k]-ex[k+1]).                    (2.22e)
```

where

```
gax[k]=1/(epsr+(sigma*dt/epsz)+(chi1*dt/t0)),      (2.23a)
gbx[k]=sigma*dt/epsz,                              (2.23b)
gbc[k]=chi1*dt/t0,                                 (2.23c)
```

and

```
del_exp=exp(-dt/t0).
```

Once again, note that everything concerning the medium is contained in Eq. (2.22b), (2.22c), and (2.22d); however, Eq. (2.22a) and (2.22e), the calculation of the flux density and the magnetic field, respectively, are unchanged.

The program fd1d_2.3.c calculates the frequency domain amplitude and phase for three frequencies. Figure 2.3 shows the simulation of a pulse going into a frequency-dependent dielectric material with the properties

$$\varepsilon_r = 2, \quad \sigma = 0.01, \quad \chi_1 = 2, \quad t_0 = \mu\text{s}.$$

This set of parameters has the following effective dielectric constants and conductivities at the three frequencies:

Frequency, MHz	ε_r	σ, S/m
50	6.55	0.024
200	3.94	0.047
500	2.46	0.06

Notice that the Fourier amplitude attenuates more rapidly at 200 MHz than at 50 MHz, and more rapidly still at 500 MHz. This is because the conductivity is higher at these frequencies. At the same time, the higher relative dielectric constant at the lower frequencies means that the amplitude just inside the medium is smaller.

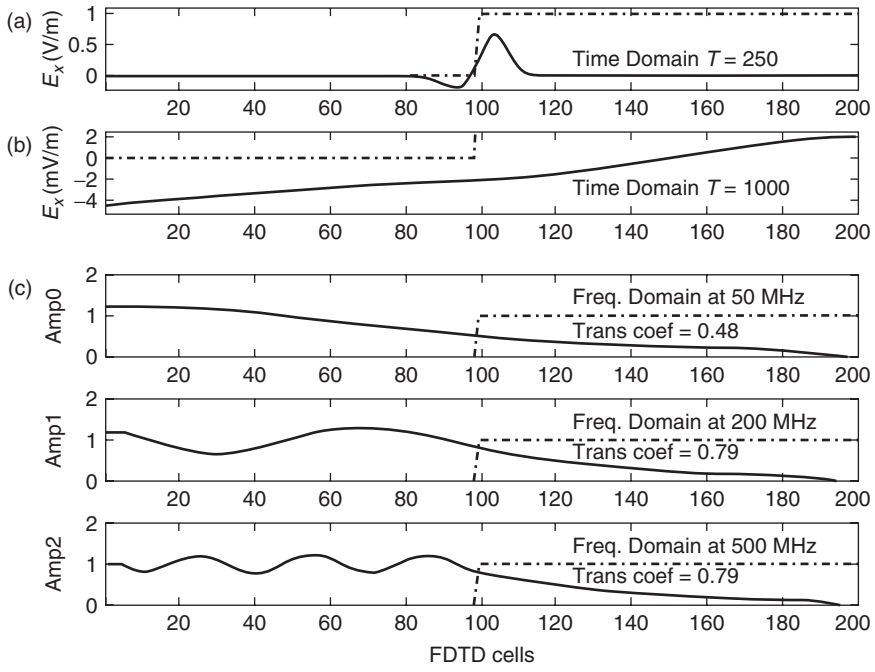

Figure 2.3 Simulation of a pulse striking a frequency-dependent dielectric medium (Debye medium) with the following properties: $\varepsilon_r = 2, \sigma = 0.1, \chi = 2,$ and $\tau = 0.001\,\mu$s. (a) After 250 time steps, the pulse has struck the medium, and part of it has been transmitted and part reflected. (b) After 1000 steps (second from top) the pulse has penetrated into the medium, but has spread. (c) Notice the different percentages of transmittance and the different rates of attenuation within the medium at each frequency due to the different effective dielectric constants and conductivities.

2.3.1 Auxiliary Differential Equation Method

We could have taken a somewhat different approach to the simulation of the dispersive medium described by Eq. (2.16), namely, the auxiliary differential equation (ADE) method (4, 5). Let us look at Eq. (2.17), but rewrite it as

$$(1 + j\omega t_0)S(\omega) = \chi_1 E(\omega). \tag{2.24}$$

Once again, we must find a way to take this to the discrete time domain for implementation in the FDTD formulation. We will start by going to the continuous time domain, and Eq. (2.24) becomes

$$s(t) + t_0 \frac{\mathrm{d}s(t)}{\mathrm{d}t} = \chi_1 \mathrm{e}(t). \tag{2.25}$$

In the sampled time domain, this becomes

$$\frac{S^n + S^{n-1}}{2} + t_0 \frac{S^n - S^{n-1}}{\Delta t} = \chi_1 E^n.$$

Notice that we approximate the $s(t)$ term over two time steps. We did this because we needed two time steps to approximate the derivatives. As before, we solve for S^n:

$$S^n = \frac{\left(1 - \dfrac{\Delta t}{2 \cdot t_0}\right)}{\left(1 + \dfrac{\Delta t}{2 \cdot t_0}\right)} S^{n-1} + \frac{\dfrac{\Delta t}{t_0} \cdot \chi_1}{\left(1 + \dfrac{\Delta t}{2 \cdot t_0}\right)} E^n. \tag{2.26}$$

We can use this instead of Eq. (2.19) to calculate E^n in Eq. (2.21). How can we be sure these will give equivalent answers? You are probably familiar with the following approximations:

$$1 - \delta \cong \mathrm{e}^{-\delta} \quad \text{if } \delta << 1$$

$$\frac{1}{1 + \delta} \cong \mathrm{e}^{-\delta} \quad \text{if } \delta << 1.$$

Putting the two together gives

$$\frac{1 - \delta}{1 + \delta} \cong \mathrm{e}^{-2\delta} \quad \text{if } \delta << 1.$$

In this case,

$$\delta = \frac{\Delta t}{2 \cdot t_0},$$

so we have

$$\frac{\left(1 - \dfrac{\Delta t}{2 \cdot t_0}\right)}{\left(1 + \dfrac{\Delta t}{2 \cdot t_0}\right)} \cong \mathrm{e}^{-\Delta t / t_0}. \tag{2.27}$$

This leaves only the following question: how do we know that $\Delta t / t_0$ is small enough? Recall that the cell size has to be small enough to get about 10 points per wavelength for the smallest wavelength in the simulation. This is a similar situation. If the medium that we are trying to simulate has a Debye term with a time constant of t_0, we must be sure that our time steps are small compared to t_0, say $\Delta t \leq t_0/10$. This insures that Eq. (2.27) is a fairly good approximation.

PROBLEM SET 2.3

1. The program fd1d_2.3.c implements the frequency-dependent formulation. Get this program running and repeat the results of Fig. 2.3.

2.4 FORMULATION USING Z TRANSFORMS

If you have been studying the Z transform theory in Appendix A, or if you are already familiar with Z transforms, you will now know the advantage of using Z transforms for the FDTD formulation of frequency-dependent media (6). Returning to the problem of calculating the E in a Debye medium, we begin with our frequency domain equations

$$D(\omega) = \left(\varepsilon_{\mathrm{r}} + \frac{\sigma}{j\omega\varepsilon_0} + \frac{\chi_1}{1 + j\omega t_0}\right) E(\omega).$$

We can avoid dealing with troublesome convolution integrals in the time domain because we will go immediately to the Z domain

$$D(z) = \varepsilon_{\mathrm{r}} E(z) + \frac{\sigma \cdot \dfrac{\Delta t}{\varepsilon_0}}{1 - z^{-1}} E(z) + \frac{\chi_1 \cdot \dfrac{\Delta t}{t_0}}{1 - \mathrm{e}^{-\Delta t/t_0} z^{-1}} E(z). \tag{2.28}$$

Notice that the factor Δt, which is the time step, had to be added to the last two terms in going from the time domain to the Z domain. Similar to what we did earlier, we will define some auxiliary parameters:

$$I(z) = \frac{\sigma \cdot \dfrac{\Delta t}{\varepsilon_0}}{1 - z^{-1}} E(z) = z^{-1} I(z) + \sigma \cdot \frac{\Delta t}{\varepsilon_0} E(z) \tag{2.29a}$$

$$S(z) = \frac{\chi_1 \cdot \dfrac{\Delta t}{t_0}}{1 - \mathrm{e}^{-\Delta t/t_0} z^{-1}} E(z) = \mathrm{e}^{-\Delta t/t_0} S(z) + \chi_1 \cdot \frac{\Delta t}{t_0} E(z). \tag{2.29b}$$

Equation (2.28) then becomes

$$\begin{aligned} D(z) = \varepsilon_{\mathrm{r}} E(z) + z^{-1} I(z) + \sigma \cdot \frac{\Delta t}{\varepsilon_0} E(z) \\ + \mathrm{e}^{-\Delta t/t_0} z^{-1} S(z) + \chi_1 \cdot \frac{\Delta t}{t_0} E(z), \end{aligned} \tag{2.30}$$

from which we can solve for $E(z)$ by

$$E(z) = \frac{D(z) - z^{-1} I(z) - \mathrm{e}^{-\Delta t/t_0} z^{-1} S(z)}{\varepsilon_{\mathrm{r}} + \sigma \cdot \dfrac{\Delta t}{\varepsilon_0} + \chi_1 \cdot \dfrac{\Delta t}{t_0}}. \tag{2.31}$$

The advantage of the Z transform formulation is that to get to the sampled time domain, replace $E(z)$ with E^n and $z^{-1}E(z)$ with E^{n-1}, and similarly replace the other parameters in Eq. (2.29a) and (2.29b) and Eq. (2.31). What you get is

$$E^n = \frac{D^{n-1} - I^{n-1} - \mathrm{e}^{-\Delta t/t_0} S^{n-1}}{\varepsilon_\mathrm{r} + \sigma \cdot \dfrac{\Delta t}{\varepsilon_0} + \chi_1 \cdot \dfrac{\Delta t}{t_0}}, \tag{2.32a}$$

$$I^n = I^{n-1} + E^n, \tag{2.32b}$$

$$S^n = \mathrm{e}^{-\Delta t/t_0} S^n + \chi_1 \cdot \frac{\Delta t}{t_0} E^n, \tag{2.32c}$$

which is *exactly* what we got in the previous section. The difference is that we avoided doing anything with integrals and their approximations. As we move to formulations that are more complicated, the advantage of the Z transform will become evident.

2.4.1 Simulation of Unmagnetized Plasma

In this section, we will demonstrate the versatility of the methods we have learned in this chapter by simulating a medium completely different from the media we have been working with so far. The permittivity of unmagnetized plasmas is given as (7)

$$\varepsilon^*(\omega) = 1 + \frac{\omega_\mathrm{p}^2}{\omega(j\nu_\mathrm{c} - \omega)}, \tag{2.33}$$

where

$\omega_\mathrm{p} = 2\pi f_\mathrm{p}$;
f_p = the plasma frequency;
ν_c = the electron collision frequency.

Using the partial fraction expansion, Eq. (2.33) can be written as

$$\varepsilon^*(\omega) = 1 + \frac{\dfrac{\omega_\mathrm{p}^2}{\nu_\mathrm{c}}}{j\omega} + \frac{\dfrac{\omega_\mathrm{p}^2}{\nu_\mathrm{c}}}{\nu_\mathrm{c} + j\omega}. \tag{2.34}$$

This is the value we will use for the complex dielectric constant in Eq. (2.3b). Notice that the form resembles Eq. (2.16), the expression for a lossy material with a Debye term. There are several ways of approaching this problem, but let us begin by taking the Z transforms of Eq. (2.34) to obtain

$$\varepsilon^*(z) = \frac{1}{\Delta t} + \frac{\dfrac{\omega_\mathrm{p}^2}{\nu_\mathrm{c}}}{1 - z^{-1}} - \frac{\dfrac{\omega_\mathrm{p}^2}{\nu_\mathrm{c}}}{1 - \mathrm{e}^{-\nu_\mathrm{c}\cdot\Delta t} z^{-1}}. \tag{2.35}$$

By the convolution theorem, the Z transform of Eq. (2.3b) is

$$D(z) = \varepsilon^*(z) \cdot E(z) \cdot \Delta t. \tag{2.36}$$

By inserting Eq. (2.35) into Eq. (2.36), we obtain

$$\begin{aligned} D(z) &= E(z) + \frac{\omega_p^2 \Delta t}{\nu_c}\left[\frac{1}{1-z^{-1}} - \frac{1}{1-e^{-\nu_c\cdot\Delta t}z^{-1}}\right]E(z) \\ &= E(z) + \frac{\omega_p^2 \Delta t}{\nu_c}\left[\frac{\left(1-e^{-\nu_c\cdot\Delta t}\right)z^{-1}}{1-(1-e^{-\nu_c\cdot\Delta t})z^{-1}+e^{-\nu_c\cdot\Delta t}z^{-2}}\right]E(z). \end{aligned}$$

Notice that we cross multiplied the term in the brackets. An auxiliary term can be defined as

$$S(z) = \frac{\omega_p^2 \Delta t}{\nu_c}\left[\frac{\left(1-e^{-\nu_c\cdot\Delta t}\right)}{1-(1-e^{-\nu_c\cdot\Delta t})z^{-1}+e^{-\nu_c\cdot\Delta t}z^{-2}}\right]E(z).$$

$E(z)$ can be solved for by

$$E(z) = D(z) - z^{-1}S(z) \tag{2.37a}$$

$$S(z) = (1-e^{-\nu_c\cdot\Delta t})z^{-1}S(z) - e^{-\nu_c\cdot\Delta t}z^{-2}S(z) + \frac{\omega_p^2\Delta t}{\nu_c}(1-e^{-\nu_c\cdot\Delta t})E(z). \tag{2.37b}$$

Therefore, the FDTD simulation becomes

```
ex[k]=dx[k]- sx[k];
```

```
sx[k]=(1 + exp(-vc*dt)*smx1[k]
```

```
      exp(-vc*dt)smx2[k])                                    (2.38a)
```

```
        +(pow(omega,2.)*dt/vc)*(1-exp(-vc*dt))*ez[k];
```

(2.38b)

```
sxm2[k]=sxm1[k];
```

(2.38c)

```
sxm1[k]=sx[k];
```

(2.38d)

Notice from Eq. (2.38b) that we need the two previous values of S in our calculation. This is accomplished by Eq. (2.38c) and (2.38d).

We will do a simulation of a pulse propagating in free space that comes upon plasma, which is a very interesting medium. At relatively low frequencies (below the plasma resonance frequency f_p), it looks like a metal, and at higher frequencies (above the plasma resonance frequency f_p), it becomes transparent. The following simulation uses the properties of silver: $\nu_c = 57$ THz, $f_p = 2000$ THz.

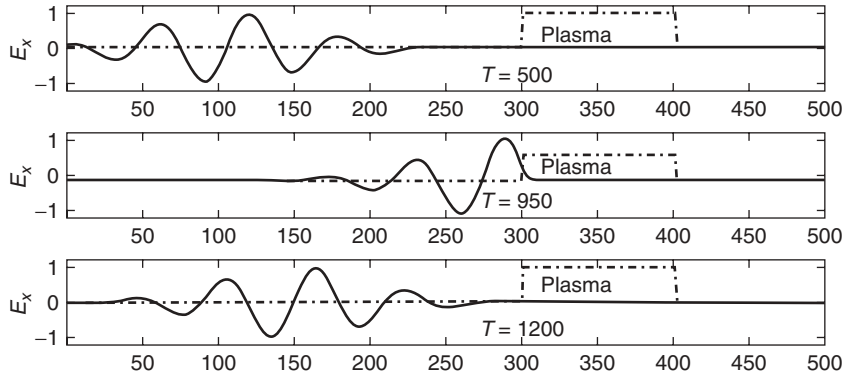

Figure 2.4 Simulation of a wave propagating in free space and striking a plasma medium. The plasma has the properties of silver: $f_p = 2000$ THz and $\nu_c = 57$ THz. The propagating wave has a center frequency of 500 THz. After 1200 time steps, it has been completely reflected by the plasma.

Of course, since we are simulating much higher frequencies, we will need a much smaller cell size. In the course of this problem, it will be necessary to simulate EM waves of 4000 THz (tera = 10^{12}). At this frequency, the free space wavelength is

$$\lambda = \frac{3 \times 10^8}{4 \times 10^{15}} = 0.75 \times 10^{-7}.$$

In following our rule of thumb of at least 10 points per wavelength, a cell size of 1 nm, that is, $\Delta x = 10^{-9}$, will be used. Figure 2.4 shows our first simulation at 500 THz, well below the plasma frequency. The incident pulse is a sine wave inside a Gaussian envelope. Notice that the pulse interacts with the plasma almost as if the plasma were a metal barrier—the pulse is almost completely reflected. Figure 2.5 is a similar simulation at 4000 THz, well above the plasma frequency. A small portion of it is reflected, but the majority of the pulse passes right through.

PROBLEM SET 2.4

1. Modify the program fd1d_2.3.c to simulate plasma and duplicate the results of Fig. 2.4 and Fig. 2.5. This is much easier than it might look. First change your cell size to 1 nm. Then replace the calculation of the E field for a lossy Debye medium with that of Eq. (2.38). After doing the simulations at 500 and 4000 THz, repeat at 2000 THz. What happens?

2. Repeat problem 2.4.1, but use a narrow Gaussian pulse as your input and calculate the frequency response at 500, 2000, and 4000 THz. Does the result look the way you would expect. Particularly at 2000 THz?

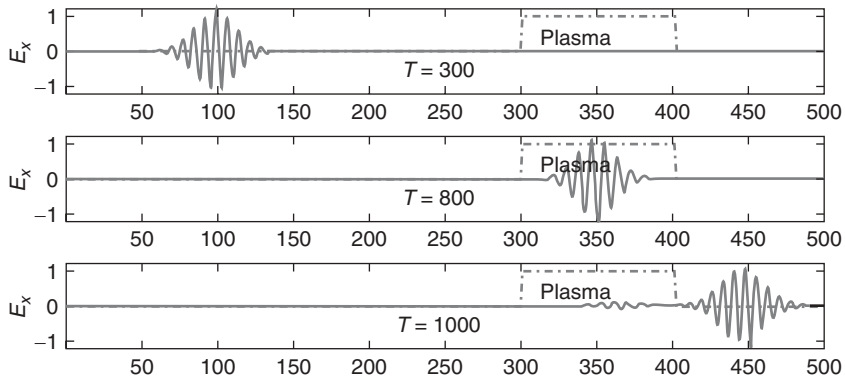

Figure 2.5 Simulation of a wave propagating in free space and striking a plasma medium. The plasma has the properties of silver: $f_p = 2000$ THz and $\nu_c = 57$ THz. The propagating wave has a center frequency of 4000 THz. After 1000 time steps, it has completely passed through the plasma.

2.5 FORMULATING A LORENTZ MEDIUM

The Debye model describes a single-pole frequency dependence. We now move to the next level, which is a two-pole dependence referred to as the *Lorentz formulation*:

$$\varepsilon_r^*(\omega) = \varepsilon_r + \frac{\varepsilon_1}{1 + j2\delta_0\left(\dfrac{\omega}{\omega_0}\right) - \left(\dfrac{\omega}{\omega_0}\right)^2}. \tag{2.39}$$

Figure 2.6 is a graph of the dielectric constant and conductivity of a material with the following Lorentz parameters: $\varepsilon_r = 2$, $\varepsilon_1 = 2$, $f_0 = 100$ MHz, ($\omega_0 = 2\pi f_0$), and $\delta_0 = 0.25$. To simulate a Lorentz medium in the FDTD formulation, we first put Eq. (2.39) into Eq. (2.3b),

$$\begin{aligned} D(\omega) &= \varepsilon_r E(\omega) + \frac{\varepsilon_1}{1 + j2\delta_0\left(\dfrac{\omega}{\omega_0}\right) - \left(\dfrac{\omega}{\omega_0}\right)^2} E(\omega) \\ &= \varepsilon_r E(\omega) + S(\omega), \end{aligned} \tag{2.40}$$

where we have defined again an auxiliary term,

$$S(\omega) = \frac{\omega_0^2 \varepsilon_1}{\omega_0^2 + j2\delta_0 + (j\omega)^2} E(\omega).$$

We will use the ADE method to move this to the time domain. Start by rewriting it in the following manner:

$$[\omega_0^2 + j2\delta_0 + (j\omega)^2] S(\omega) = \omega_0^2 \varepsilon_1 E(\omega).$$

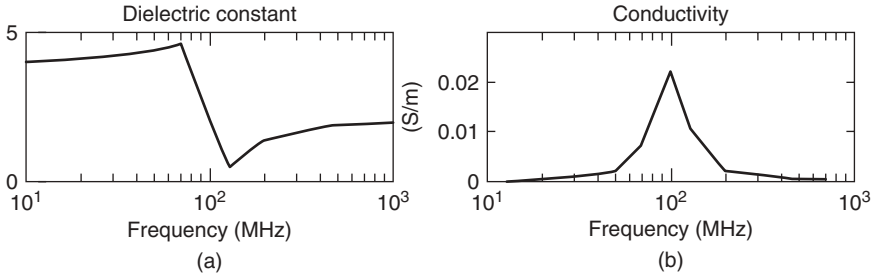

Figure 2.6 (a) Relative dielectric constant and (b) conductivity as functions of frequency for a Lorentz medium with the following properties: $\varepsilon_r = 2, \varepsilon_1 = 2, f_0 = 100$ MHz, and $\delta = 0.25$.

Finally, we proceed to the finite-difference approximations

$$\omega_0^2 S^{n-1} + 2\delta_0\omega_0 \frac{S^n - S^{n-2}}{2\Delta t} + \frac{S^n - 2S^{n-1} + S^{n-2}}{(\Delta t)^2} = \omega_0^2 \varepsilon_1 E^{n-1}.$$

A few things are worth noting: the second-order derivative generated a second-order differencing

$$\frac{\mathrm{d}^2 s(t)}{\mathrm{d}t^2} \cong \frac{S^n - 2S^{n-1} + S^{n-2}}{(\Delta t)^2}.$$

The first-order derivative is taken over two time steps instead of one:

$$\frac{\mathrm{d}s(t)}{\mathrm{d}t} \cong \frac{S^n - S^{n-2}}{2\Delta t}.$$

This was done because the second-order derivative spanned two time steps. Next, we solve for the newest value of S^n:

$$S^n \left(\frac{\delta_0\omega_0}{\Delta t} + \frac{1}{\Delta t^2}\right) + S^{n-1}\left(\omega_0^2 - \frac{2}{\Delta t}\right) + S^{n-2}\left(-\frac{\delta_0\omega_0}{\Delta t} + \frac{1}{\Delta t^2}\right) = \omega_0^2\varepsilon_1 E^{n-1},$$

$$S^n = \frac{\left(\omega_0^2 - \dfrac{2}{\Delta t}\right)}{\left(\dfrac{\delta_0\omega_0}{\Delta t} + \dfrac{1}{\Delta t^2}\right)} S^{n-1} - \frac{(1 - \Delta t\delta_0\omega_0)}{(1 + \Delta t\delta_0\omega_0)} S^{n-2} + \frac{\Delta t^2\omega_0^2\varepsilon_1}{\left(\dfrac{\delta_0\omega_0}{\Delta t} + \dfrac{1}{\Delta t^2}\right)} E^{n-1},$$

$$S^n = \frac{(2 - \Delta t^2\omega_0^2)}{(1 + \Delta t\delta_0\omega_0)} S^{n-1} - \frac{(1 - \Delta t\delta_0\omega_0)}{(1 + \Delta t\delta_0\omega_0)} S^{n-2} + \frac{\Delta t^2\omega_0^2\varepsilon_1}{(1 + \Delta t\delta_0\omega_0)} E^{n-1}. \quad (2.41)$$

Now we will return to Eq. (2.40) and take it to the sampled time domain

$$D^n = \varepsilon_r E^n + S^n,$$

which we rewrite as

$$E^n = \frac{D^n - S^n}{\varepsilon_r}.$$

Notice that we already have a solution for S^n in Eq. (2.41). Also, because we need only the previous value of E, that is, E^{n-1}, we are done.

Let us try a different approach. Go back to Eq. (2.39). An alternative form of the Lorentz formulation is

$$S(\omega) = \frac{\gamma\beta}{(\alpha^2 + \beta^2) + j\omega 2\alpha + (j\omega)^2}\varepsilon_1 E(\omega), \tag{2.42}$$

where

$$\gamma = \frac{\omega_0}{\sqrt{1 - \delta_0^2}},$$
$$\alpha = \delta_0\omega_0,$$
$$\beta = \omega_0\sqrt{1 - \delta_0^2}.$$

At this point, we can go to Table A.1 and look up the corresponding Z transform, which is

$$S(z) = \frac{e^{-\alpha\cdot\Delta t}\cdot\sin(\beta\cdot\Delta t)\cdot\Delta t\cdot z^{-1}}{1 - 2e^{-\alpha\cdot\Delta t}\cdot\cos(\beta\cdot\Delta t)\cdot z^{-1} + e^{-2\alpha\cdot\Delta t}\cdot z^{-2}}\gamma\varepsilon_1 E(z).$$

The corresponding sampled time domain equation is

$$S^n = e^{-\alpha\cdot\Delta t}\cdot\cos(\beta\cdot\Delta t)S^{n-1} - e^{-2\alpha\cdot\Delta t}S^{n-2} + e^{-2\alpha\cdot\Delta t}\cdot\sin(\beta\cdot\Delta t)\cdot\Delta t\cdot\gamma\varepsilon_1 E(z). \tag{2.43}$$

If both methods are correct, they should yield the same answers, which means the corresponding terms in Eq. (2.41) and Eq. (2.43) should be the same. Let us see if that is true.

S^{n-2} term: We saw earlier that

$$e^{-2\delta_0\omega_0\cdot\Delta t} \cong \frac{1 - \delta_0\omega_0\cdot\Delta t}{1 + \delta_0\omega_0\cdot\Delta t}.$$

E^{n-1} term: If $\beta\cdot\Delta t \ll 1$, then $\sin(\beta\cdot\Delta t) \cong \beta\cdot\Delta t$, and

$$e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t)\cdot\gamma\cdot\Delta t = \frac{\beta\cdot\Delta t}{1 + \alpha\cdot\Delta t}\gamma\cdot\Delta t$$
$$= \frac{\omega_0\sqrt{1 - \delta_0^2}\cdot\Delta t}{1 + \delta_0\omega_0\cdot\Delta t}\frac{\omega_0}{\sqrt{1 - \delta_0^2}}\cdot\Delta t = \frac{\omega_0^2\cdot\Delta t^2}{1 + \delta_0\omega_0\cdot\Delta t}.$$

S^{n-1} term: By using a Taylor series expansion of the cosine function, we get

$$2e^{-\alpha\cdot\Delta t}\cos(\beta\cdot\Delta t)\cdot\gamma\cdot\Delta t = \frac{2}{1+\alpha\cdot\Delta t}\left[1-\frac{(\beta\cdot\Delta t)^2}{2}\right]$$
$$= \frac{2-(\beta\cdot\Delta t)^2}{1+\alpha\cdot\Delta t} = \frac{2-\omega_0^2\cdot(1-\delta_0^2)\Delta t^2}{1+\delta_0\omega_0\cdot\Delta t}.$$

The last step is perhaps our weakest one. It depends on δ_0 being small enough compared to 1 that δ_0^2 will be negligible.

2.5.1 Simulation of Human Muscle Tissue

We will conclude with the simulation of human muscle tissue. Muscle tissue can be adequately simulated over a frequency range of about two decades with the following formulation:

$$\varepsilon_r^*(\omega) = \varepsilon_r + \frac{\sigma}{j\omega\varepsilon_0} + \varepsilon_1\frac{\omega_0}{(\omega_0^2+\alpha^2)+j\alpha\omega+\omega^2}. \tag{2.44}$$

(Problem 2.5.2 addresses how one determines the specific parameters, but we will not be concerned with that here.) By inserting Eq. (2.44) into Eq. (2.1b) and taking the Z transform, we get

$$D(z) = \varepsilon_r E(z) + \frac{\sigma\cdot\dfrac{\Delta t}{\varepsilon_0}}{1-z^{-1}}E(z)$$
$$+\varepsilon_1\frac{e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t)\cdot\Delta t\cdot z^{-1}}{1-2e^{-\alpha\cdot\Delta t}\cos(\beta\cdot\Delta t)\cdot z^{-1}+e^{-2\alpha\cdot\Delta t}z^{-2}}E(z). \tag{2.45}$$

We will define two auxiliary parameters

$$I(z) = \frac{\sigma\cdot\dfrac{\Delta t}{\varepsilon_0}}{1-z^{-1}}E(z) \tag{2.46a}$$

$$S(z) = \varepsilon_1\frac{e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t)\cdot\Delta t\cdot z^{-1}}{1-2e^{-\alpha\cdot\Delta t}\cos(\beta\cdot\Delta t)\cdot z^{-1}+e^{-2\alpha\cdot\Delta t}z^{-2}}E(z). \tag{2.46b}$$

Now Eq. (2.45) becomes

$$D(z) = \varepsilon_r E(z) + I(z) + z^{-1}S(z). \tag{2.47}$$

Once we have calculated $E(z)$, $I(z)$ and $S(z)$ can be calculated from Eq. (2.46a) and (2.46b):

$$E(z) = \frac{D(z) - z^{-1}I(z) - z^{-1}S(z)}{\varepsilon_r + \sigma \cdot \dfrac{\Delta t}{\varepsilon_0}}, \tag{2.48a}$$

$$I(z) = z^{-1}I(z) + \sigma \cdot \frac{\Delta t}{\varepsilon_0}E(z), \tag{2.48b}$$

$$S(z) = 2e^{-\alpha\cdot\Delta t}\cos(\beta \cdot \Delta t) \cdot z^{-1}S(z) - e^{-2\alpha\cdot\Delta t}z^{-2}S(z) + \varepsilon_1 e^{-\alpha\cdot\Delta t}\sin(\beta \cdot \Delta t) \cdot \Delta t \cdot z^{-1}E(z). \tag{2.48c}$$

PROBLEM SET 2.5

1. Modify fd1d_2.3.c to simulate a Lorentz medium with the properties used in Fig. 2.6. Calculate the Fourier amplitudes at 50, 100, and 200 MHz. Considering Fig. 2.6, do you get the results you expect?

2. Quantify the results of problem 2.5.1 by comparing them with analytic calculations of the reflection coefficient and transmission coefficient at the three frequencies using Appendix 1.A.

3. FDTD simulation has been used extensively to model the effects of electromagnetic radiation on human tissue for both safety (8) and for therapeutic applications (9). Human muscle tissue is highly frequency dependent. Table 2.1 shows how the dielectric constant and the conductivity of muscle vary with frequency (10). Other tissues display similar frequency dependence (11).

Write a program to calculate the values of ε_r, σ, χ_1, and t_0 that would give an adequate Debye representation of the data in Table 2.1. You probably will not be able to fit the values exactly. (Hint: do *not* try for a purely analytic solution; do it by trial and error.)

TABLE 2.1 Properties of Human Muscle

Frequency, MHz	Dielectric Constant	Conductivity, S/m
10	160	0.625
40	97	0.693
100	72	0.89
200	56.5	1.28
300	54	1.37
433	53	1.43
915	51	1.6

4. Using the parameters found in problem 2.5.3, do a simulation of a pulse striking a medium of muscle tissue. Calculate the frequency domain results at 50, 100, and 433 MHz.

REFERENCES

1. C. M. Furse, S. P. Mathur, and O. P. Gandhi, Improvements to the finite-difference time-domain method for calculating the radar cross section of a perfectly conducting target, *IEEE Trans. Microw. Theor. Tech.*, vol. MTT-38, July 1990, pp. 919–927.
2. D. M. Sullivan, Mathematical methods for treatment planning in deep regional hyperthermia, *IEEE Trans. Microw. Theor. Tech.*, vol. MTT-39, May 1991, pp. 862–872.
3. R. Luebbers, F. Hunsberger, K. Kunz, R. Standler, and M. Schneider, S frequency-dependent finite-difference time-domain formulation for dispersive materials, *IEEE Trans. Electromagn. C.*, vol. EMC-32, August 1990, pp. 222–227.
4. R. M. Joseph, S.C. Hagness, and A. Taflove, Direct time integration of Maxwell's equations in linear dispersive media with absorption for scattering and propagation of femtosecond electromagnetic pulses, *Optic. Lett.*, vol. 16, September 1991, pp. 1412–1411.
5. O. P. Gandhi, B. Q. Gao, and Y. Y. Chen, A frequency-dependent finite-difference time-domain formulation for general dispersive media, *IEEE Trans. Microw. Theor. Tech.*, vol. MTT-41, April 1993, pp. 658–665.
6. D. M. Sullivan, Frequency-dependent FDTD methods using Z transforms, *IEEE Trans. Antenn. Propag.*, vol. AP-40, October 1992, pp. 1223–1230.
7. A. Ishimaru, *Electromagnetic Wave Propagation, Radiation, and Scattering*, Englewood Cliffs, NJ: Prentice Hall, 1991.
8. D. M. Sullivan, Use of the finite-difference time-domain method in calculating EM absorption in human tissues, *IEEE Trans. Biomed. Eng.*, vol. BME-34, February 1987, pp. 148–157.
9. D. M. Sullivan, Three dimensional computer simulation in deep regional hyperthermia using the finite-difference time-domain method, *IEEE Trans. Microw. Theor. Tech.*, vol. MTT-38, February 1990, pp. 204–211.
10. C. C. Johnson and A. W. Guy, Nonionizing electromagnetic wave effects in biological materials and systems, *Proc. IEEE*, vol. 60, June 1972, pp. 692–718.
11. M. A. Stuchly and S. S. Stuchly, Dielectric properties of biological substances—tabulated, *J. Microw. Power*, vol. 15, 1980, pp. 16–19.


```
/* FD2D_2.c.  1D FDTD simulation of a dielectric slab */

# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE 200

main ()
```

```
{
    float dx[KE],ex[KE],hy[KE],ix[KE];
    float ga[KE],gb[KE];
    int n,k,kc,ke,kstart,nsteps;
    float ddx,dt,T,epsz,epsilon,sigma;
    float t0,spread,pi,freq_in,arg,pulse;
    FILE *fp, *fopen();
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;

    pi  = 3.14159;
    kc = KE/2;              /* Center of the problem space */
    ddx = .01;              /* Cell size */
    dt =ddx/6e8;            /* Time steps */
    epsz = 8.8e-12;

    for ( k=0; k < KE; k++ ) {  /* Initialize to free space */
      ga[k] = 1.;
      gb[k] = 0.;
      ex[k] = 0.;
      dx[k] = 0.;
      hy[k] = 0.;
    }
       printf( "Dielectric starts at --> ");
       scanf("%d", &kstart);
       printf( "Epsilon --> ");
       scanf("%f", &epsilon);
       printf( "Conductivity --> ");
       scanf("%f", &sigma);
       printf("%d %6.2f  %6.2f  \n", kstart,epsilon, sigma);

    for ( k=kstart; k <= KE; k++ ) {
      ga[k] = 1./(epsilon + sigma*dt/epsz) ;
      gb[k] =  sigma*dt/epsz ;
    }

       for ( k=1; k <= KE; k++ )
       { printf( "%2d   %4.2f %4.2f\n",k,ga[k],gb[k]); }

    /* These parameters specify the input pulse */
    t0 = 50.0;
    spread = 20.0;
    T = 0;
    nsteps = 1;

 /* Main part of the program */

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
```

```
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;

       /* Calculate the Dx field */
       for ( k=1; k < KE; k++ )
       { dx[k] = dx[k] + 0.5*( hy[k-1] - hy[k] ) ; }

       /* Put a Gaussian pulse at the low end */

       freq_in = 3e8;
       /* pulse = sin( 2*pi*freq_in*T*dt); */
        pulse =  exp(-.5*(pow((t0-T)/spread,2.0)));
       dx[kc] =  dx[kc] + pulse;
       printf( "%5.1f %6.2f %6.2f\n",T,pulse,dx[kc]);

       /* Calculate Ex from Dx */
       for ( k=0; k < KE-1; k++ )
       { ex[k] = ga[k]*(dx[k] - ix[k] )  ;
         ix[k] = ix[k] + gb[k]*ex[k]    ;  }

       /* Boundary conditions */
       ex[0]      = ex_low_m2;
       ex_low_m2  = ex_low_m1;
       ex_low_m1  = ex[1];

       ex[KE-1]    = ex_high_m2;
       ex_high_m2  = ex_high_m1;
       ex_high_m1  = ex[KE-2];

       /* Calculate the Hy field */
       for ( k=0; k < KE-1; k++ )
       { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }

     }

       for ( k=0; k < KE; k++ )
       { printf( "%2d   %6.2f %6.2  \n",k,dx[k],ex[k]); }

         /* Write the E field out to a file "Ex" */
       fp = fopen( "Ex","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %6.3f \n",ex[k]); }
       fclose(fp);

       printf( "%5.1f \n",T);
}
```

```
}

/* FD1D_2.2.c. The Fourier Transform has been added.*/

# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE 200

main ()
{
    float dx[KE],ex[KE],hy[KE],ix[KE];
    float ga[KE],gb[KE];
    int n,m,k,kc,ke,kstart,nsteps;
    float ddx,dt,T,epsz,epsilon,sigma;
    float t0,spread,pi,pulse;
    FILE *fp, *fopen();
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;
    float real_pt[5][KE],imag_pt[5][KE];
    float freq[5],arg[5],ampn[5][KE],phasen[5][KE];
    float real_in[5],imag_in[5],amp_in[5],phase_in[5];
    float mag[KE];

    kc = KE/2;                 /* Center of the space */
    pi  = 3.14159;
    epsz = 8.8e-12;
    ddx = .01;                 /* Cells size */
    dt = ddx/6e8;              /* Time steps */
       printf(" %6.4f  %10.5e \n",ddx,dt);

    for ( k=1; k < KE; k++ ) {  /* Initialize to free space */
      ga[k] = 1.;
      gb[k] = 0.;
      dx[k] = 0.;
      ex[k] = 0.;
      hy[k] = 0.;
      ix[k] = 0.;
      mag[k] = 0.;

      for ( m=0; m <= 2; m++ ) {
      real_pt[m][k] = 0.;     /* Real and imaginary parts    */
      imag_pt[m][k] = 0.;     /* of the Fourier Transform */
      ampn[m][k]    = 0.;     /* Amplitude and phase of the  */
      phasen[m][k]  = 0.;     /*    Fourier Transforms       */
      }
    }
      for ( m=0; m <= 2; m++ ) {
```

```
      real_in[m] = 0.;     /*  Fourier Trans. of input pulse */
          imag_in[m] = 0.;
      }

    ex_low_m1 = 0.;
    ex_low_m2 = 0.;
    ex_high_m1 = 0.;
    ex_high_m2 = 0.;

      /* Parameters for the Fourier Transform */

       freq[0] = 100.e6;
       freq[1] = 200.e6;
       freq[2] = 500.e6;

       for ( n=0; n<= 2; n++ )
       {  arg[n] = 2*pi*freq[n]*dt;
       printf( "%2d %6.2f %7.5f \n",n,freq[n]*1e-6,arg[n]);
       }

       printf( "Dielectric starts at --> ");
       scanf("%d", &kstart);
       printf( "Epsilon --> ");
       scanf("%f", &epsilon);
       printf( "Conductivity --> ");
       scanf("%f", &sigma);
       printf("%d %6.2f  %6.2f  \n", kstart,epsilon, sigma);

    for ( k=kstart; k <= KE; k++ ) {
      ga[k] = 1./(epsilon + sigma*dt/epsz ) ;
      gb[k] =  sigma*dt/epsz ;
    }

       for ( k=1; k <= KE; k++ )
       { printf( "%2d   %6.2f %6.4f \n",k,ga[k],gb[k]); }

    /* These parameters specify the input pulse */
    t0 = 50.0;
    spread = 10.0;

    T = 0;
    nsteps = 1;

    /* Main part of the program */

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);
```

```
    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;

       /* Calculate the Dx field */
       for ( k=0; k < KE; k++ )
       { dx[k] = dx[k] + 0.5*( hy[k-1] - hy[k] ) ; }

       /* Initialize with a pulse */

        pulse =  exp(-.5*(pow((t0-T)/spread,2.0)));
       dx[25] =  dx[25] + pulse;
       printf( "%5.1f %6.2f %6.2f \n",T,pulse,dx[5]);

       /* Calculate Ex from Dx */
       for ( k=0; k < KE-1; k++ )
       { ex[k] = ga[k]*(dx[k] - ix[k] )  ;
         ix[k] = ix[k] + gb[k]*ex[k]    ;
       }

       /* Calculate the Fourier transform of Ex. */
       for ( k=0; k < KE; k++ )
           { mag[k] = mag[k] + pow(ex[k],2.);
             for ( m=0; m <= 4; m++ )
            {real_pt[m][k]=real_pt[m][k]+cos(arg[m]*T)*ex[k] ;
                                imag_pt[m][k]= imag_pt[m]
[k]-sin(arg[m]*T)*ex[k] ; }
       }

      /* Fourier Transform of the input pulse */
        if  (T < 100 ) {
             for ( m=0; m <= 2 ; m++ )
            { real_in[m] = real_in[m] + cos(arg[m]*T)*ex[10] ;
             imag_in[m] = imag_in[m] - sin(arg[m]*T)*ex[10] ; }

        }

       /* Boundary conditions */
       ex[0]      = ex_low_m2;
       ex_low_m2  = ex_low_m1;
       ex_low_m1  = ex[1];

       ex[KE-1]    = ex_high_m2;
       ex_high_m2  = ex_high_m1;
       ex_high_m1  = ex[KE-2];

       /* Calculate the Hy field */
       for ( k=0; k < KE-1; k++ )
```

```
      { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }

    }

  /* End of the main loop */

     /*  for ( k=0; k < KE; k++ )
       { printf( "%2d   %6.2f %6.2  \n",k,dx[k],ex[k]); }  */
         /* Write the E field out to a file "Ex" */
       fp = fopen( "Ex","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %6.3f \n",ex[k]); }
       fclose(fp);

  /* Calculate the amplitude and phase of each frequency */

  /* Amplitude and phase of the input pulse */
    for (m=0; m <= 2; m++ )
    {     amp_in[m] = sqrt( pow(imag_in[m],2.)
                          + pow(real_in[m],2.) );
        phase_in[m] = atan2( imag_in[m],real_in[m]);
        printf( "%d  Input Pulse : %8.4f %8.4f %8.4f  %7.2f\n",
   m,real_in[m],imag_in[m],amp_in[m],(180.0/pi)*phase_in[m]);

      for ( k=1; k < KE; k++ )
       {ampn[m][k] = (1./amp_in[m])*sqrt( pow(real_pt[m][k],2.)
                         + pow(imag_pt[m][k],2.) );
    phasen[m][k]=atan2(imag_pt[m][k],real_pt[m][k])-phase_in[m];
          printf( "%d  %8.4f %8.4f %8.5f  %8.2f  \n",
k,real_pt[m][k],imag_pt[m][k],ampn[m][k],(180.0/pi)*phasen[m]
[k]) ;
         }
    }
      /*  for ( k=1; k < KE; k++ )
             { printf( "%d  %6.3f %6.3f %6.3f \n",
              k,ampn[0][k]),ampn[1][k],ampn[2][k]; } */

      /* Write the amplitude field out to a file "Amp" */
         fp = fopen( "Amp0","w");
         for ( k=0; k < KE; k++ )
             { fprintf( fp,"  %8.5f \n",ampn[0][k]); }
         fclose(fp);
         fp = fopen( "Amp1","w");
         for ( k=0; k < KE; k++ )
             { fprintf( fp,"  %8.5f \n",ampn[1][k]); }
         fclose(fp);
         fp = fopen( "Amp2","w");
         for ( k=0; k < KE; k++ )
             { fprintf( fp,"  %8.5f \n",ampn[2][k]); }
```

```
         fclose(fp);
         fclose(fp);
         fp = fopen( "Real0","w");
         for ( k=0; k < KE; k++ )
             { fprintf( fp,"  %8.5f \n",real_pt[0][k]); }
         fclose(fp);
         fp = fopen( "Imag0","w");
         for ( k=0; k < KE; k++ )
             { fprintf( fp,"  %8.5f \n",imag_pt[0][k]); }
         fclose(fp);

       printf( "%5.1f \n",T);
}

}

/* FD1D_2.3. 1D FDTD simulation of a frequency dependent material
*/

# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE 200

main ()
{
    float dx[KE],ex[KE],hy[KE],ix[KE],sx[KE];
    float ga[KE],gb[KE],gc[KE];
    int n,m,k,kc,ke,kstart,nsteps;
    float ddx,dt,T,epsz,epsilon,sigma;
    float t0,spread,pi,pulse;
    FILE *fp, *fopen();
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;
    float tau,chi1,del_exp;
    float real_pt[5][KE],imag_pt[5][KE];
    float freq[5],arg[5],ampn[5][KE],phasen[5][KE];
    float real_in[5],imag_in[5],amp_in[5],phase_in[5];
    float mag[KE];

    kc = KE/2;                 /* Center of the space */
    pi  = 3.14159;
    epsz = 8.8e-12;
    ddx = .01;               /* Cells size */
    dt = ddx/6e8;              /* Time steps */
       printf(" %6.4f  %10.5e \n",ddx,dt);

    for ( k=1; k < KE; k++ ) {  /* Initialize to free space */
      ga[k] = 1.;
```

```
      gb[k] = 0.;
      gc[k] = 0.;
      dx[k] = 0.;
      ex[k] = 0.;
      hy[k] = 0.;
      ix[k] = 0.;
      sx[k] = 0.;

      for ( m=0; m <= 4; m++ ) {
      real_pt[m][k] = 0.;     /* Real and imaginary parts    */
      imag_pt[m][k] = 0.;     /* of the Fourier Transform */
      ampn[m][k]    = 0.;     /* Amplitude and phase of the  */
      phasen[m][k]  = 0.;     /*    Fourier Transforms       *
      }
    }
      for ( m=0; m <= 4; m++ ) {
    real_in[m] = 0.;     /*    Fourier Trans. of input pulse */
    imag_in[m] = 0.;
      }

    ex_low_m1 = 0.;
    ex_low_m2 = 0.;
    ex_high_m1 = 0.;
    ex_high_m2 = 0.;

      /* Parameters for the Fourier Transform */

       freq[0] = 50.e6;
       freq[1] = 200.e6;
       freq[2] = 500.e6;

       for ( n=0; n<= 2; n++ )
       {  arg[n] = 2*pi*freq[n]*dt; printf( "%2d %6.2f %7.5f
%12.0f\n",n,freq[n]*1e-6,arg[n],1/arg[n]);
       }

       printf( "Dielectric starts at --> ");
       scanf("%d", &kstart);
       printf( "Epsilon --> ");
       scanf("%f", &epsilon);
       printf( "Conductivity --> ");
       scanf("%f", &sigma);
       printf( "chi1 --> ");
       scanf("%f", &chi1);
       tau = 1000.;                /* Make sure tau is > 0. */
       if(chi1 > 0.0001 ) {
       printf( "tau (in microseconds) --> ");
       scanf("%f", &tau);
          del_exp = exp(-dt/tau); }
```

```
   printf("%d %6.2f  %6.2f  %6.2f  %6.2f\n", kstart,epsilon,
sigma,tau,chi1);
       tau = 1.e-6*tau;
       { printf( "del_exp =  %8.5f \n",del_exp); }

    for ( k=kstart; k <= KE; k++ ) {
      ga[k] = 1./(epsilon + sigma*dt/epsz + chi1*dt/tau) ;
      gb[k] =  sigma*dt/epsz ;
      gc[k] =  chi1*dt/tau ;
    }

       for ( k=1; k <= KE; k++ )
 { printf( "%2d   %6.2f %6.4f %6.4f \n",k,ga[k],gb[k],gc[k]); }

    /* These parameters specify the input pulse */
    t0 = 50.0;
    spread = 10.0;

    T = 0;
    nsteps = 1;
   /* Main part of the program */

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;

       /* Calculate the Dx field */
       for ( k=0; k < KE; k++ )
       { dx[k] = dx[k] + 0.5*( hy[k-1] - hy[k] ) ; }

       /* Initialize with a pulse */

        pulse =  exp(-.5*(pow((t0-T)/spread,2.0)));
       dx[5] =  dx[5] + pulse;
       printf( "%5.1f %6.2f %6.2f \n",T,pulse,dx[5]);

       /* Calculate Ex from Dx */
       for ( k=0; k < KE-1; k++ )
       { ex[k] = ga[k]*(dx[k] - ix[k] - sx[k])  ;
         ix[k] = ix[k] + gb[k]*ex[k]    ;
         sx[k] = del_exp*sx[k] + gc[k]*ex[k] ; }

/* Calculate the Fourier transform of Ex. */
    for ( k=0; k < KE; k++ )
```

```
      {  for ( m=0; m <= 2; m++ )
       { real_pt[m][k] = real_pt[m][k] + cos(arg[m]*T)*ex[k] ;
        imag_pt[m][k] = imag_pt[m][k] - sin(arg[m]*T)*ex[k] ; }
      }

       /* Fourier Transform of the input pulse */
       if  (T < 3*t0 ) {
            for ( m=0; m <= 2 ; m++ )
            { real_in[m] = real_in[m] + cos(arg[m]*T)*ex[10] ;
            imag_in[m] = imag_in[m] - sin(arg[m]*T)*ex[10] ; }
       }

      /* Boundary conditions */
      ex[0]      = ex_low_m2;
      ex_low_m2  = ex_low_m1;
      ex_low_m1  = ex[1];

      ex[KE-1]    = ex_high_m2;
      ex_high_m2  = ex_high_m1;
      ex_high_m1  = ex[KE-2];

      /* Calculate the Hy field */
      for ( k=0; k < KE-1; k++ )
      { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }

   }

 /* End of the main loop */

    /*  for ( k=0; k < KE; k++ )
      { printf( "%2d   %6.2f %6.2  \n",k,dx[k],ex[k]); }  */

        /* Write the E field out to a file "Ex" */
      fp = fopen( "Ex","w");
      for ( k=0; k < KE; k++ )
      { fprintf( fp,"  %8.5f \n",ex[k]); }
      fclose(fp);

 /* Calculate the amplitude and phase of each frequency */

 /* Amplitude and phase of the input pulse */
   for (m=0; m <= 2; m++ )
   {     amp_in[m] = sqrt( pow(imag_in[m],2.)
                         + pow(real_in[m],2.) );
         phase_in[m] = atan2( imag_in[m],real_in[m]);
      printf( "%d  Input Pulse :  %8.4f %8.4f %8.4f  %7.2f\n",
                              m,real_in[m],imag_in[m],amp_in
[m],(180.0/pi)*phase_in[m]);
```

```
for ( k=1; k < KE; k++ )
 { ampn[m][k] = (1./amp_in[m])*sqrt( pow(real_pt[m][k],2.)
                          + pow(imag_pt[m][k],2.) );
   phasen[m][k]=atan2(imag_pt[m][k],real_pt[m][k])-phase_in[m];
     /*     printf( "%d  %8.4f %8.4f %8.5f  %8.2f  \n",
                      k,real_pt[m][k],imag_pt[m][k],ampn[m]
 [k],(180.0/pi)*phasen[m][k]) ;*/
        }
    }

    /*  for ( k=kstart; k < KE; k++ )
             { printf( "%d  %6.3f %6.3f %6.3f \n",
              k,ampn[1][k]),ampn[2][k],ampn[3][k]; } */

      /* Write the amplitude field out to a file "Amp" */
         fp = fopen( "Amp0","w");
         for ( k=1; k < KE; k++ )
             { fprintf( fp,"  %6.2f \n",ampn[0][k]); }
         fclose(fp);
         fp = fopen( "Amp1","w");
         for ( k=1; k < KE; k++ )
             { fprintf( fp,"  %6.2f \n",ampn[1][k]); }
         fclose(fp);
         fp = fopen( "Amp2","w");
         for ( k=1; k < KE; k++ )
             { fprintf( fp,"  %6.2f \n",ampn[2][k]); }
         fclose(fp);

       printf( "%5.1f \n",T);
}

}
```

3

TWO-DIMENSIONAL SIMULATION

This chapter introduces two-dimensional simulation. It begins with the basic two-dimensional formulation in FDTD and a simple example using a point source. Then the absorbing boundary conditions (ABCs) are described, along with their implementation into the FDTD program. Finally, the generation of electromagnetic plane waves using FDTD is described.

3.1 FDTD IN TWO DIMENSIONS

Once again, we will start with the normalized Maxwell's equations that we used in Chapter 2:

$$\frac{\partial \widetilde{\boldsymbol{D}}}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \widetilde{\boldsymbol{H}}, \tag{3.1a}$$

$$\widetilde{\boldsymbol{D}}(\omega) = \varepsilon_{\mathrm{r}}^{*}(\omega) \cdot \widetilde{\boldsymbol{E}}(\omega), \tag{3.1b}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \widetilde{\boldsymbol{E}}. \tag{3.1c}$$

When we get to three-dimensional simulation, we will wind up dealing with six different fields: $\widetilde{\boldsymbol{E}}_x, \widetilde{\boldsymbol{E}}_y, \widetilde{\boldsymbol{E}}_z, \widetilde{\boldsymbol{H}}_x, \widetilde{\boldsymbol{H}}_y$, and $\widetilde{\boldsymbol{H}}_z$. For two-dimensional simulation, we choose between one of the two groups of three vectors each: (1) the transverse

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

magnetic (TM) mode, which is composed of $\widetilde{\boldsymbol{E}}_z$, $\widetilde{\boldsymbol{H}}_x$, and $\widetilde{\boldsymbol{H}}_y$, or (2) the transverse electric (TE) mode, which is composed of $\widetilde{\boldsymbol{E}}_x$, $\widetilde{\boldsymbol{E}}_y$, and $\widetilde{\boldsymbol{H}}_z$. We will work with the TM mode. Equation (3.1) is now reduced to

$$\frac{\partial D_z}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right), \tag{3.2a}$$

$$\boldsymbol{D}(\omega) = \varepsilon_{\mathrm{r}}^*(\omega) \cdot \boldsymbol{E}(\omega), \tag{3.2b}$$

$$\frac{\partial H_x}{\partial t} = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \frac{\partial E_z}{\partial y}, \tag{3.2c}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \frac{\partial E_z}{\partial x}. \tag{3.2d}$$

(Once again, we have dropped the tilde over the E and D fields. But it is understood that we are using the normalized units described in Chapter 1.)

As in one-dimensional simulation, it is important that there is a systematic interleaving of the fields to be calculated. This is illustrated in Fig. 3.1. Putting Eq. (3.2a), (3.2c), and (3.2d) into the finite-differencing scheme results in the following difference equations (1):

$$\begin{aligned} \frac{D_z^{n+1/2}(i,j) - D_z^{n-1/2}(i,j)}{\Delta t} = {} & \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{H_y^n\left(i+\frac{1}{2}, j\right) - H_y^n\left(i-\frac{1}{2}, j\right)}{\Delta x} \right) \\ & - \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{H_x^n\left(i, j+\frac{1}{2}\right) - H_x^n\left(i, j-\frac{1}{2}\right)}{\Delta x} \right), \end{aligned} \tag{3.3a}$$

$$\begin{aligned} & \frac{H_x^{n+1}\left(i+\frac{1}{2}, j\right) - H_x^n\left(i+\frac{1}{2}, j\right)}{\Delta t} \\ & \quad = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \frac{E_z^{n+1/2}(i, j+1) - E_z^{n+1/2}(i, j)}{\Delta x}, \end{aligned} \tag{3.3b}$$

$$\begin{aligned} & \frac{H_y^{n+1}\left(i+\frac{1}{2}, j\right) - H_y^n\left(i-\frac{1}{2}, j\right)}{\Delta t} \\ & \quad = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \frac{E_z^{n+1/2}(i+1, j) - E_z^{n+1/2}(i, j)}{\Delta x}. \end{aligned} \tag{3.3c}$$

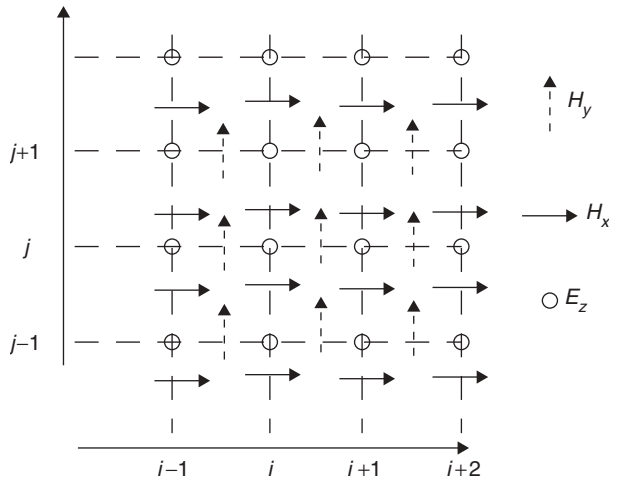

Figure 3.1 Interleaving of the E and H fields for the two-dimensional TM formulation.

Using the same type of manipulation as in Chapter 1, including

$$\Delta t = \frac{\Delta x}{2 \cdot c_0},$$

we get the equations:

```
Dz[i][j]=dz[i][j]
          +.5*(hy[i][j]-hy[i-i][j]-hx[i][j]+hx[i][j-1]);
```
(3.4a)

```
Ez[i][j]=gaz[i][j]*(dz[i][j]-iz[i][j]);
```
(3.4b)

```
iz[i][j]=iz[i][j]+gbz[i][j]*ez[i][j];
```
(3.4c)

```
hx[i][j]=hx[i][j]+.5*(ez[i][j]-ez[i][j+1]);
```
(3.4d)

```
hy[i][j]=hy[i][j]+.5*(ez[+1][j]-ez[i][j]);
```
(3.4e)

Note that the relationship between E_z and D_z is the same as that for the simple lossy dielectric in the one-dimensional case. Obviously, the same modification can be made to include frequency-dependent terms.

The program fd2d_3.1 implements the above equations. It has a simple Gaussian pulse source that is generated in the middle of the problem space. Figure 3.2 presents the results of a simulation for the first 50 time steps.

PROBLEM SET 3.1

1. Get the program fd2d_3.1.c running. Duplicate the results of Fig. 3.2. Let it run until it hits the boundary. What happens?

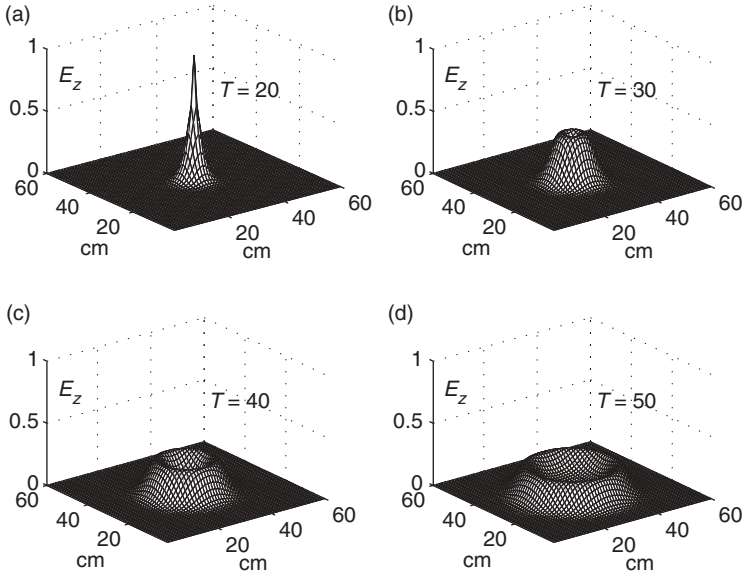

Figure 3.2 Results of a simulation using the program fd2d_3.1.c. A Gaussian pulse is initiated in the middle and travels outward. (a) $T = 20$, (b) $T = 30$, (c) $T = 40$, and (d) $T = 50$.

3.2 THE PERFECTLY MATCHED LAYER (PML)

Until now, we have only briefly mentioned the issue of ABCs. The size of the area that can be simulated using FDTD is limited by computer resources. For instance, in the two-dimensional simulation of the previous section, the program contains two-dimensional matrices for the values of all the fields, `dz, ez, hx,` and `hy,` as well as matrices to hold the parameters `gaz, gbz,` and `iz`.

Suppose we are simulating a wave generated from a point source propagating in free space as in Fig. 3.2. As the wave propagates outward, it will eventually come to the edge of the allowable space, which is dictated by how the matrices have been dimensioned in the program. If nothing were done to address this, reflections would be generated that propagate inward. There would be no way to determine the real wave from the reflected junk. This is the reason that ABCs have been an issue for as long as FDTD has been used. There have been numerous approaches to this problem (2, 3).

The most flexible and efficient ABC is the perfectly matched layer (PML) developed by Berenger (4). The basic idea behind the PML is that if a wave is propagating in medium A and it impinges on medium B, the amount of reflection is dictated by the intrinsic impedances of the two media as given by the equation

$$\Gamma = \frac{\eta_A - \eta_B}{\eta_A + \eta_B}, \tag{3.5}$$

where Γ is the reflection coefficient and η_A and η_B are the impedances of the media as calculated from the dielectric constants ε and the permeability μ of the two media:

$$\eta = \sqrt{\frac{\mu}{\varepsilon_r \varepsilon_0}}. \tag{3.6}$$

So far, we have assumed that μ was a constant, so when a propagating pulse went from $\varepsilon_r = 1$ to $\varepsilon_r = 4$ as in Fig. 2.1, it saw a change in impedance and reflected a portion of the pulse, given by Eq. (3.5). However, if μ changed with ε so that η remained a constant, Γ would remain constant and no reflection would occur. This still does not solve our problem because the pulse will continue propagating in the new medium. What we really want is a medium that is also lossy so the pulse will die out before it reaches the end of the problem space. This is accomplished by making both ε and μ complex because the imaginary part represents the part that causes decay (Appendix 1.A).

Let us go back to Eq. (3.2) but move everything to the Fourier domain. (We are going to the Fourier domain in *time*, so $\mathrm{d}/\mathrm{d}t$ becomes $j\omega$. This does not affect the spatial derivatives.)

$$j\omega D_z = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right), \tag{3.7a}$$

$$D_z(\omega) = \varepsilon_r^*(\omega) \cdot E_z(\omega), \tag{3.7b}$$

$$j\omega H_x = -c_0 \frac{\partial E_z}{\partial y}, \tag{3.7c}$$

$$j\omega H_y = c_0 \frac{\partial E_z}{\partial x}. \tag{3.7d}$$

Remember that we have eliminated ε and μ from the spatial derivatives in Eq. (3.7a), (3.7c), and (3.7d) for the normalized units. Instead of putting them back to implement the PML, we will add the fictitious dielectric constants and permeabilities $\varepsilon_{F_z}^*$, $\mu_{F_x}^*$, and $\mu_{F_y}^*$(5):

$$j\omega D_z \cdot \varepsilon_{F_z}^*(x) \cdot \varepsilon_{F_z}^*(y) = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right), \tag{3.8a}$$

$$D_z(\omega) = \varepsilon_r^*(\omega) \cdot E_z(\omega), \tag{3.8b}$$

$$j\omega H_x \cdot \mu_{F_x}^*(x) \cdot \mu_{F_y}^*(y) = -c_0 \frac{\partial E_z}{\partial y}, \tag{3.8c}$$

$$j\omega H_y \cdot \mu_{F_y}^*(x) \cdot \mu_{F_y}^*(y) = c_0 \frac{\partial E_z}{\partial x}. \tag{3.8d}$$

A few things are worth noting: (i) the value ε_F is associated with the flux density D, not the electric field E and (ii) we have added two values each of ε_F in Eq. (3.8a) and μ_F in Eq. (3.8c) and (3.8d). These fictitious values to implement

the PML have nothing to do with the *real* values of $\varepsilon_r^*(\omega)$, which specify the medium.

Sacks et al. (6) showed that there are two conditions to form a PML:

1. The impedance going from the background medium to the PML must be constant,
$$\eta_0 = \eta_m = \sqrt{\frac{\mu_{F_x}}{\varepsilon_{F_x}^*}} = 1. \tag{3.9}$$
The impedance is 1 because of our normalized units.
2. In the direction perpendicular to the boundary (e.g., the x direction), the relative dielectric constant and relative permeability must be the inverse of those in the other directions, that is,
$$\varepsilon_{F_x}^* = \frac{1}{\varepsilon_{F_y}^*} \tag{3.10a}$$
$$\mu_{F_x}^* = \frac{1}{\mu_{F_y}^*}. \tag{3.10b}$$
We will assume that each of these is a complex quantity of the form
$$\varepsilon_{F_m}^* = \varepsilon_{F_m} + \frac{\sigma_{D_m}}{j\omega\varepsilon_0} \quad \text{for } m = x \text{ or } y \tag{3.11a}$$
$$\mu_{F_m}^* = \mu_{F_m} + \frac{\sigma_{D_m}}{j\omega\mu_0} \quad \text{for } m = x \text{ or } y. \tag{3.11b}$$

The following selection of parameters satisfies Eq. (3.10a) and (3.10b) (7):

$$\varepsilon_{F_m} = \mu_{F_m} = 1 \tag{3.12a}$$
$$\frac{\sigma_{D_m}}{\varepsilon_0} = \frac{\sigma_{H_m}}{\mu_0} = \frac{\sigma_D}{\varepsilon_0}. \tag{3.12b}$$

Substituting Eq. (3.12) into Eq. (3.11), the value in Eq. (3.9) becomes

$$\eta_0 = \eta_m = \sqrt{\frac{\mu_{F_x}^*}{\varepsilon_{F_x}^*}} = \sqrt{\frac{\dfrac{1+\sigma(x)}{j\omega\varepsilon_0}}{\dfrac{1+\sigma(x)}{j\omega\varepsilon_0}}} = 1.$$

This fulfills the first requirement mentioned earlier. If σ increases gradually as it goes into the PML, Eq. (3.8a), (3.8c), and (3.8d) will cause D_z and H_y to be attenuated.

We will start by implementing a PML only in the X direction. Therefore, we will retain only the x-dependent values of $\varepsilon_{\mathrm{F}}^*$ and μ_{F}^* in Eq. (3.8)

$$j\omega D_z \cdot \varepsilon_{\mathrm{F}_z}^*(x) = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right),$$

$$j\omega H_x \cdot \mu_{\mathrm{F}_x}^*(x) = -c_0 \frac{\partial E_z}{\partial y},$$

$$j\omega H_y \cdot \mu_{\mathrm{F}_y}^*(x) = c_0 \frac{\partial E_z}{\partial x},$$

and use the values of Eq. (3.12):

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\varepsilon_0}\right) D_z = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right), \tag{3.13a}$$

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\varepsilon_0}\right)^{-1} H_x = -c_0 \frac{\partial E_z}{\partial y}, \tag{3.13b}$$

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\varepsilon_0}\right) H_y = c_0 \frac{\partial E_z}{\partial x}. \tag{3.13c}$$

Note that the permeability of H_x in Eq. (3.13b) is the inverse of that of H_y in Eq. (3.13c) in keeping with Eq. (3.10b). Therefore, we have fulfilled the second requirement for the PML. (Equation (3.10a) is irrelevant for this two-dimensional case because we only have an E field in the z direction, which is perpendicular to both x and y, the directions of propagation.)

Now Eq. (3.13) must be put into the FDTD formulation. First, look at the left side of Eq. (3.13a):

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\varepsilon_0}\right) D_z = j\omega D + \frac{\sigma_D(x)}{\varepsilon_0} D_z.$$

Moving to the time domain and taking the finite-difference approximations, we get the following:

$$j\omega D_z + \frac{\sigma_D(x)}{\varepsilon_0} D_z \cong \frac{D_z^{n+1/2}(i,j) - D_z^{n-1/2}}{\Delta t} + \frac{\sigma_D(i)}{\varepsilon_0} \frac{D_z^{n+1/2}(i,j) + D_z^{n-1/2}(i,j)}{2}$$

$$= D_z^{n+1/2}(i,j) \frac{1}{\Delta t}\left(1 + \frac{\sigma_D(i)\cdot\Delta t}{2\cdot\varepsilon_0}\right) - D_z^{n-1/2}(i,j) \frac{1}{\Delta t}\left(1 - \frac{\sigma_D(i)\cdot\Delta t}{2\cdot\varepsilon_0}\right).$$

If we put this into Eq. (3.13a), along with the spatial derivatives, we get

$$\begin{aligned} D_z^{n+1/2}(i,j) = {} & gi3(i) \cdot D_z^{n-1/2}(i,j) \\ & + gi2(i) \cdot 0.5 \cdot y[H_y^n(i+1/2,j) - H_y^n(i-1/2,j) \\ & - H_y^n(i,j+1/2) + H_y^n(i,j-1/2)]. \end{aligned} \tag{3.14}$$

Once again we have used the fact that

$$\frac{\Delta t}{\Delta x}c_0 = \frac{\left(\frac{\Delta x}{2c_0}\right)}{\Delta x}c_0 = \frac{1}{2}.$$

The new parameters $gi2$ and $gi3$ are given by

$$gi2(i) = \frac{1}{1 + \sigma_D(i) \cdot \frac{\Delta t}{2\varepsilon_0}} \tag{3.15a}$$

$$gi3(i) = \frac{1 - \sigma_D(i) \cdot \frac{\Delta t}{2\varepsilon_0}}{1 + \sigma_D(i) \cdot \frac{\Delta t}{2\varepsilon_0}}. \tag{3.15b}$$

An almost identical treatment of Eq. (3.13c) gives

$$\begin{aligned} H_y^{n+1}\left(i + \frac{1}{2}, j\right) = {} & fi3\left(i + \frac{1}{2}\right) \cdot H_y^n\left(i + \frac{1}{2}, j\right) \\ & + gi2\left(i + \frac{1}{2}\right) \cdot 0.5 \cdot (D_z^{n+1/2}(i + 1, j) \\ & - D_z^{n+1/2}(i, j)), \end{aligned} \tag{3.16}$$

where

$$fi2\left(i + \frac{1}{2}\right) = \frac{1}{1 + \sigma_D\left(i + \frac{1}{2}\right) \cdot \frac{\Delta t}{2\varepsilon_0}}, \tag{3.17a}$$

$$fi3\left(i + \frac{1}{2}\right) = \frac{1 - \sigma_D\left(i + \frac{1}{2}\right) \cdot \frac{\Delta t}{2\varepsilon_0}}{1 + \sigma_D\left(i + \frac{1}{2}\right) \cdot \frac{\Delta t}{2\varepsilon_0}}. \tag{3.17b}$$

Notice that these parameters are calculated at $i + 1/2$ because of the position of H_y in the FDTD grid (Fig. 3.1).

Equation (3.13b) will require a somewhat different treatment than the previous two equations. Start by rewriting it as

$$j\omega H_x = -c_0\left(\frac{\partial E_z}{\partial y} + \frac{\sigma_D(x)}{j\omega\varepsilon_0}\frac{\partial E_z}{\partial y}\right).$$

Remember that $(1/j\omega)$ can be regarded as an integration operator in time and $j\omega$ as a derivative in time. The spatial derivative will be written as

$$\frac{\partial E_z}{\partial y} \cong \frac{E_z^{n+1/2}(i, j+1) - E_z^{n+1/2}(i, j)}{\Delta x} = -\frac{curl\_x}{\Delta x}.$$

Implementing this into an FDTD formulation gives

$$\frac{H_x^{n+1}\left(i, j+\frac{1}{2}\right) - H_x^n\left(i, j+\frac{1}{2}\right)}{\Delta t} = -c_0\left(-\frac{curl\_x}{\Delta x} - \frac{\sigma_D(i)}{\varepsilon_0}\Delta t\sum_{n=0}^{T}\frac{curl\_x}{\Delta x}\right).$$

Note the extra Δt in front of the summation. This is part of the approximation of the time domain integral. Finally, we get

$$H_x^{n+1}\left(i, j+\frac{1}{2}\right) = H_x^n\left(i, j+\frac{1}{2}\right) + \frac{c_0\cdot\Delta t}{\Delta x} + \frac{c_0\cdot\Delta t}{\Delta x}\frac{\sigma_D(i)}{\varepsilon_0} I_{H_x}^{n+1/2}\left(i, j+\frac{1}{2}\right).$$

Equation (3.13b) is implemented as the following series of equations:

$$curl\_e = (E_z^{n+1/2}(i, j) - E_z^{n+1/2}(i, j+1)), \tag{3.18a}$$

$$I_{H_x}^{n+1/2}\left(i, j+\frac{1}{2}\right) = I_{H_x}^{n-1/2}\left(i, j+\frac{1}{2}\right) + curl\_e, \tag{3.18b}$$

$$H_x^{n+1}\left(i, j+\frac{1}{2}\right) = H_x^n\left(i, j+\frac{1}{2}\right) + curl\_e + fi1(i)\cdot I_x^{n+1/2}\left(i, j+\frac{1}{2}\right), \tag{3.18c}$$

with

$$fi1(i) = \frac{\sigma(i)\cdot\Delta t}{2\varepsilon_0}. \tag{3.19}$$

In calculating the f and g parameters, it is not necessary to actually vary the conductivities. Instead, we calculate an auxiliary parameter,

$$xn = \frac{\sigma(i)\cdot\Delta t}{2\varepsilon_0},$$

which increases as it goes into the PML. The f and g parameters are then calculated:

$$xn(i) = 0.333\left(\frac{i}{\text{length\_pml}}\right)^3 \quad i = 1, 2, \ldots, \text{length\_pml}. \tag{3.20}$$

$$fi1(i) = xn, \tag{3.21a}$$

$$gi2(i) = \left(\frac{1}{1 + xn\,(i)}\right), \tag{3.21b}$$

$$gi3(i) = \left(\frac{1 - xn\,(i)}{1 + xn(i)}\right). \tag{3.21c}$$

Notice that the quantity in the parentheses in Eq. (3.20) ranges between 0 and 1. The factor 0.333 was found empirically to be the largest number that remained stable. Similarly, the cubic factor in Eq. (3.20) was found empirically to be the most effective variation. (The parameters $fi2$ and $fi3$ are different only because they are computed at the half intervals, $i + 1/2$.) The parameters vary in the following manner:

$$fi1(i) \quad \text{from 0 to 0.333}, \tag{3.22a}$$

$$gi2(i) \quad \text{from 1 to 0.75}, \tag{3.22b}$$

$$gi3(i) \quad \text{from 1 to 0.5}. \tag{3.22c}$$

Throughout the main problem space, $fi1$ is 0, while $gi2$ and $gi3$ are 1. Therefore, there is a seamless transition from the main part of the program to the PML (Fig. 3.3).

So far, we have shown the implementation of the PML in the x direction. Obviously, it must also be done in the y direction. To do this, we must go back and add the y-dependent terms from Eq. (3.8) that were set aside. So instead of

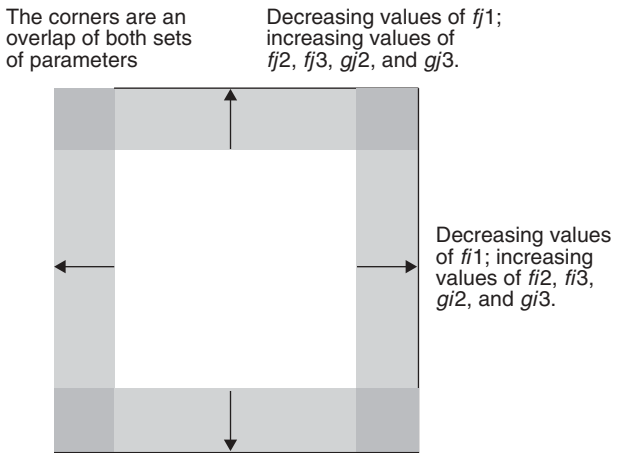

Figure 3.3 Parameters related to the perfectly matched layer (PML).

Eq. (3.13), we have

$$j\omega\left(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\right)\left(1+\frac{\sigma_D(y)}{j\omega\varepsilon_0}\right)D_z = c_0\cdot\left(\frac{\partial H_y}{\partial x}-\frac{\partial H_x}{\partial y}\right), \tag{3.23a}$$

$$j\omega\left(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\right)^{-1}\left(1+\frac{\sigma_D(y)}{j\omega\varepsilon_0}\right)H_x = -c_0\frac{\partial E_z}{\partial y}, \tag{3.23b}$$

$$j\omega\left(1+\frac{\sigma_D(x)}{j\omega\varepsilon_0}\right)\left(1+\frac{\sigma_D(y)}{j\omega\varepsilon_0}\right)^{-1}H_y = c_0\frac{\partial E_z}{\partial x}. \tag{3.23c}$$

Using the same procedure as before, the following replaces Eq. (3.14):

$$D_z^{n+1/2}(i,j) = gi3(i)\cdot gj3(j)\cdot D_z^{n-1/2}(i,j) + gi2(i)\cdot gj2(j)\cdot 0.5$$
$$\left[H_y^n\left(i+\frac{1}{2},j\right)-H_y^n\left(i-\frac{1}{2},j\right)-H_y^n\left(i,j+\frac{1}{2}\right)+H_y^n\left(i,j-\frac{1}{2}\right)\right].$$

In the Y direction, H_y will require an implementation similar to the one used for H_x in the X direction giving,

$$curl\_e = (E_z^{n+1/2}(i+1,j) - E_z^{n+1/2}(i,j)), \tag{3.24a}$$

$$I_{H_y}^{n+1/2}\left(i+\frac{1}{2},j\right) = I_{H_y}^{n-1/2}\left(i+\frac{1}{2},j\right) + curl\_e, \tag{3.24b}$$

$$\begin{aligned} H_y^{n+1}\left(i+\frac{1}{2},j\right) = {} & fi3\left(i+\frac{1}{2}\right)H_y^n\left(i+\frac{1}{2},j\right) \\ & - fi2\left(i+\frac{1}{2}\right)\cdot 0.5\cdot curl\_e \\ & - fj1(j)\cdot I_{H_y}^{n+1/2}\left(i+\frac{1}{2},j\right). \end{aligned} \tag{3.24c}$$

Finally, the H_x in the x direction becomes

$$curl\_e = (E_z^{n+1/2}(i,j) - E_z^{n+1/2}(i,j+1)),$$

$$I_{H_x}^{n+1/2}\left(i,j+\frac{1}{2}\right) = I_{H_x}^{n-1/2}\left(i,j+\frac{1}{2}\right) + curl\_e,$$

$$\begin{aligned} H_x^{n+1}\left(i,j+\frac{1}{2}\right) = {} & fj3\left(j+\frac{1}{2}\right)\cdot H_x^n\left(i,j+\frac{1}{2}\right) \\ & + fj2\left(i+\frac{1}{2}\right)\cdot 0.5\cdot curl\_e \\ & + fi1(i)\cdot I_{H_x}^{n+1/2}\left(i,j+\frac{1}{2}\right). \end{aligned}$$

Now the full set of parameters associated with the PML are the following:

$$fi1(i) \text{ and } fj1(j) \quad \text{from 0 to 0.333,} \tag{3.25a}$$

$$fi2(2), gi2(i), fj2(j), \text{ and } gj2(j) \quad \text{from 1 to 0.75,} \tag{3.25b}$$

$$fi3(i), gi3(i), fj3(j), \text{ and } gj3(j) \quad \text{from 1 to 0.5.} \tag{3.25c}$$

Notice that we could simply turn off the PML in the main part of the problem space by setting $fi1$ and $fj1$ to 0 and the other parameters to 1. They are only one-dimensional parameters, so they add very little to the memory requirements. However, I_{H_x} and I_{H_y} are two-dimensional parameters. Although memory requirements are not a main issue while we are in two dimensions, when we get to three dimensions, we will think twice before introducing two new parameters that are defined throughout the problem space, but are needed only in a small fraction of the space.

The PML is implemented in the program fd2d_3.2.c. Figure 3.4 illustrates the effectiveness of an eight-point PML, with the source offset five cells from the center in both the X and Y directions. Note that the outgoing contours remain

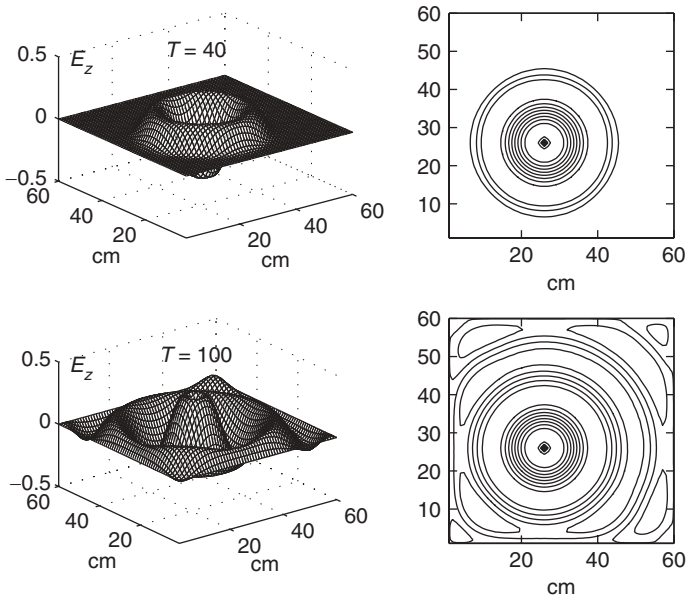

Figure 3.4 Results of a simulation using the program fd2d_3.2.c. A sinusoidal source is initiated at a point that is offset five cells from the center of the problem space in each direction. As the wave reaches the perfectly matched layer (PML), which is eight cells on every side, it is absorbed. The effectiveness of the PML is apparent in the bottom figure because the contours would not be concentric circles if the outgoing wave was partially reflected.

concentric. Only when the wave gets within eight points of the edge, which is inside the PML, does distortion begin.

PROBLEM SET 3.2

1. The program fd2d_3.2.c is the same as fd2d_3.1.c, but with the two-dimensional PML added. Add the PML to your version of fd2d_3.1.c. Offset each point source by setting `ic = IE/2-5` and `jc = JE/2-5`. Verify the results in Fig. 3.4.

3.3 TOTAL/SCATTERED FIELD FORMULATION

The simulation of plane waves is often of interest in computational electromagnetic. Many problems, such as the calculation of radar cross sections (2, 3), deal with plane waves. Furthermore, after a distance on the order of tens of wavelengths, the field from most antennas can be approximated as a plane wave.

In order to simulate a plane wave in a two-dimensional FDTD program, the problem space will be divided into two regions, the *total field* and the *scattered field* (Fig. 3.5). The two primary reasons for doing this are (i) the propagating plane wave should not interact with the ABCs and (ii) the load on the ABCs should be minimized. These boundary conditions are not perfect, that is, a certain portion of the impinging wave is reflected back into the problem space. By subtracting the incident field, the amount of the radiating field hitting the boundary is minimized, thereby reducing the amount of error.

Figure 3.5 illustrates how this is accomplished. Note that there is an auxiliary one-dimensional array called the *incident array*. Because it is a one-dimensional array, it is easy to generate a plane wave: a source point is chosen and the incident

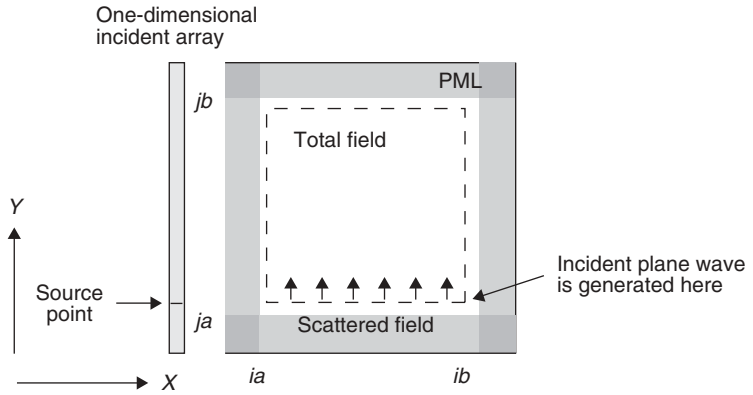

Figure 3.5 Total field/scattered field of the two-dimensional problem space.

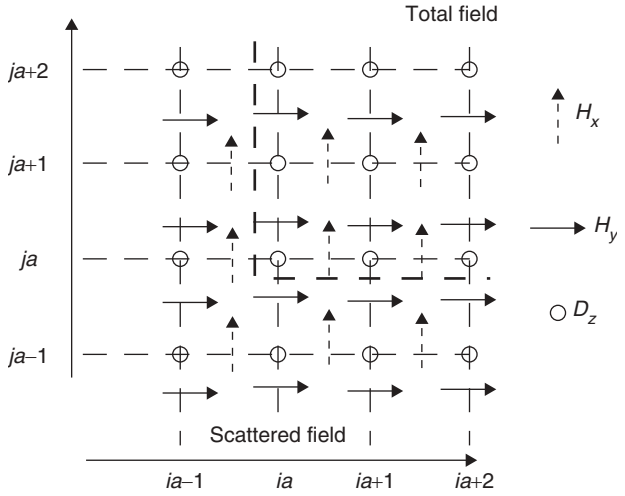

Figure 3.6 Every point is in either the total field or the scattered field.

E_z field is just added at that point. Then a plane wave propagates away in both directions. Again, because it is a one-dimensional array, the boundary conditions are perfect.

As illustrated in Fig. 3.6, in the two-dimensional field, every point in the problem space is either in the total field or it is not; no point lies on the border. Therefore, if a point is in the total field but it uses points outside to calculate the spatial derivatives when updating its value, it must be modified. The same is true of a point lying just outside that uses points inside the total field. This is the reason for the incident array; it contains the needed values to make these modifications.

There are three places that must be modified:

1. The D_z value at $j = j_a$ or $j = j_b$:

$$D_z(i, j_a) = D_z(i, j_a) + 0.5 \cdot H_{x\text{inc}}\left(j_a - \frac{1}{2}\right) \tag{3.26a}$$

$$D_z(i, j_b) = D_z(i, j_b) - 0.5 \cdot H_{x\text{inc}}\left(j_a - \frac{1}{2}\right). \tag{3.26b}$$

2. The H_x field just outside at $j = j_a$ or $j = j_b$:

$$H_x\left(i, j_a - \frac{1}{2}\right) = H_x\left(i, j_a - \frac{1}{2}\right) + 0.5 \cdot E_{z\text{inc}}(j_a) \tag{3.27a}$$

$$H_x\left(i, j_b + \frac{1}{2}\right) = H_x\left(i, j_b + \frac{1}{2}\right) - 0.5 \cdot E_{z\text{inc}}(j_b). \tag{3.27b}$$

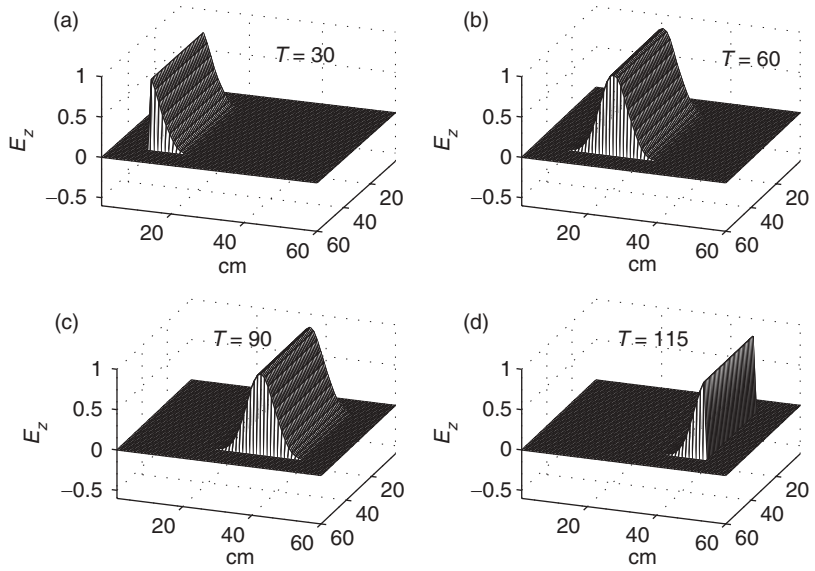

Figure 3.7 Simulation of a plane wave pulse propagating in free space. The incident pulse is generated at one end and subtracted at the other end. (a) $T = 30$, (b) $T = 60$, (c) $T = 90$, and (d) $T = 115$.

3. H_y just outside at $i = i_a$ and $i = i_b$:

$$H_y\left(i_a - \frac{1}{2}, j\right) = H_y\left(i_a - \frac{1}{2}, j\right) - 0.5 \cdot E_{z\text{inc}}(j) \tag{3.28a}$$

$$H_y\left(i_b - \frac{1}{2}, j\right) = H_y\left(i_b - \frac{1}{2}, j\right) + 0.5 \cdot E_{z\text{inc}}(j). \tag{3.28b}$$

Figure 3.7 illustrates the propagation of a Gaussian pulse through the problem space. Notice how the pulse is generated at one end and completely subtracted out at the other end.

3.3.1 A Plane Wave Impinging on a Dielectric Cylinder

Now we have the ability to simulate a plane wave. To simulate a plane wave interacting with an object, we must specify the object according to its electromagnetic properties—the dielectric constant and the conductivity. For instance, suppose we are simulating a plane wave striking a dielectric cylinder 20 cm in diameter, which has a dielectric constant specified by the parameter `epsilon` and a conductivity specified by the parameter `sigma`. The cylinder is specified by the following computer code:

```
for (j=ja; j< jb; j++) }
```

```
      for (i=ia; i<ib; i++) {
           xdist = (ic-i);
           ydist = (jc-j);
           dist = sqrt(pow(xdist,2.) + pow(ydist,2.));
           if ( dist <= radius) {
               ga[i][j] = 1./(epsilon + (sigma*dt/epsz));
               gb[i][j] = sigma*dt/epsz;
 }    }
```

Of course, this assumes that the problem space is initialized to free space. For every cell, the distance to the center of the problem space is calculated, and if it is less than the radius, the dielectric constant and conductivity are reset to `epsilon` and `sigma`, respectively. A diagram of the problem space to simulate a plane wave interacting with a dielectric cylinder is shown in Fig. 3.8.

The weakness of this approach is obvious: because we determined the properties by a simple "in-or-out" approach, we are left with the "staircasing" at the edge of the cylinder. It would be better if we had a way to make a smooth transition from one medium to another. One method is to divide every FDTD cell up into subcells and then determine the average dielectric properties according to the number of subcells in one medium as well as in the other medium. The following code implements such a procedure. The ability to model multiple layers of different material has also been added.

```
for (j=ja; j< jb; j++) {
      for (i=ia; i<ib; i++) {
      eps = epsilon[0];
      cond = sigma[0];
      for (jj=-1; jj< 1; j++) {
      for (ii=-1; ii< 1; j++) {
           xdist = (ic-i) + .333*ii;
           ydist = (jc-j) + .333*jj;
           dist = sqrt(pow(xdist,2.) + pow(ydist,2.));
           if ( dist <= radius) {
           eps = eps + (1./9.)*(epsilon[n] - epsilon[n-1]);
           cond = cond + (1./9.)*(sigma[n] - sigma[n-1]);
    } }
                ga[i][j] = 1./(eps + (sigma*dt/epsz));
                gb[i][j] = cond*dt/epsz;
 }   }
```

Each cell is initialized to the values `epsilon [0]` and `sigma [0]`. Notice that each of the inner loops, which are iterated by the parameters `ii` and `jj`, move the distance one-third of a cell length. Thus, the cell has been divided into nine subcells. Also, notice that as a subcell is determined to be within the radius, its contribution is added to the total epsilon `(epsilon [n])`, while subtracting the previous epsilon `(epsilon[n-1])`. The above code is valid for an arbitrary number of layers in the cylinder.

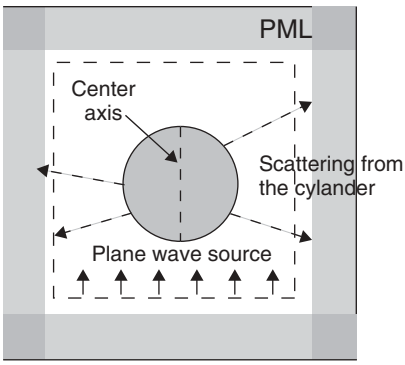

Figure 3.8 Simulation of a plane wave striking a dielectric cylinder. The fields scattered from the cylinder are the only fields to leave the total field and strike the PML.

The simulation of a plane wave pulse hitting a dielectric cylinder with $\varepsilon_r = 30$ and $\sigma = 0.3$ is shown in Fig. 3.9. After 25 time steps, the plane wave has started from the side; after 50 time steps, the pulse is interacting with the cylinder. Some of the pulse passes through the cylinder, and some of it goes around it. After 100

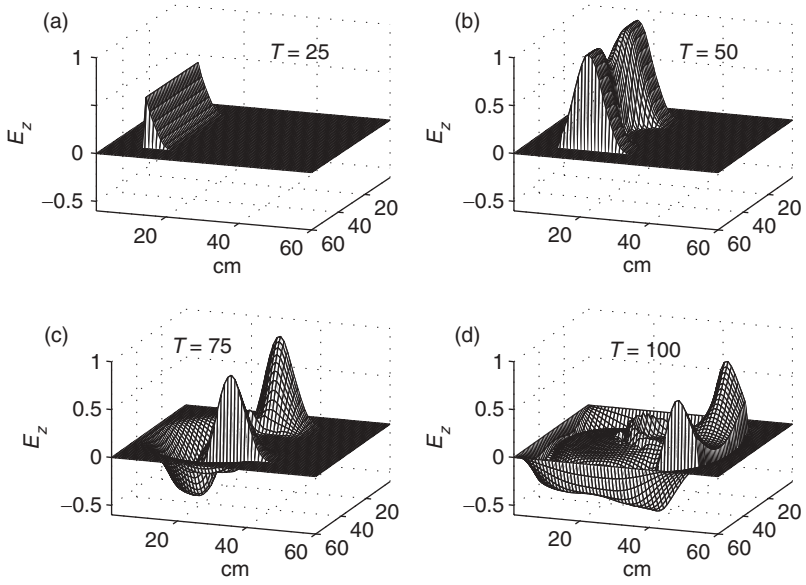

Figure 3.9 Simulation of a plane wave impinging on a dielectric cylinder. The cylinder is 20 cm in diameter and has a relative dielectric constant of 30 and a conductivity of 0.3 S/m. (a) $T = 25$, (b) $T = 50$, (c) $T = 75$, and (d) $T = 100$.

steps, the main part of the propagating pulse is being subtracted from the end of the total field.

3.3.2 Fourier Analysis

Suppose we want to determine how the EM energy is deposited within the cylinder for a plane wave propagating at a specific frequency. Recall that we did something similar in Section 2.4 where we were able to determine the attenuation of the E field at several frequencies by using a pulse for a source and calculating the discrete Fourier transform at the frequencies of interest. This is exactly what we will do here. The only difference is that we have a larger number of points because it is a two-dimensional space. Furthermore, we can use the Fourier transform of the pulse in the one-dimensional incident buffer to calculate the amplitude and phase of the incident pulse.

There is a reason why a dielectric cylinder was used as the object: it has an analytical solution. The fields resulting from a plane wave at a single frequency interacting with a dielectric cylinder can be calculated through a Bessel function expansion (8). This gives us an opportunity to check the accuracy of our

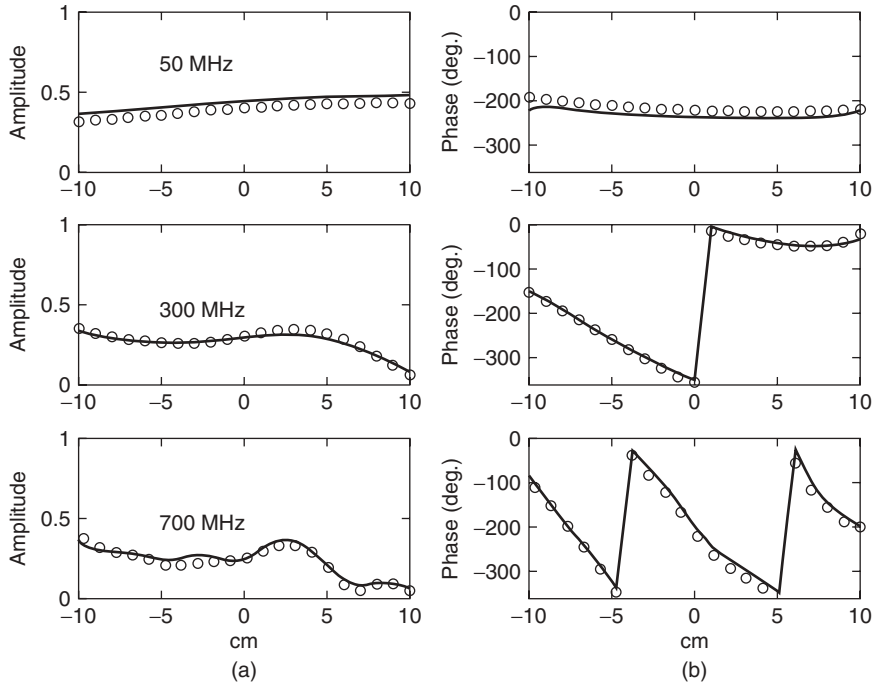

Figure 3.10 Comparison of (a) FDTD results with (b) the Bessel function expansion results along the propagation center axis of a cylinder at three frequencies. The cylinder is 20 cm in diameter and has a relative dielectric constant of 30 and a conductivity of 0.3 S/m.

simulations. Figure 3.10 is a comparison of the FDTD calculations with the analytic solutions from the Bessel function expansions along the center axis of the cylinder in the propagation directions (Fig. 3.8) at 50, 300, and 700 MHz. This was calculated with the program that averaged the values across the boundaries, as described earlier. The accuracy is quite good. Remember, we were able to calculate all three frequencies with one computer run by the impulse response method.

PROBLEM SET 3.3

1. The program fd2d_3.3.c implements the two-dimensional TM FDTD algorithm with an incident plane wave. Note that Eq. (3.26) is implemented to give the correct E_z field at the total/scattered field boundary and Eq. (3.27) is implemented to give the correct H_x field. Why is there no modification of the H_y field?

2. Get the program fd2d_3.3.c running. You should be able to observe the pulse that is generated at `j=ja`, propagates through the problem space, and is subtracted out at `j=jb`.

3. The program fd2d_3.4.c differs from fd2d_3.3.c because it simulates a plane wave hitting a dielectric cylinder. It also calculates the frequency response at three frequencies within the cylinder. Get this running and verify the results in Fig. 3.10.

4. The cylinder generated in fd2d_3.4.c generates the cylinder by the "in-or-out" method. Change this to the averaged values and add the ability to generate layered cylinders, as described in Section 3.3.2.

REFERENCES

1. K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations, in isotropic media, *IEEE Trans. Antenn. Propag.*, vol. AP-17, 1996, pp. 585–589.
2. A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Boston: Artech House, 1995.
3. K. S. Kunz and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*, Boca Raton, FL: CRC Press, 1993.
4. J. P. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, *J. Comput. Phys.*, vol. 114, 1994, pp. 185–200.
5. D. M. Sullivan, A simplified PML for use with the FDTD method, *IEEE Microw. Guid. Wave Lett.*, vol. 6, February 1996, pp. 97–99.
6. Z. S. Sacks, D. M. Kingsland, T. Lee, and J. F. Lee, A perfectly matched anisotropic absorber for use as an absorbing boundary condition, *IEEE Trans. Antenn. Propag.*, vol. 43, December 1995, pp. 1460–1463.

7. D. M. Sullivan, An unsplit step 3D PML for use with the FDTD method, *IEEE Microw. Guid. Wave Lett.*, vol. 7, February 1997, pp. 184–186.
8. R. Harrington, *Time-Harmonic Electromagnetic Fields*, New York: McGraw-Hill, 1961.

```
/* fd2d_3.1.c. 2D TM program */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE  60
#define JE  60

main ()
{
float ga[IE][JE],dz[IE][JE],ez[IE][JE];
float hx[IE][JE],hy[IE][JE];
int l,n,i,j,ic,jc,nsteps;
float ddx,dt,T,epsz,pi,epsilon,sigma,eaf;
float t0,spread,pulse;
 FILE *fp, *fopen();

    ic = IE/2;
    jc = JE/2;
    ddx = .01;                  /* Cell size */
    dt =ddx/6e8;                /* Time steps */
    epsz = 8.8e-12;
    pi=3.14159;

    for ( j=0; j < JE; j++ ) {
       printf( "%2d  ",j);
       for ( i=0; i < IE; i++ ) {
           dz[i][j] = 0.;
           ez[i][j] = 0.;
           hx[i][j] = 0.;
           hy[i][j] = 0.;
           ga[i][j]= 1.0  ;
           printf( "%5.2f ",ga[i][j]);
       }
       printf( " \n");
    }

    t0 = 20.0;
    spread = 6.0;
    T = 0;
    nsteps = 1;

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
```

```
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)  {
       T = T + 1;

/*  ----   Start of the Main FDTD loop ----  */

       /* Calculate the Ex field */
       for ( j=1; j < IE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              dz[i][j] = dz[i][j]
        + .5*( hy[i][j] - hy[i-1][j] - hx[i][j] + hx[i]
 [j-1]);
          }
       }

       /* Put a Gaussian pulse in the middle */

       pulse =  exp(-.5*(pow((t0-T)/spread,2.0)));
       dz[ic][jc] = pulse;

       /* Calculate the Ex field */
       for ( j=1; j < JE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              ez[i][j] = ga[i][j]*dz[i][j] ;
          }
       }

       /* Calculate the Hx field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i < IE-1; i++ ) {
          hx[i][j] = hx[i][j]  + .5*( ez[i][j] - ez[i][j+1] ) ;
       }

       /* Calculate the Hy field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i <= IE-1; i++ ) {
          hy[i][j] = hy[i][j]  + .5*( ez[i+1][j] - ez[i][j] ) ;
          }
       }
    }
/*  ----  End of the main FDTD loop ---- */

       for ( j=1; j < jc; j++ ) {
          printf( "%2d  ",j);
          for ( i=1; i < ic; i++ ) {
             printf( "%5.2f ",ez[2*i][2*j]);
          }
```

```
            printf( " \n");
       }
            printf( "T = %5.0f \n",T);

     /* Write the E field out to a file "Ez" */
       fp = fopen( "Ez","w");
       for ( j=0; j < JE; j++ ) {
          for ( i=0; i < IE; i++ ) {
             fprintf( fp,"%6.3f ",ez[i][j]);
          }
             fprintf( fp," \n");
       }
       fclose(fp);

 }

}

/*  Fd2d_3.2.c. 2D TM program with the PML */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE  60
#define JE  60

main ()
{
    float ga[IE][JE],dz[IE][JE],ez[IE][JE],hx[IE][JE],hy[IE]
[JE];
    int l,n,i,j,ic,jc,nsteps,npml;
    float ddx,dt,T,epsz,pi,epsilon,sigma,eaf;
    float xn,xxn,xnum,xd,curl_e;
    float t0,spread,pulse;
    float gi2[IE],gi3[IE];
    float gj2[JE],gj3[IE];
    float fi1[IE],fi2[IE],fi3[JE];
    float fj1[JE],fj2[JE],fj3[JE];
    float ihx[IE][JE],ihy[IE][JE];
    FILE *fp, *fopen();

    ic = IE/2-10;
    jc = JE/2-10;
    ddx = .01;                  /* Cell size */
    dt =ddx/6e8;                /* Time steps */
    epsz = 8.8e-12;
    pi=3.14159;

      /* Initialize the arrays */
```

```
    for ( j=0; j < JE; j++ ) {
       printf( "%2d  ",j);
       for ( i=0; i < IE; i++ ) {
           dz[i][j]= 0.0  ;
           hx[i][j]= 0.0  ;
           hy[i][j]= 0.0  ;
           ihx[i][j]= 0.0  ;
           ihy[i][j]= 0.0  ;
           ga[i][j]= 1.0  ;
           printf( "%5.2f ",ga[i][j]);
       }
       printf( " \n");
    }

  /* Calculate the PML parameters */

       for (i=0;i< IE; i++) {
          gi2[i] = 1.0;
          gi3[i] = 1.0;
          fi1[i] = 0.0;
          fi2[i] = 1.0;
          fi3[i] = 1.0;
       }
       for (j=0;j< IE; j++) {
          gj2[j] = 1.0;
          gj3[j] = 1.0;
          fj1[j] = 0.0;
          fj2[j] = 1.0;
          fj3[j] = 1.0;
       }

      printf( "Number of PML cells --> ");
       scanf("%d", &npml);

       for (i=0;i<= npml; i++) {
       xnum  = npml - i;
       xd  = npml;
       xxn = xnum/xd;
       xn  = 0.25*pow(xxn,3.0);
       printf(" %d %7.4f  %7.4f \n",i,xxn,xn);
          gi2[i] = 1.0/(1.0+xn);
          gi2[IE-1-i] = 1.0/(1.0+xn);
          gi3[i] = (1.0 - xn)/(1.0 + xn);
          gi3[IE-i-1] = (1.0 - xn)/(1.0 + xn);
       xxn = (xnum-.5)/xd;
       xn  = 0.25*pow(xxn,3.0);
          fi1[i] = xn;
          fi1[IE-2-i] = xn;
          fi2[i] = 1.0/(1.0+xn);
```

```
         fi2[IE-2-i] = 1.0/(1.0+xn);
         fi3[i] = (1.0 - xn)/(1.0 + xn);
         fi3[IE-2-i] = (1.0 - xn)/(1.0 + xn);
      }

 for (j=0;j<= npml; j++) {
      xnum  = npml - j;
      xd  = npml;
      xxn = xnum/xd;
      xn  = 0.25*pow(xxn,3.0);
      printf(" %d %7.4f  %7.4f \n",i,xxn,xn);
         gj2[j] = 1.0/(1.0+xn);
         gj2[JE-1-j] = 1.0/(1.0+xn);
         gj3[j] = (1.0 - xn)/(1.0 + xn);
         gj3[JE-j-1] = (1.0 - xn)/(1.0 + xn);
      xxn = (xnum-.5)/xd;
      xn  = 0.25*pow(xxn,3.0);
         fj1[j] = xn;
         fj1[JE-2-j] = xn;
         fj2[j] = 1.0/(1.0+xn);
         fj2[JE-2-j] = 1.0/(1.0+xn);
         fj3[j] = (1.0 - xn)/(1.0 + xn);
         fj3[JE-2-j] = (1.0 - xn)/(1.0 + xn);
      }

  printf("gi + fi \n");
      for (i=0; i< IE; i++) {
          printf( "%2d   %5.2f  %5.2f \n",
                   i,gi2[i],gi3[i]),
          printf( "     %5.2f  %5.2f  %5.2f \n ",
                      fi1[i],fi2[i],fi3[i]);
      }

   printf("gj + fj \n");
      for (j=0; j< JE; j++) {
          printf( "%2d   %5.2f  %5.2f \n",
                   j,gj2[j],gj3[j]),
          printf( "     %5.2f  %5.2f  %5.2f \n ",
                      fj1[j],fj2[j],fj3[j]);
      }

    t0 = 40.0;
    spread = 12.0;
    T = 0;
    nsteps = 1;
while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);
```

```
    for ( n=1; n <=nsteps ; n++)  {
       T = T + 1;

/*  ----   Start of the Main FDTD loop ----  */

       /* Calculate the Dz field */
       for ( j=1; j < IE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              dz[i][j] = gi3[i]*gj3[j]*dz[i][j]
             + gi2[i]*gj2[j]*.5*( hy[i][j] - hy[i-1][j]
                                  - hx[i][j] + hx[i][j-1]) ;
          }

/* Sinusoidal Source */

      /* pulse =  sin(2*pi*1500*1e6*dt*T);; */
       pulse = exp(-.5*pow( (T-t0)/spread,2.));
       dz[ic][jc] = pulse;

       /* Calculate the Ez field */
       /* Leave the Ez edges to 0, as part of the PML */
       for ( j=1; j < JE-1; j++ ) {
          for ( i=1; i < IE-1; i++ ) {
              ez[i][j] = ga[i][j]*dz[i][j] ;
       }   }

     printf("%3f  %6.2f \n ",T,ez[ic][jc]);

       /* Calculate the Hx field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i < IE; i++ ) {
             curl_e =   ez[i][j] - ez[i][j+1]  ;
             ihx[i][j] = ihx[i][j]  + fi1[i]*curl_e ;
             hx[i][j] = fj3[j]*hx[i][j]
                      + fj2[j]*.5*(curl_e + ihx[i][j]);
        }  }

       /* Calculate the Hy field */
       for ( j=0; j <= JE-1; j++ ) {
          for ( i=0; i < IE-1; i++ ) {
             curl_e  = ez[i+1][j] - ez[i][j];
             ihy[i][j] = ihy[i][j]  + fj1[j]*curl_e ;
             hy[i][j] = fi3[i]*hy[i][j]
                      + fi2[i]*.5*(curl_e + ihy[i][j]);
   }  }    }

/*  ----  End of the main FDTD loop ---- */
      for ( j=1; j < JE; j++ ) {
```

```
          printf( "%2d  ",j);
          for ( i=1; i <= IE; i++ ) {
             printf( "%4.1f",ez[i][j]);
          }
             printf( " \n");
       }

        /* Write the E field out to a file "Ez" */
       fp = fopen( "Ez","w");
       for ( j=0; j < JE; j++ ) {
           for ( i=0; i < IE; i++ ) {
             fprintf( fp,"%6.3f ",ez[i][j]);
          }
             fprintf( fp," \n");
        }

        fclose(fp);

       printf("T =  %6.0f \n ",T);

} }

/* Fd2d_3.3.c. 2D TM program with plane wave source */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE 60
#define JE 60

main ()
{

    float ez_inc[JE],hx_inc[JE];
    float ez_inc_low_m1,ez_inc_low_m2;
    float ez_inc_high_m1,ez_inc_high_m2;
    int ia,ib,ja,jb;

    ic = IE/2;
    jc = JE/2;
    ia = 7;              /* Total/scattered field
 boundaries */
    ib = IE-ia-1;
    ja = 7;
    jb = JE-ja-1;
    ddx = .01;                /* Cell size */
    dt =ddx/6e8;              /* Time step */
    epsz = 8.8e-12;
    pi=3.14159;
```

```
    t0 = 20.0;
    spread = 8.0;
    T = 0;
    nsteps = 1;

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)  {
       T = T + 1;

/*  ----   Start of the Main FDTD loop ----  */

      for (j=1; j< JE; j++) {
         ez_inc[j] = ez_inc[j] + .5*(hx_inc[j-1]-hx_inc[j]);
      }

      /* ABC for the incident buffer */
      ez_inc[0]      = ez_inc_low_m2;
      ez_inc_low_m2  = ez_inc_low_m1;
      ez_inc_low_m1  = ez_inc[1];

      ez_inc[JE-1]    = ez_inc_high_m2;
      ez_inc_high_m2  = ez_inc_high_m1;
      ez_inc_high_m1  = ez_inc[JE-2];

       /* Calculate the Dz field */
       for ( j=1; j < IE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              dz[i][j] = gi3[i]*gj3[j]*dz[i][j]
             + gi2[i]*gj2[j]*.5*( hy[i][j] - hy[i-1][j]
                                  - hx[i][j] + hx[i][j-1]) ;
        }  }

      /*  Source */

       /* pulse =  sin(2*pi*400*1e6*dt*T) ; */
       pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
       ez_inc[3] = pulse;

       /* Incident Dz values */

       for (i=ia; i<= ib; i++ )  {
          dz[i][ja] = dz[i][ja] + 0.5*hx_inc[ja-1];
          dz[i][jb] = dz[i][jb] - 0.5*hx_inc[jb];
       }
```

```
 /* Calculate the Ez field */
       for ( j=1; j < JE-1; j++ ) {
          for ( i=1; i < IE-1; i++ ) {
              ez[i][j] = ga[i][j]*dz[i][j] ;
        }  }

   printf("%3f.0  %6.2f \n ",T,ez[ic-5][jc-5]);

      for (j=0; j< JE; j++) {
         hx_inc[j] = hx_inc[j] + .5*(ez_inc[j]-ez_inc[j+1]);
      }

       /* Calculate the Hx field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i < IE; i++ ) {
             curl_e =   ez[i][j] - ez[i][j+1]  ;
             ihx[i][j] = ihx[i][j]  + fi1[i]*curl_e ;
             hx[i][j] = fj3[j]*hx[i][j]
                      + fj2[j]*.5*(curl_e + ihx[i][j]);
       }   }

    /* Incident Hx values */

       for (i=ia; i<= ib; i++ )  {
          hx[i][ja-1] = hx[i][ja-1] + .5*ez_inc[ja];
          hx[i][jb]   = hx[i][jb] - .5*ez_inc[jb];
       }

       /* Calculate the Hy field */
       for ( j=0; j <= JE-1; j++ ) {
          for ( i=0; i < IE-1; i++ ) {
             curl_e  = ez[i+1][j] - ez[i][j];
             ihy[i][j] = ihy[i][j]  + fj1[j]*curl_e ;
             hy[i][j] = fi3[i]*hy[i][j]

   /* Incident Hy values */
       for (j=ja; j<= jb; j++ )  {
          hy[ia-1][j] = hy[ia-1][j] - .5*ez_inc[j];
          hy[ib][j]   = hy[ib][j]   + .5*ez_inc[j];
       }

     }
/*  ----  End of the main FDTD loop ---- */                +

/*  Fd2d_3.4.c. 2D TM simulation of a plane
 wave source impinging on a dielectric cylinder.
 Analysis using Fourier Transforms */
# include <math.h>
```

```
# include <stdlib.h>
# include <stdio.h>

#define IE 50
#define JE 50
#define NFREQS 3

    float freq[NFREQS],arg[NFREQS];
    float real_pt[NFREQS][IE][JE],imag_pt[NFREQS][IE][JE];
    float amp[IE][JE],phase[IE][JE];
    float real_in[5],imag_in[5],amp_in[5],phase_in[5];

    /* Parameters for the Fourier Transforms */

    freq[0] =  50.e6;
    freq[1] = 300.e6;
    freq[2] = 700.e6;

   for ( n=0; n < NFREQS; n++)
   { arg[n] = 2*pi*freq[n]*dt;
   printf( "%d %6.2f %7.5f \n",n,freq[n]*1e-6,arg[n]);
   }

  /* Specify the dielectric cylinder */

       printf( "Cylinder radius (cells), epsilon, sigma --> ");
       scanf("%f %f %f", &radius, &epsilon, &sigma);
       printf( "Radius = %5.2f  Eps = %6.2f Sigma = %6.2f \n ",
         radius,epsilon,sigma);

       for ( j = ja; j < jb; j++ ) {
          for ( i=ia; i < ib; i++ ) {
              xdist = (ic-i);
              ydist = (jc-j);
              dist = sqrt(pow(xdist,2.) + pow(ydist,2.));
                if( dist <= radius) {
                   ga[i][j] = 1./(epsilon + (sigma*dt/epsz));
                   gb[i][j] = sigma*dt/epsz;
        }   }   }

  /* Write the ga field out to a file "Ga" */
       fp = fopen( "Ga","w");
       printf( " Ga  \n");
       for ( j=ja; j <= jb; j++ ) {
          for ( i=ia; i <= ib; i++ ) {
             printf( "%5.2f",ga[i][j]);
             fprintf( fp,"%6.3f ",ga[i][j]);
          }
             printf( " \n");
```

```
            fprintf( fp," \n");
       }

       printf( " Gb  \n");
       for ( j=ja; j < jb; j++ ) {
          for ( i=ia; i <= ib; i++ ) {
             printf( "%5.2f",gb[i][j]);
          }
             printf( " \n");
       }
        fclose(fp);

/*  ----   Start of the Main FDTD loop ----  */

      /* Calculate the incidnet Ez  */
      for (j=1; j< JE; j++) {
         ez_inc[j] = ez_inc[j] + .5*(hx_inc[j-1]-hx_inc[j]);
      }

   /* Fourier Tramsform of the incident field */
       for ( m=0; m < NFREQS ; m++ )
       { real_in[m] = real_in[m] + cos(arg[m]*T)*ez_inc
 [ja-1] ;
         imag_in[m] = imag_in[m] - sin(arg[m]*T)*ez_inc
 [ja-1] ;
        }

      /* Calculate the Dz field */
       for ( j=1; j < IE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              dz[i][j] = gi3[i]*gj3[j]*dz[i][j]
             + gi2[i]*gj2[j]*.5*( hy[i][j] - hy[i-1][j]
                                  - hx[i][j] + hx[i][j-1]) ;
        }  }

       /*  Source */

       /* pulse =  sin(2*pi*400*1e6*dt*T) ; */
       pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
         ez_inc[3] = pulse;
  /*     dz[ic-5][jc-5] = pulse; */

       printf("%3.0f  %6.2f \n ",T,ez_inc[3]);
       /* Incident Dz values */

       for (i=ia; i<= ib; i++ )  {
          dz[i][ja] = dz[i][ja] + 0.5*hx_inc[ja-1];
          dz[i][jb] = dz[i][jb] - 0.5*hx_inc[jb];
       }
```

```
  /* Calculate the Ez field */
  for ( j=1; j < JE-1; j++ ) {
     for ( i=1; i < IE-1; i++ ) {
         ez[i][j] = ga[i][j]*( dz[i][j] - iz[i][j] ) ;
         iz[i][j] = iz[i][j] + gb[i][j]*ez[i][j] ;
     }
  }

       /* Calculate the Fourier transform of Ex. */
for ( j=0; j < JE; j++ )
{   for ( i=0; i < JE; i++ )
   {   for ( m=0; m < NFREQS; m++ )
{real_pt[m][i][j] = real_pt[m][i][j] +
                  cos(arg[m]*T)*ez[i][j] ;
    imag_pt[m][i][j] = imag_pt[m][i][j] +
                       sin(arg[m]*T)*ez[i][j] ;
    } } }

 /* Calculate the incident Hx */
 for (j=0; j< JE; j++) {
    hx_inc[j] = hx_inc[j] + .5*(ez_inc[j]-ez_inc[j+1]);
 }

  /* Calculate the Hx field */
  for ( j=0; j < JE-1; j++ ) {
     for ( i=0; i < IE; i++ ) {
        curl_e =   ez[i][j] - ez[i][j+1]  ;
        ihx[i][j] = ihx[i][j]  + fi1[i]*curl_e ;
        hx[i][j] = fj3[j]*hx[i][j]
                 + fj2[j]*.5*(curl_e + ihx[i][j]);
     }
  }
  /* Incident Hx values */

  for (i=ia; i<= ib; i++ )  {
     hx[i][ja-1] = hx[i][ja-1] + .5*ez_inc[ja];
     hx[i][jb]   = hx[i][jb] - .5*ez_inc[jb];
  }

  /* Calculate the Hy field */
  for ( j=0; j <= JE-1; j++ ) {
     for ( i=0; i < IE-1; i++ ) {
        curl_e  = ez[i+1][j] - ez[i][j];
        ihy[i][j] = ihy[i][j]  + fj1[j]*curl_e ;
        hy[i][j] = fi3[i]*hy[i][j]
                 + fi2[i]*.5*(curl_e + ihy[i][j]);
     }
  }
```

```
       /* Incident Hy values */
       for (j=ja; j<= jb; j++ )  {
          hy[ia-1][j] = hy[ia-1][j] - .5*ez_inc[j];
          hy[ib][j]   = hy[ib][j]   + .5*ez_inc[j];
       }

    }
/*  ----  End of the main FDTD loop ---- */

  /* Calculate the Fouier amplitude and phase of the
 incident pulse */
       for ( m=0; m < NFREQS; m++ )
       {    amp_in[m]   = sqrt( pow(real_in[m],2.)
+ pow(imag_in[m],2.));
            phase_in[m] = atan2(imag_in[m],real_in[m]) ;
             printf( "%d  Input Pulse :  %8.4f %8.4f %8.4f
%7.2f\n",
             m,real_in[m],imag_in[m],amp_in[m],
(180.0/pi)*phase_in[m]);
       }

   /* Calculate the Fouier amplitude and phase of the
 total field field */
       for ( m=0; m < NFREQS; m++ )
       {
       if( m == 0)       fp = fopen( "amp1","w");
       else if( m == 1)  fp = fopen( "amp2","w");
       else if( m == 2)  fp = fopen( "amp3","w");
       if( m == 0)       fp1 = fopen( "phs1","w");
       else if( m == 1)  fp1 = fopen( "phs2","w");
       else if( m == 2)  fp1 = fopen( "phs3","w");
       {   printf( "%2d  %7.2f  MHz\n",m,freq[m]*1.e-6);
           for ( j=ja; j <= jb; j++ )
           {  if( ga[ic][j] < 1.00 )
              {  amp[ic][j]  = (1./amp_in[m])
                 *sqrt( pow(real_pt[m][ic][j],2.)
+ pow(imag_pt[m][ic][j],2.));
            phase[ic][j] = atan2(imag_pt[m][ic][j],real_pt[m]
[ic][j]) ;
                printf( "%2d %9.4f \n",jc-j,amp[ic][j]);
                fprintf( fp," %9.4f \n",amp[ic][j]);
                fprintf( fp1," %9.4f \n",phase[ic][j]);
              }
           }
        }
        fclose(fp);
        fclose(fp1);
        }
```

4

THREE-DIMENSIONAL SIMULATION

At last we have come to three-dimensional simulation. In actuality, three-dimensional FDTD simulation is very much like two-dimensional simulation—only harder. It is harder because of logistical problems: we use all vector fields and each one is in three dimensions. Nonetheless, paying attention and building the programs carefully, the process is straightforward.

4.1 FREE SPACE SIMULATION

The original FDTD paradigm was described by the *Yee cell* (Fig. 4.1), named after Kane Yee (1). Note that the E and H fields are assumed interleaved around a cell whose origin is at the locations i, j, and k. Every E field is located half a cell width from the origin in the direction of its orientation; every H field is offset half a cell in each direction except that of its orientation.

Not surprisingly, we will start with Maxwell's equations:

$$\frac{\partial \widetilde{\boldsymbol{D}}}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \widetilde{\boldsymbol{H}}, \tag{4.1a}$$

$$\widetilde{\boldsymbol{D}}(\omega) = \varepsilon_{\mathrm{r}}^{*}(\omega) \cdot \widetilde{\boldsymbol{E}}(\omega), \tag{4.1b}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \widetilde{\boldsymbol{E}}. \tag{4.1c}$$

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

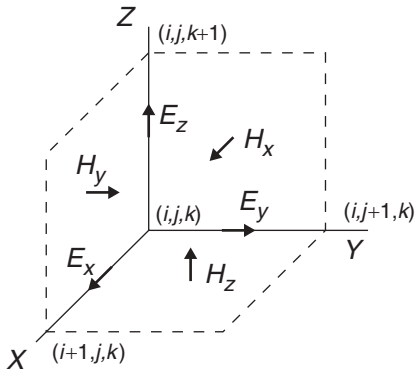

Figure 4.1 The Yee cell.

Once again, we will drop the notation "$\sim$", but it will always be assumed that we are referring to the normalized values.

Equation (4.1a) and (4.1c) produces six scalar equations:

$$\frac{\partial D_x}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \right), \tag{4.2a}$$

$$\frac{\partial D_y}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \right), \tag{4.2b}$$

$$\frac{\partial D_z}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right), \tag{4.2c}$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} \right), \tag{4.2d}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} \right), \tag{4.2e}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \left(\frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \right). \tag{4.2f}$$

The first step is to take the finite-difference approximations. We will use only Eq. (4.2c) and (4.2f) as examples:

$$D_z^{n+1/2}\left(i, j, k+\frac{1}{2}\right) = D_z^{n+1/2}\left(i, j, k+\frac{1}{2}\right) + \frac{\Delta t}{\Delta x \cdot \sqrt{\varepsilon_0 \mu_0}} \left[H_y^n \left(i+\frac{1}{2}, j, k+\frac{1}{2} \right) \right.$$

$$- H_y^n\left(i-\frac{1}{2}, j, k+\frac{1}{2}\right)$$
$$- H_x^{n+1}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right)+H_x^{n+1}\left(i+\frac{1}{2}, j-\frac{1}{2}, k\right)\Bigg] \tag{4.3a}$$

$$H_z^{n+1}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) = H_z^{n+1}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) + \frac{\Delta t}{\Delta x \cdot \sqrt{\varepsilon_0 \mu_0}}\Bigg[E_x^{n+1/2}\left(i+\frac{1}{2}, j+1, k\right) + E_x^{n+1/2}\left(i+\frac{1}{2}, j, k\right) - E_y^{n+1/2}\left(i+1, j+\frac{1}{2}, k\right)+E_y^{n+1/2}\left(i, j+\frac{1}{2}, k\right)\Bigg]. \tag{4.3b}$$

From the difference equations, the computer equations can be written:

```
dx[i][j][k] = dx[i][j][k]
              + .5*( hz[i][j][k] - hz[i][j-1][k]
              - hy[i][j][k] + hy[i][j][k-1] );

dy[i][j][k] = dz[i][j][k]
              + .5*( hx[i][j][k] - hx[i][j][k-1]
              - hz[i][j][k] + hz[i-1][j][k] );

dz[i][j][k] = dz[i][j][k]
              + .5*( hy[i][j][k] - hy[i-1][j][k]
              - hx[i][j][k] + hx[i][j-1][k] );

hx[i][j][k] = hx[i][j][k]
              + .5*( ey[i][j][k+1] - ey[i][j][k]
              - ez[i][j+1][k] + ez[i][j][k] );

hy[i][j][k] = hy[i][j][k]
              + .5*( ez[i+1][j][k] - ez[i][j][k]
              - ex[i][j][k+1] + ex[i][j][k] );

hz[i][j][k] = hz[i][j][k]
              + .5*( ex[i][j+1][k] - ex[i][j][k]
              - ey[i+1][j][k] + ey[i][j][k] );
```

The relationship between E and D, corresponding to Eq. (4.1b), is exactly the same as the one- or two-dimensional cases, except there will be three dimensions.

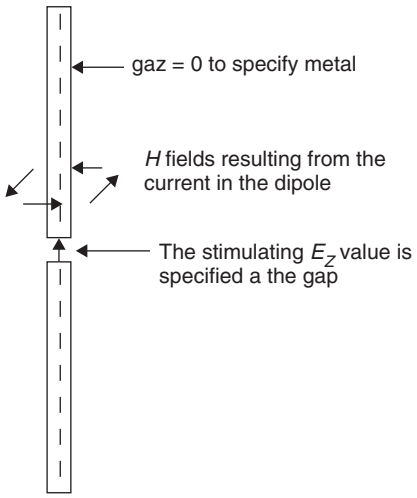

Figure 4.2 A dipole antenna. The FDTD program specifies the metal arms of the dipole by setting gaz = 0. The source is specified by setting the E_z field to a value in the gap.

The program fd3d_4.1.c at the end of this chapter is a very basic three-dimensional FDTD program with a source in the middle of the problem space. This is similar to the two-dimensional program fd2d_3.1.c, except that the source is not a simple point source. In three dimensions, the E field attenuates as the square of the distance as it propagates out from the point source, so we would have trouble just seeing it. Instead, we use a dipole antenna as the source. A simple dipole antenna is illustrated in Fig. 4.2, and it consists of two metal arms. A dipole antenna functions by having current run through the arms, which results in radiation. FDTD simulates a dipole in the following manner: the metal of the arms is specified by setting the `gaz` parameters to zero in the cells corresponding to metal (Eq. (2.9) and Eq. (2.10)). This insures that the corresponding E_z field at this point remains zero, as it would if that point were inside metal. The source is specified by setting the E_z field in the gap to a certain value. In fd3d_4.1.c, we specify a Gaussian pulse. (In a real dipole antenna, the E_z field in the gap would be the result of the current running through the metal arms.) Notice that we could have specified a current in the following manner: Ampere's circuital law says (2)

$$\int_C \boldsymbol{H} \cdot \mathrm{d}l = \boldsymbol{I},$$

that is, the current through a surface is equal to the line integral of the H field around the surface. We could specify the current by setting the appropriate H fields around the gap (Fig. 4.2). We can see that specifying the E field in the gap is easier.

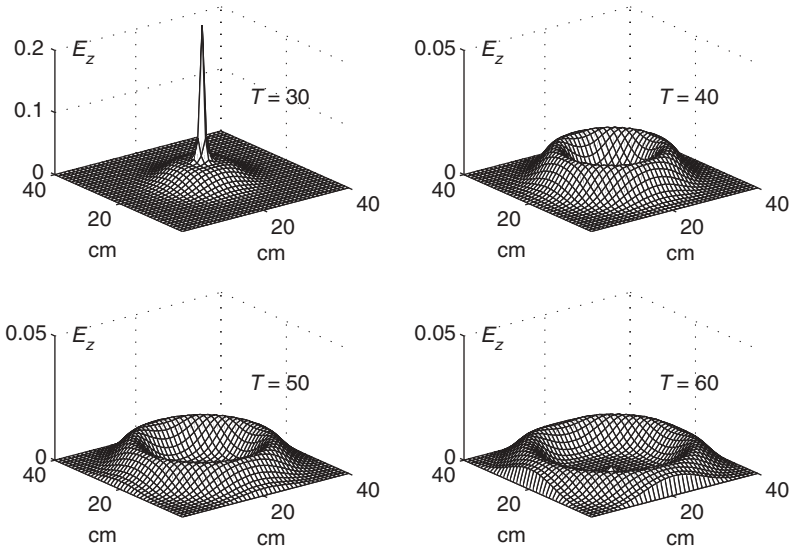

Figure 4.3 E_z field radiation from a dipole antenna in a three-dimensional FDTD program.

Figure 4.3 shows the propagation of the E_z from the dipole in the XY plane level with the gap of the dipole. Of course, there is radiation in the Z direction as well. This illustrates a major problem in three-dimensional simulations: unless one has unusually good graphics, visualizing three dimensions can be difficult.

PROBLEM SET 4.1

1. Get the program fd3d_4.1.c running. Duplicate the results of Fig. 4.3. (Remember, there is no PML yet.)

4.2 THE PML IN THREE DIMENSIONS

The development of the PML for three dimensions closely follows the two-dimensional version. The only difference is that we deal with three dimensions instead of two (3). For instance, Eq. (3.23a) becomes

$$j\omega\left[1+\frac{\sigma_x(x)}{j\omega\varepsilon_0}\right]\left[1+\frac{\sigma_y(y)}{j\omega\varepsilon_0}\right]\left[1+\frac{\sigma_z(z)}{j\omega\varepsilon_0}\right]^{-1} D_z = c_0\cdot\left(\frac{\partial H_y}{\partial x}-\frac{\partial H_x}{\partial y}\right). \tag{4.4}$$

Implementing it will closely follow the two-dimensional development. Start by rewriting Eq. (4.4) as

$$j\omega\left[1+\frac{\sigma_x(x)}{j\omega\varepsilon_0}\right]\left[1+\frac{\sigma_y(y)}{j\omega\varepsilon_0}\right]D_z = c_0\left[1+\frac{\sigma_z(z)}{j\omega\varepsilon_0}\right]\cdot\left(\frac{\partial H_y}{\partial x}-\frac{\partial H_x}{\partial y}\right)$$
$$= c_0\cdot \text{curl\_}h + c_0\frac{\sigma_z(z)}{\varepsilon_0}\frac{1}{j\omega}\text{curl\_}h. \quad (4.5)$$

We will define

$$I_{D_z} = \frac{1}{j\omega}\text{curl\_}h,$$

which is an integration when it goes to the time domain, so Eq. (4.5) becomes

$$j\omega\left[1+\frac{\sigma_x(x)}{j\omega\varepsilon_0}\right]\left[1+\frac{\sigma_y(y)}{j\omega\varepsilon_0}\right]D_z = c_0\left[1+\frac{\sigma_z(z)}{j\omega\varepsilon_0}\right]\cdot\left(\frac{\partial H_y}{\partial x}-\frac{\partial H_x}{\partial y}\right)$$
$$= c_0\cdot\left[\text{curl\_}h+\frac{\sigma_z(z)}{\varepsilon_0}I_{D_z}\right].$$

The implementation of this into FDTD parallels that of the two-dimensional PML, except the right side contains the integration term I_{D_z}. Therefore, following the same math we used in Chapter 3, we get

$$\text{curl\_}h = H_y^n\left(i+\frac{1}{2},j,k+\frac{1}{2}\right)-H_y^n\left(i-\frac{1}{2},j,k+\frac{1}{2}\right)$$
$$-H_x^{n+1}\left(i+\frac{1}{2},j+\frac{1}{2},k\right)+H_x^{n+1}\left(i+\frac{1}{2},j-\frac{1}{2},k\right), \quad (4.6a)$$

$$I_{D_z}^n\left(i,j,k+\frac{1}{2}\right) = I_{D_z}^{n-1}\left(i,j,k+\frac{1}{2}\right)+\text{curl\_}h, \quad (4.6b)$$

$$D_z^{n+1/2}\left(i,j,k+\frac{1}{2}\right) = gi3(i)\cdot gj3(j)\cdot D_z^{n+1/2}\left(i,j,k+\frac{1}{2}\right)$$
$$+gi2(i)\cdot gj2(j)$$
$$\cdot 0.5\cdot\left[\text{curl\_}h+gk1(k)\cdot I_z^{n+1/2}\left(i,j,k+\frac{1}{2}\right)\right]. \quad (4.6c)$$

The one-dimensional g parameters are defined the same as in Eq. (3.25). The computer code for Eq. (4.6) is as follows:

```
for ( i=1; i < IE; i++ ) {
   for ( j=1; j < JE; j++ ) {
       for ( k=0; k < KE; k++ ) {
          curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                   - hx[i][j][k] + hx[i][j-1][k]) ;
          idz[i][j][k] = idz[i][j][k] + curl_h;
```

```
        dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
        + gi2[i]*gj2[j]*.5*( curl_h + gk1[k]*idz[i][j][k] );
    }  }  }
```

The PML described in this section is a variation of the original by Berenger (3). The PML was modified to fit the FDTD formulation utilizing a separate calculation of E from D (4). This version of the PML has proven to be very effective when the background medium is air, as well as when other media, such as human tissues, are present and the PML are present in the other media (5). The reader should be aware that a large number of specialized PMLs have been developed. Sometimes a specialized PML is necessary when used with a particular medium or application (6). One of the most popular approaches used in the development of a PML is the use of stretched coordinates (7). A summary of the diverse approaches to the PML is given in Reference 8.

PROBLEM SET 4.2

1. Add the PML to your three-dimensional program. The program fd3d_4.2.c at the end of the chapter has the PML. The program also has the things described in the next section, but you should be able to pick out those things having to do with the PML. Duplicate the results of Fig. 4.4.

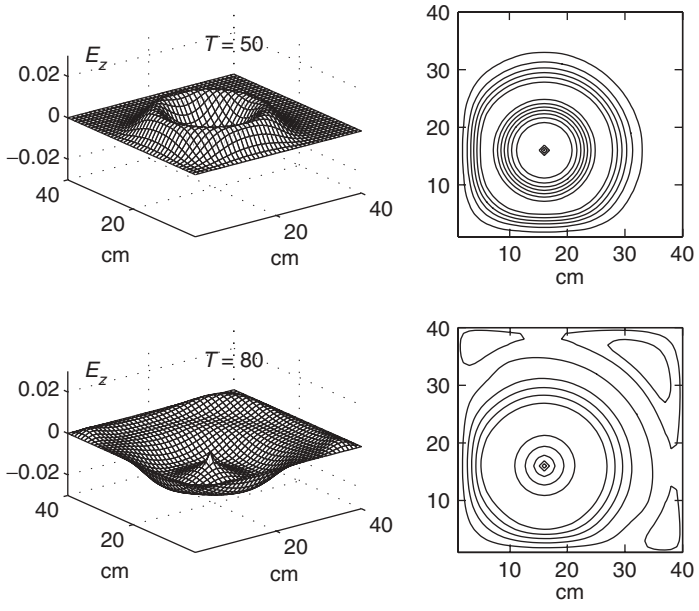

Figure 4.4 Radiation from a dipole antenna in an FDTD program with a seven-point PML. Contour diagrams on the right illustrate the fact that the fields remain concentric until they reach the PML.

4.3 TOTAL/SCATTERED FIELD FORMULATION IN THREE DIMENSIONS

Generating plane waves in three dimensions is similar to that in two dimensions. The three-dimensional problem is illustrated in Fig. 4.5. A plane wave is generated in one plane of the three-dimensional problem space, in this case an *XZ* plane, at $j = ja$ and subtracted out at $j = jb$. Therefore, in free space with no obstacle in the total field, we should see only E_z and H_x. The plane wave is generated at one side and subtracted from the other side by adding to D or H fields that are on the boundary and subtracting from D or H fields that are next to the boundary. Therefore, Eq. (3.26), Eq. (3.27), and Eq. (3.28) are still used, but they are imposed on an entire plane instead of simply a line as in two dimensions. Another difference in three dimensions is the additional surfaces at $k = ka$ and $k = kb$. Figure 4.6 illustrates the $k = ka$ boundary. The calculation of $D_y(i, j, ka)$, which is in the scattered field, requires the values of $H_x(i, j, ka)$, which is in the total field. The difference between the two is the incident component of the H_x field. Therefore, beside equations similar to Eq. (3.26), Eq. (3.27), and Eq. (3.28), the following are necessary:

$$D_y\left(i, j + \frac{1}{2}, ka\right) = D_y\left(i, j + \frac{1}{2}, ka\right) - 0.5 \cdot H_{x\_\text{inc}}(j) \tag{4.7a}$$

$$D_y\left(i, j + \frac{1}{2}, kb + 1\right) = D_y\left(i, j + \frac{1}{2}, kb + 1\right) - 0.5 \cdot H_{x\_\text{inc}}(j). \tag{4.7b}$$

4.3.1 A Plane Wave Impinging on a Dielectric Sphere

Now that we have a program that generates a plane wave in three dimensions, we will want to start putting objects in the problem space to see how the plane wave

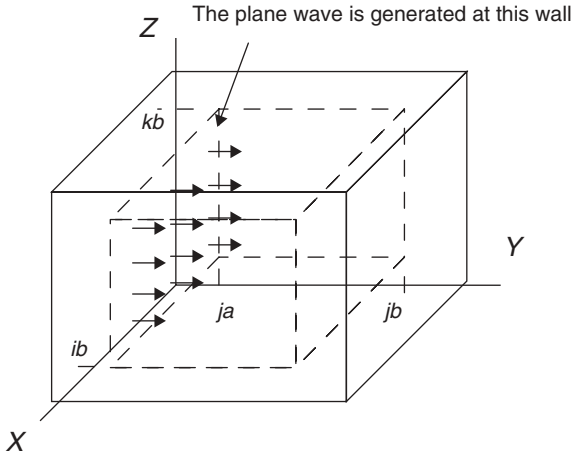

Figure 4.5 Total/scattered fields in three dimensions.

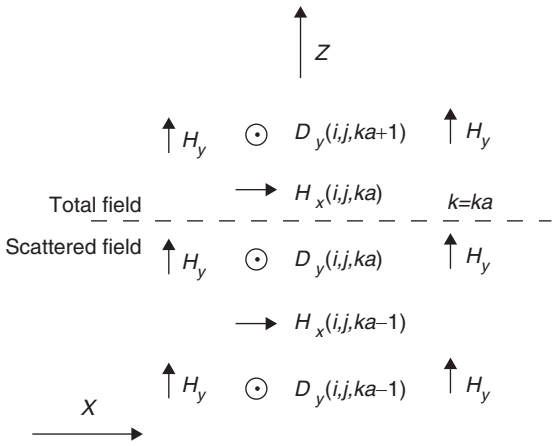

Figure 4.6 Total/scattered field boundary at $k = ka$.

interacts with them. In two dimensions, we chose a cylinder because we had an analytic solution with which we could check the accuracy of our calculation via a Bessel function expansion. It turns out that the interaction of a plane wave with a dielectric sphere can be determined by an expansion of the modified Bessel functions (9).

Specifying a sphere in three dimensions is similar to specifying a cylinder in two dimensions. The major difference is that it must be done for all three electric fields. The following code specifies the parameters needed for the E_z field calculation:

```
for ( i=ia; i < ib; i++) {
   for ( j=ja; j<jb; j++ ) {
      for ( k=ka; k<kb; k++ ) {
           xdist = (ic-i);
           ydist = (jc-j);
           zdist = (kc-k-.5);
           dist = sqrt(pow(xdis,2.)+pow(ydist,2.)+pow(zdist,2.))
           if( dist<= radius) {
              gaz[i][j][k] = 1./(epsilon + (sigma*dt/epsz));
              gbz[i][j][k] = sigma*dt/epsz;
            }
}  }  }
```

Besides the obvious differences, for example, three loops instead of two, there is another difference to be pointed out. The parameter `zdist` calculates the distance to the point as if it were half a cell further in the z direction. That is because it is! As shown in Fig. 4.1, each E field is assumed offset from i, j, and k by half a cell in its own direction.

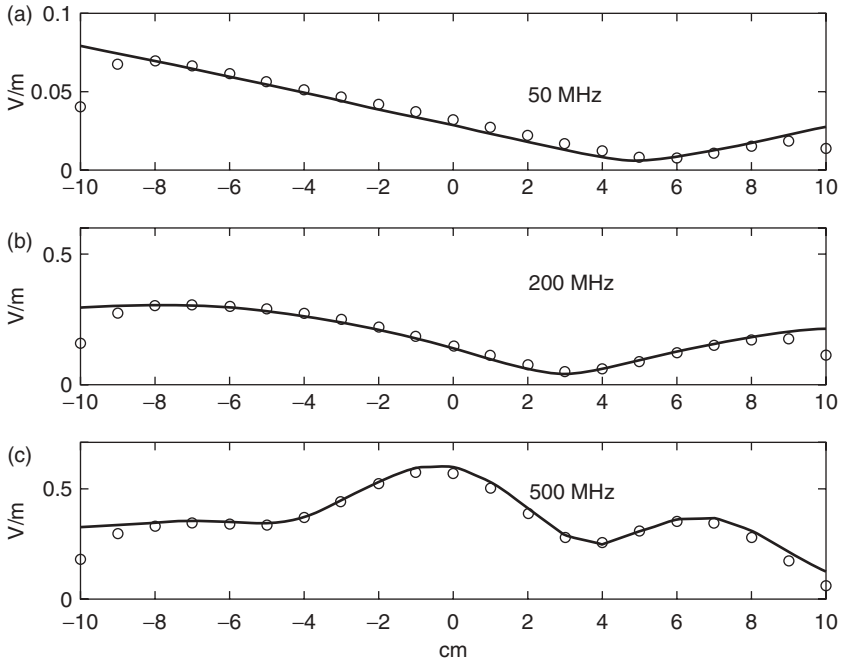

Figure 4.7 Comparison of the FDTD calculation (circles) with the Bessel function expansion calculation (lines) along the main axis of a dielectric sphere, 20 cm in diameter, with $\varepsilon_r = 30$ and $\sigma = 0.3$. The program for the FDTD calculation uses the simple "in-or-out" strategy to determine the parameters. (a) 50, (b) 200, and (c) 500 MHz.

In Fig. 4.7, we show a comparison between FDTD results and the Bessel function expansion results for a plane wave incident on a dielectric sphere. The sphere had a diameter of 20 cm, a dielectric constant of 30, and a conductivity of 0.3. The FDTD program is $50 \times 50 \times 50$ cells and uses a seven-point PML. Apparently, the "in-and-out method" of determining the parameters such as `gaz` and `gbz` is not as forgiving as it was in two dimensions.

We can do an averaging over subcells to improve efficiency. We might conclude that because this means averaging subcells in a plane in two dimensions, we would have to average subcubes in three dimensions. It turns out that this is not the case. Figure 4.8 illustrates the calculation of E_z, which uses the surrounding H_x and H_y values. As this calculation is confined to a plane, we may think of the calculation of E_z as being a two-dimensional problem. Therefore, in determining the parameters `gaz` and `gbz`, we will go by how much area of the plane containing H_x and H_y is in each of the different media. The following code averages the properties over nine subcells:

```
for ( k=ka; k<kb; k++ ) {
  for ( i=ia; i < ib; i++) {
```

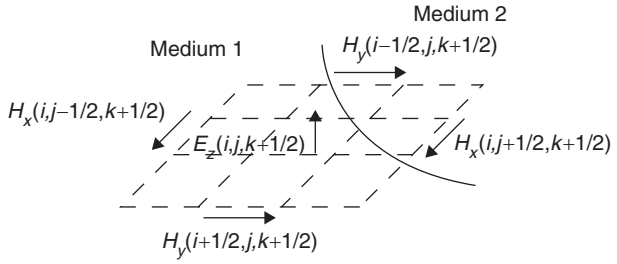

Figure 4.8 E_z is calculated by the surrounding H_x and H_y values. The parameters used to calculate E_z are determined by the percentage of subcells in each medium. In this example, six subcells are in medium 1 and three subcells are in medium 2.

```
     for ( j=ja; j<jb; j++ ) {
     eps = epsilon(0);
     cond = sigma(0);
          for ( jj=-1; jj<=1:jj++ ) {
          for ( ii=-1; ii <=1; ii++) {
           xdist = (ic-i)+.333*ii;
           ydist = (jc-j)+.333*jj;
           zdist = (kc-k-.5);
           dist = sqrt(pow(xdis,2.)+pow(ydist,2.)+pow(zdist,2.))
             for (n=1; n <+ numcyl; n++1) {
                if( dist <= radius) {
                   eps = eps + (1/9)*(epsilon[n] - epsilon[n-1]);
                   cond = cond + (1/9)*(sigma[n] - sigma[n-1]);
              }  }
          }}
              gaz[i][j][k] = 1./(epsilon + (sigma*dt/epsz));
              gbz[i][j][k] = sigma*dt/epsz;
}  }  }
```

Figure 4.9 repeats the FDTD/Bessel comparison, but with the averaged parameter values. Clearly, it results in an improvement.

PROBLEM SET 4.3

1. Add the plane wave propagation to your three-dimensional FDTD program. Make sure that it can propagate a wave through and subtract it out the other end. (See the program fd3d_4.2.c.)

2. The program fd3d_4.2.c creates the spheres by "in-or-out" method. Get this program running and duplicate the results of Fig. 4.7.

3. Use the averaging technique to get a more accurate calculation of the parameters and duplicate the results of Fig. 4.9.

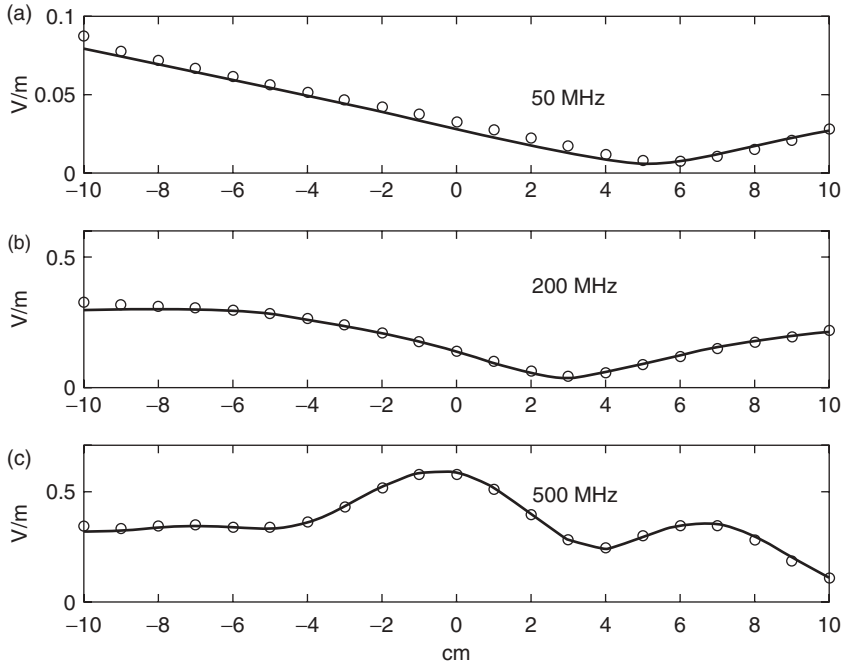

Figure 4.9 Comparison of the FDTD calculation (circles) with the Bessel function expansion calculation (lines) along the main axis of a dielectric sphere, 20 cm in diameter, with $\varepsilon_r = 30$ and $\sigma = 0.3$. The program used for the FDTD calculation averaged over nine subcells within each cell to determine the parameters. (a) 50, (b) 200, and (c) 500 MHz.

REFERENCES

1. K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antenn. Propag.*, vol. AP-17, 1996, pp. 585–589.
2. D. K. Cheng, *Field and Wave Electromagnetics*, Menlo Park, CA: Addison-Wesley, 1992.
3. J. P. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, *J. Comput. Phys.*, vol. 114, 1994, pp. 185–200.
4. D. M. Sullivan, An unsplit step 3D PML for use with the FDTD Method, *IEEE Microw. Guid. Wave Lett.*, vol. 7, February. 1997, pp. 184–186.
5. D. M. Sullivan, P. Wust, and J. Nadobny, Accurate FDTD simulation of RF coils for MRI using the thin-rod approximation, *IEEE Trans. Antenn. Propag.*, vol. 58, August 2003, pp. 1780–1796.
6. D. M. Sullivan, Y. Xia, and D. Butherus, A perfectly matched layer for lossy media at extremely low frequencies, *IEEE Antenn. Wireless Propag. Lett.*, vol. 8, 2009, pp. 1080–1083.

7. W. C. Chew and W. H. Weedon, A 3D perfectly matched medium form modified Maxwell's equations with stretched coordinates, *Microw. Opt. Tech. Lett.*, vol. 3, 1998, pp. 559–605.
8. J-P Berenger, *Perfectly Matched Layer (PML) for Computational Electromagnetics*, San Francisco: Morgan & Claypool Publishers, 2007.
9. R. Harrington, *Time-Harmonic Electromagnetic Fields*, New York: McGraw-Hill, 1961.

```
/* Fd3d_4.1.c. 3D FDTD. Dipole in free space. */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE  50
#define JE  50
#define KE  50

main ()
{
    float gax[IE][JE][KE],gay[IE][JE][KE],gaz[IE][JE][KE];
    float dx[IE][JE][KE],dy[IE][JE][KE],dz[IE][JE][KE];
    float ex[IE][JE][KE],ey[IE][JE][KE],ez[IE][JE][KE];
    float hx[IE][JE][KE],hy[IE][JE][KE],hz[IE][JE][KE];
    int l,n,i,j,k,ic,jc,kc,nsteps,npml;
    float ddx,dt,T,epsz,pi,epsilon,sigma,eaf;
    float xn,xxn,xnum,xd,curl_e;
    float t0,spread,pulse;
    FILE *fp, *fopen();
    float Eav;

    ic = IE/2;
    jc = JE/2;
    kc = KE/2;
    ddx = .01;                  /* Cell size */
    dt =ddx/6e8;                /* Time steps */
    epsz = 8.8e-12;
    pi=3.14159;

    /* Initialize the arrays */
    for ( k=0; k < KE; k++ ) {
       for ( j=0; j < JE; j++ ) {
           for ( i=0; i < IE; i++ ) {
           ex[i][j][k]= 0.0 ;
           ey[i][j][k]= 0.0 ;
           ez[i][j][k]= 0.0 ;
           dx[i][j][k]= 0.0 ;
           dy[i][j][k]= 0.0 ;
           dz[i][j][k]= 0.0 ;
```

```
          hx[i][j][k]= 0.0 ;
          hy[i][j][k]= 0.0 ;
          hz[i][j][k]= 0.0 ;
          gax[i][j][k]= 1.0 ;
          gay[i][j][k]= 1.0 ;
          gaz[i][j][k]= 1.0 ;
   }  }   }

   /* Specify the dipole */
      for ( k=kc-10; k < kc+10; k++ ) {
         gaz[ic][jc][k] = 0.;
      }
      gaz[ic][jc][kc] = 1.0;

   t0 = 20.0;
   spread = 6.0;
   T = 0;
   nsteps = 1;

/*  ----   Start of the Main FDTD loop ----  */

      /* Calculate the Dx field */
            for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
      for ( k=1; k < KE; k++ ) {
                dx[i][j][k] = dx[i][j][k]
               + .5*( hz[i][j][k] - hz[i][j-1][k]
                    - hy[i][j][k] + hy[i][j][k-1]) ;
      }  }  }

      /* Calculate the Dy field */
            for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
      for ( k=1; k < KE; k++ ) {
                dy[i][j][k] = dy[i][j][k]
               + .5*( hx[i][j][k] - hx[i][j][k-1]
                    - hz[i][j][k] + hz[i-1][j][k]) ;
      }  }  }

      /* Calculate the Dz field */
            for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
      for ( k=1; k < KE; k++ ) {
                dz[i][j][k] = dz[i][j][k]
               + .5*( hy[i][j][k] - hy[i-1][j][k]
                    - hx[i][j][k] + hx[i][j-1][k]) ;
      }  }  }
```

```
    /* Add the source at the gap */
       pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
       dz[ic][jc][kc] = pulse;

       /* Calculate the E from D field */
       for ( i=1; i < IE-1; i++ ) {
          for ( j=1; j < JE-1; j++ ) {
             for ( k=1; k < KE-1; k++ ) {
             ex[i][j][k] = gax[i][j][k]*dx[i][j][k];
             ey[i][j][k] = gay[i][j][k]*dy[i][j][k];
             ez[i][j][k] = gaz[i][j][k]*dz[i][j][k];
       }  }  }

       printf("%4.0f  %6.2f \n ",T,ez[ic][jc][kc]);

        /* Calculate the Hx field */
             for ( i=1; i < IE; i++ ) {
          for ( j=1; j < JE-1; j++ ) {
       for ( k=1; k < KE-1; k++ ) {
                hx[i][j][k] = hx[i][j][k]
               + .5*( ey[i][j][k+1] - ey[i][j][k]
                    - ez[i][j+1][k] + ez[i][j][k]) ;
       }  }  }

       /* Calculate the Hy field */
             for ( i=1; i < IE-1; i++ ) {
          for ( j=1; j < JE; j++ ) {
       for ( k=1; k < KE-1; k++ ) {
                hy[i][j][k] = hy[i][j][k]
               + .5*( ez[i+1][j][k] - ez[i][j][k]
                    - ex[i][j][k+1] + ex[i][j][k]) ;
       }  }  }

       /* Calculate the Hz field */
             for ( i=1; i < IE-1; i++ ) {
          for ( j=1; j < JE-1; j++ ) {
       for ( k=1; k < KE; k++ ) {
                hz[i][j][k] = hz[i][j][k]
               + .5*( ex[i][j+1][k]   - ex[i][j][k]
                    - ey[i+1][j][k] + ey[i][j][k]) ;
       }  }  }

    }
/*  ----  End of the main FDTD loop ---- */

       /* Write the Hy field out to a file "Hyk" */
       fp = fopen( "Hyk","w");
       for ( k=0; k < KE; k++ ) {
```

```
          for ( i=ic-10; i < ic+10; i++ ) {
             fprintf( fp,"%7.4f ",hy[i][jc][k]);
             fprintf( fp,"%7.4f ",hy[i][jc][k]);
          }
             fprintf( fp," \n");
             fprintf( fp," \n");
       }
        fclose(fp);

        /* Write the E field out to a file "Ez" */
       fp = fopen( "Ez","w");
       for ( j=0; j < JE; j++ ) {
           for ( i=0; i < IE; i++ ) {
             fprintf( fp,"%7.4f ",ez[i][j][kc]);
          }
             fprintf( fp," \n");
        }
        fclose(fp);

        /* Write the E field out to a file "Ezk" */
       fp = fopen( "Ezk","w");
       for ( k=0; k < KE; k++ ) {
           for ( i=0; i < IE; i++ ) {
            Eav = ( ez[i][jc][k-1] + ez[i][jc][k])/2.;
             fprintf( fp,"%7.4f ",Eav);
          }
             fprintf( fp," \n");
        }
        fclose(fp);

        /* Write the E field out to a file "Exk" */
       fp = fopen( "Exk","w");
       for ( k=0; k < KE; k++ ) {
           for ( i=0; i < IE; i++ ) {
             Eav = ( ex[i-1][jc][k] + ex[i][jc][k])/2.;
             fprintf( fp,"%7.4f ",Eav);
          }
             fprintf( fp," \n");
        }
        fclose(fp);

       printf("T = %4.0f  \n ",T);
}
}

/* Fd3d_4.3.c. 3D FDTD, plane wave on a dielectric sphere. */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>
```

```
#define IE  40
#define JE  40
#define KE  40
#define ia  7
#define ja  7
#define ka  7
#define NFREQS 3

main ()
{
    float dx[IE][JE][KE],dy[IE][JE][KE],dz[IE][JE][KE];
    float ex[IE][JE][KE],ey[IE][JE][KE],ez[IE][JE][KE];
    float hx[IE][JE][KE],hy[IE][JE][KE],hz[IE][JE][KE];
    float ix[IE][JE][KE],iy[IE][JE][KE],iz[IE][JE][KE];
    float gax[IE][JE][KE],gay[IE][JE][KE],gaz[IE][JE][KE];
    float gbx[IE][JE][KE],gby[IE][JE][KE],gbz[IE][JE][KE];
    int l,m,n,i,j,k,ic,jc,kc,nsteps,n_pml;
    float ddx,dt,T,epsz,muz,pi,eaf,npml;
    int ib,jb,kb;
    float xn,xxn,xnum,xd,curl_e;
    float t0,spread,pulse;
    FILE *fp, *fopen();
    float ez_inc[JE],hx_inc[JE];
    float ez_low_m1,ez_low_m2,ez_high_m1,ez_high_m2;

      int ixh, jyh, kzh;
      float idx[IE][JE][KE], ihx[IE][JE][KE];
      float idy[IE][JE][KE], ihy[IE][JE][KE];
      float idz[IE][JE][KE], ihz[IE][JE][KE];

      float gi1[IE],gi2[IE],gi3[IE];
      float gj1[JE],gj2[JE],gj3[JE];
      float gk1[KE],gk2[KE],gk3[KE];
      float fi1[IE],fi2[IE],fi3[IE];
      float fj1[JE],fj2[JE],fj3[JE];
      float fk1[KE],fk2[KE],fk3[KE];

    float curl_h,curl_d;    float
radius[10],epsilon[10],sigma[10],eps,cond;
    int ii,jj,kk,numsph;
    float dist,xdist,ydist,zdist;

    float freq[NFREQS],arg[NFREQS];
    float real_pt[NFREQS][IE][JE],imag_pt[NFREQS][IE][JE];
    float amp[IE][JE],phase[IE][JE];
    float real_in[5],imag_in[5],amp_in[5],phase_in[5];

    ic = IE/2  ;
    jc = JE/2 ;
```

```
    kc = KE/2  ;
    ib = IE - ia - 1;
    jb = JE - ja - 1;
    kb = KE - ka - 1;
    pi = 3.14159;
    epsz = 8.8e-12;
    muz  = 4*pi*1.e-7;
    ddx = .01;                  /* Cell size */
    dt = ddx/6e8;               /* Time steps */

    /* Initialize the arrays */
    for ( j=0; j < JE; j++ ) {
       ez_inc[j] = 0.;
       hx_inc[j] = 0.;
       for ( k=0; k < KE; k++ ) {
           for ( i=0; i < IE; i++ ) {
           ex[i][j][k]= 0.0 ;
           ey[i][j][k]= 0.0 ;
           ez[i][j][k]= 0.0 ;
           dx[i][j][k]= 0.0 ;
           dy[i][j][k]= 0.0 ;
           dz[i][j][k]= 0.0 ;
           hx[i][j][k]= 0.0 ;
           hy[i][j][k]= 0.0 ;
           hz[i][j][k]= 0.0 ;
           ix[i][j][k]= 0.0 ;
           iy[i][j][k]= 0.0 ;
           iz[i][j][k]= 0.0 ;
           gax[i][j][k]= 1. ;
           gay[i][j][k]= 1. ;
           gaz[i][j][k]= 1. ;
           gbx[i][j][k]= 0. ;
           gby[i][j][k]= 0. ;
           gbz[i][j][k]= 0. ;
    }  }   }

    for ( n=0; i < NFREQS; n++ ) {
       real_in[n] = 0.;
       imag_in[n] = 0.;
       for ( j=0; j < JE; j++ ) {
          for ( i=0; i < IE; i++ ) {
             real_pt[n][i][j] = 0.;
             imag_pt[n][i][j] = 0.;
    }  }  }

 /* Parameters for the Fourier Transforms */

    freq[0] =  10.e6;
    freq[1] = 100.e6;
```

```
    freq[2] = 433.e6;

    for ( n=0; n < NFREQS; n++)
    {arg[n] = 2*pi*freq[n]*dt;
    printf( "%2d %6.2f %7.5f \n",n,freq[n]*1.e-6,arg[n]);
    }

/*   Boundary Conditions */

   for ( i=0; i < IE; i++ ) {
       for ( j=0; j < JE; j++ ) {
          for ( k=0; k < KE; k++ ) {
           idx[i][j][k] = 0.0;
           ihx[i][j][k] = 0.0;
           idy[i][j][k] = 0.0;
           ihy[i][j][k] = 0.0;
           idz[i][j][k] = 0.0;
           ihz[i][j][k] = 0.0;
    }   }  }

      for ( i=0; i < IE; i++ ) {
      gi1[i] = 0.;
      fi1[i] = 0.;
      gi2[i] = 1.;
      fi2[i] = 1.;
      gi3[i] = 1.;
      fi3[i] = 1.;
      }

      for ( j=0; j < JE; j++ ) {
      gj1[j] = 0.;
      fj1[j] = 0.;
      gj2[j] = 1.;
      fj2[j] = 1.;
      gj3[j] = 1.;
      fj3[j] = 1.;
      }

      for ( k=0; k < IE; k++ ) {
      gk1[k] = 0.;
      fk1[k] = 0.;
      gk2[k] = 1.;
      fk2[k] = 1.;
      gk3[k] = 1.;
      fk3[k] = 1.;
      }

       printf( "npml --> ");      printf( "f \n");
      for ( i=0; i < IE; i++ ) {
```

```
   printf( "%2d  %6.4f %6.4f %6.4f %6.4f\n",
      i,fi1[i],gi2[i],gi3[i]);
   printf( "    %6.4f %6.4f %6.4f %6.4f\n",
        gi1[i],fi2[i],fi3[i]);
}

for ( j=0; j < n_pml; j++ ) {
  xxn = (npml-j)/npml;
  xn  = .33*pow(xxn,3.);
  fj1[j] = xn;
  fj1[JE-j-1] = xn;
  gj2[j] = 1./(1.+xn);
  gj2[JE-j-1] = 1./(1.+xn);
  gj3[j] = (1.-xn)/(1.+xn);
  gj3[JE-j-1] = (1.-xn)/(1.+xn);
 xxn = (npml-j-.5)/npml;
  xn  = .33*pow(xxn,3.);
  gj1[j] = xn;
  gj1[JE-j-2] = xn;
  fj2[j] = 1./(1.+xn);
  fj2[JE-j-2] = 1./(1.+xn);
  fj3[j] = (1.-xn)/(1.+xn);
  fj3[JE-j-2] = (1.-xn)/(1.+xn);
}

 printf( "fj & gj \n");
for ( j=0; j < JE; j++ ) {
   printf( "%2d  %6.4f %6.4f %6.4f %6.4f\n",
      j,fj1[j],gj2[j],gj3[j]);
   printf( "    %6.4f %6.4f %6.4f %6.4f\n",
      gj1[j],fj2[j],fj3[j]);
}

for ( k=0; k < n_pml; k++ ) {
  xxn = (npml-k)/npml;
  xn  = .33*pow(xxn,3.);
  fk1[k] = xn;
  fk1[KE-k-1] = xn;
  gk2[k] = 1./(1.+xn);
  gk2[KE-k-1] = 1./(1.+xn);
  gk3[k] = (1.-xn)/(1.+xn);
  gk3[KE-k-1] = (1.-xn)/(1.+xn);
 xxn = (npml-k-.5)/npml;
  xn  = .33*pow(xxn,3.);
  gk1[k] = xn;
  gk1[KE-k-2] = xn;
  fk2[k] = 1./(1.+xn);
  fk2[KE-k-2] = 1./(1.+xn);
  fk3[k] = (1.-xn)/(1.+xn);
```

```
      fk3[KE-k-2] = (1.-xn)/(1.+xn);
    }
     scanf("%f", &npml);
     printf("%f \n", npml);
     n_pml = npml;

    for ( i=0; i < n_pml; i++ ) {
      xxn = (npml-i)/npml;
      xn  = .33*pow(xxn,3.);
       printf( "%d xn =  %8.4f xn = %8.4f \n",
                 i,xxn,xn);
      fi1[i] = xn;
      fi1[IE-i-1] = xn;
      gi2[i] = 1./(1.+xn);
      gi2[IE-i-1] = 1./(1.+xn);
      gi3[i] = (1.-xn)/(1.+xn);
      gi3[IE-i-1] = (1.-xn)/(1.+xn);
     xxn = (npml-i-.5)/npml;
      xn  = .33*pow(xxn,3.);
      gi1[i] = xn;
      gi1[IE-i-2] = xn;
      fi2[i] = 1./(1.+xn);
      fi2[IE-i-2] = 1./(1.+xn);
      fi3[i] = (1.-xn)/(1.+xn);
      fi3[IE-i-2] = (1.-xn)/(1.+xn);
    }

    printf( "fk & gk \n");
    for ( k=0; k < JE; k++ ) {
       printf( "%2d  %6.4f %6.4f %6.4f %6.4f\n",
          k,fk1[k],gk2[k],gk3[k]);
       printf( "    %6.4f %6.4f %6.4f %6.4f\n",
            gk1[k],fk2[k],fk3[k]);
    }

  /* Specify the dielectric sphere */

     epsilon[0] = 1.;
     sigma[0]   = 0;

     printf( "Number spheres --> ");
     scanf("%d", &numsph);
        printf( "numsph= %d \n ",numsph);

     for (n = 1; n <= numsph; n++) {
        printf( "Sphere radius (cells), epsilon, sigma --> ");
        scanf("%f %f %f", &radius[n], &epsilon[n], &sigma[n]);
        printf( "Radius = %6.2f  Eps = %6.2f Sigma = %6.2f \n ",
           radius[n],epsilon[n],sigma[n]);
```

```
    }

    for (n = 0; n <= numsph; n++) {
       printf( "Radius = %5.2f  Eps = %6.2f Sigma = %6.2f \n ",
       radius[n],epsilon[n],sigma[n]);
    }

 /*       Calculate gax,gbx  */
    for ( i = ia; i < ib; i++ ) {
    for ( j = ja; j < jb; j++ ) {
    for ( k = ka; k < kb; k++ ) {
    eps = epsilon[0];
    cond = sigma[0];
             ydist = (jc-j);
             xdist = (ic-i-.5);
             zdist = (kc-k);
          dist = sqrt(pow(xdist,2.) + pow(ydist,2.)
               + pow(zdist,2.));
             for (n=1; n<= numsph; n++) {
                if( dist <= radius[n]) {
                eps  =  epsilon[n];
                cond =  sigma[n] ;
                }
             }
           gax[i][j][k] = 1./(eps + (cond*dt/epsz));
           gbx[i][j][k] = cond*dt/epsz;
     }}}

 /*       Calculate gay,gby  */
    for ( i = ia; i < ib; i++ ) {
    for ( j = ja; j < jb; j++ ) {
    for ( k = ka; k < kb; k++ ) {
    eps = epsilon[0];
    cond = sigma[0];
             xdist = (ic-i);
             ydist = (jc-j-.5);
             zdist = (kc-k);
    dist = sqrt(pow(xdist,2.) + pow(ydist,2.) + pow(zdist,2.));
             for (n=1; n<= numsph; n++) {
                if( dist <= radius[n]) {
                eps  = epsilon[n] ;
                cond = sigma[n] ;
                }
             }
           gay[i][j][k] = 1./(eps + (cond*dt/epsz));
           gby[i][j][k] = cond*dt/epsz;
     }}}
```

```
       printf( " Gay  \n");
       for ( j=ja; j <= jb; j++ ) {
          printf( "%3d",j);
          for ( i=ia; i <= ib; i++ ) {
             printf( "%5.2f",gay[i][j][kc]);
          }  printf( " \n");
       } fclose(fp);

  /*       Calculate gaz,gbz  */
       for ( i = ia; i < ib; i++ ) {
       for ( j = ja; j < jb; j++ ) {
       for ( k = ka; k < kb; k++ ) {
       eps = epsilon[0];
       cond = sigma[0];
                xdist = (ic-i);
                ydist = (jc-j);
                zdist = (kc-k-.5);
                dist = sqrt(pow(xdist,2.) + pow(ydist,2.) +
pow(zdist,2.));
                for (n=1; n<= numsph; n++) {
                   if( dist <= radius[n]) {
                   eps  =  epsilon[n] ;
                   cond =  sigma[n];
                }  }
              gaz[i][j][k] = 1./(eps + (cond*dt/epsz));
              gbz[i][j][k] = cond*dt/epsz;
        }}}

       printf( " Gaz  \n");
       for ( j=ja; j <= jb; j++ ) {
          printf( "%3d",j);
          for ( i=ia; i <= ib; i++ ) {
             printf( "%5.2f",gaz[i][j][kc]);
          }  printf( " \n");
       } fclose(fp);

    t0 = 40.0;
    spread = 10.0;
    T = 0;
    nsteps = 1;

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)  {
       T = T + 1;
```

```
/*  ----   Start of the Main FDTD loop ----  */

    /* Calculate the incident buffer */

          for ( j=1; j < JE; j++ ) {
            ez_inc[j] = ez_inc[j] + .5*( hx_inc[j-1] - hx_inc[j] );
          }

    /* Fourier Tramsform of the incident field */
        for ( m=0; m < NFREQS ; m++ )
        {  real_in[m] = real_in[m] + cos(arg[m]*T)*ez_inc[ja-1] ;
           imag_in[m] = imag_in[m] - sin(arg[m]*T)*ez_inc[ja-1] ;
        }

       /*  Source */

    /* pulse =  sin(2*pi*400*1e6*dt*T);  */
       pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
       ez_inc[3]  = pulse;
       printf("%4.0f  %6.2f \n ",T,pulse);

    /* Boundary conditions for the incident buffer*/

      ez_inc[0] = ez_low_m2;
      ez_low_m2 = ez_low_m1;
      ez_low_m1 = ez_inc[1];

      ez_inc[JE-1]  = ez_high_m2;
      ez_high_m2 = ez_high_m1;
      ez_high_m1 = ez_inc[JE-2];

     /* Calculate the Dx field */

      for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
            for ( k=1; k < KE; k++ ) {
                curl_h = ( hz[i][j][k] - hz[i][j-1][k]
                          - hy[i][j][k] + hy[i][j][k-1]) ;
                idx[i][j][k] = idx[i][j][k] + curl_h;
                dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
              + gj2[j]*gk2[k]*.5*(curl_h + gi1[i]*idx[i][j][k]);
      }  }  }

     /* Calculate the Dy field */

       for ( i=1; i < IE; i++ ) {
          for ( j=1; j < JE; j++ ) {
             for ( k=1; k < KE; k++ ) {
                curl_h = ( hx[i][j][k] - hx[i][j][k-1]
```

```
                    - hz[i][j][k] + hz[i-1][j][k]) ;
               idy[i][j][k] = idy[i][j][k] + curl_h;
               dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
               + gi2[i]*gk2[k]*.5*( curl_h + gj1[j]*idy[i][j][k]);
     }  } }

      /* Incident Dy */
      for ( i=ia; i <= ib; i++ ) {
         for ( j=ja; j <= jb-1; j++ ) {
            dy[i][j][ka]   = dy[i][j][ka]   - .5*hx_inc[j];
            dy[i][j][kb+1] = dy[i][j][kb+1] + .5*hx_inc[j];
      }  }

   /* Calculate the Dz field */

      for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
            for ( k=0; k < KE; k++ ) {
               curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                        - hx[i][j][k] + hx[i][j-1][k]) ;
               idz[i][j][k] = idz[i][j][k] + curl_h;
               dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
               + gi2[i]*gj2[j]*.5*( curl_h + gk1[k]*idz[i][j][k] );
     }  }  }

      /* Incident Dz */
      for ( i=ia; i <= ib; i++ ) {
         for ( k=ka; k <= kb; k++ ) {
            dz[i][ja][k] = dz[i][ja][k] + .5*hx_inc[ja-1];
            dz[i][jb][k] = dz[i][jb][k] - .5*hx_inc[jb];
      }  }

     /*  Source */

/* Calculate the E from D field */
      /* Remember: part of the PML is E=0 at the edges */
      for ( i=1; i < IE-1; i++ ) {
         for ( j=1; j < JE-1; j++ ) {
            for ( k=1; k < KE-1; k++ ) {
            ex[i][j][k] = gax[i][j][k]*(dx[i][j][k] - ix[i][j][k]);
            ix[i][j][k] = ix[i][j][k] + gbx[i][j][k]*ex[i][j][k];
            ey[i][j][k] = gay[i][j][k]*(dy[i][j][k] - iy[i][j][k]);
            iy[i][j][k] = iy[i][j][k] + gby[i][j][k]*ey[i][j][k];
            ez[i][j][k] = gaz[i][j][k]*(dz[i][j][k] - iz[i][j][k]);
            iz[i][j][k] = iz[i][j][k] + gbz[i][j][k]*ez[i][j][k];
      }  }  }

   /* Calculate the Fourier transform of Ex. */
    for ( j=0; j < JE; j++ )
```

```
{   for ( i=0; i < JE; i++ )
   {   for ( m=0; m < NFREQS; m++ )
      { real_pt[m][i][j] = real_pt[m][i][j] +
                           cos(arg[m]*T)*ez[i][j][kc] ;
        imag_pt[m][i][j] = imag_pt[m][i][j] +
                           sin(arg[m]*T)*ez[i][j][kc] ;
}  }  }

/* Calculate the incident field */

     for ( j=0; j < JE-1; j++ ) {
       hx_inc[j] = hx_inc[j] + .5*( ez_inc[j] - ez_inc[j+1] );
     }

/* Calculate the Hx field */
 for ( i=0; i < IE; i++ ) {
    for ( j=0; j < JE-1; j++ ) {
       for ( k=0; k < KE-1; k++ ) {
           curl_e = ( ey[i][j][k+1] - ey[i][j][k]
                    - ez[i][j+1][k] + ez[i][j][k]) ;
           ihx[i][j][k] = ihx[i][j][k]  + curl_e;
            hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
           + fj2[j]*fk2[k]*.5*( curl_e + fi1[i]*ihx[i][j][k] );
 }  }  }

  /* Incident Hx */
  for ( i=ia; i <= ib; i++ ) {
     for ( k=ka; k <= kb; k++ ) {
        hx[i][ja-1][k] = hx[i][ja-1][k] + .5*ez_inc[ja];
        hx[i][jb][k]   = hx[i][jb][k]   - .5*ez_inc[jb];
  }  }

 /* Calculate the Hy field */
  for ( i=0; i < IE-1; i++ ) {
     for ( j=0; j < JE; j++ ) {
        for ( k=0; k < KE-1; k++ ) {
           curl_e = ( ez[i+1][j][k] - ez[i][j][k]
                    - ex[i][j][k+1] + ex[i][j][k]) ;
           ihy[i][j][k] = ihy[i][j][k] + curl_e ;
           hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
           + fi2[i]*fk3[k]*.5*( curl_e + fj1[j]*ihy[i][j][k] );
 }  }  }

  /* Incident Hy */

  for ( j=ja; j <= jb; j++ ) {
     for ( k=ka; k <= kb; k++ ) {
        hy[ia-1][j][k] = hy[ia-1][j][k] - .5*ez_inc[j];
        hy[ib][j][k]   = hy[ib][j][k]   + .5*ez_inc[j];
```

```
       }  }

      /* Calculate the Hz field */

       for ( i=0; i < IE-1; i++ ) {
          for ( j=0; j < JE-1; j++ ) {
             for ( k=0; k < KE; k++ ) {
                curl_e = ( ex[i][j+1][k] - ex[i][j][k]
                       -   ey[i+1][j][k] + ey[i][j][k] );
                ihzl[i][j][k] = ihzl[i][j][k] + curl_e;
                hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
               + fi2[i]*fj2[j]*.5*( curl_e + fk1[k]*ihzl[i][j][k] );
      }  }  }

    }
/*  ----  End of the main FDTD loop ---- */

/* Write the E field out to a file "Ez" */
   fp = fopen( "Ez","w");
   for ( j=0; j < JE; j++ ) {
       for ( i=0; i < IE; i++ ) {
         fprintf( fp,"%9.6f ",ez[i][j][kc]);
        }
         fprintf( fp," \n");
        }
        fclose(fp);

       printf("T = %4.0f \n",T);

/* Calculate the Fouier amplitude and phase of the total field */
   for ( m=0; m < NFREQS; m++ )
       {
       if( m == 0)       fp = fopen( "amp1","w");
       else if( m == 1)  fp = fopen( "amp2","w");
       else if( m == 2)  fp = fopen( "amp3","w");
       {   printf( "%2d  %7.2f  MHz\n",m,freq[m]*1.e-6);
           for ( j=ja; j <= jb; j++ )
           {  if( gaz[ic][j][kc] < 1.00 )
              {  amp[ic][j]  = (1./amp_in[m])
                 *sqrt( pow(real_pt[m][ic][j],2.) +
                        pow(imag_pt[m][ic][j],2.));
                 printf( "%2d %9.4f \n",jc-j,amp[ic][j]);
                 fprintf( fp," %9.4f \n",amp[ic][j]);
               }
           }
       }
        fclose(fp);
       }
}
}
```

5

EXAMPLES OF ELECTROMAGNETIC SIMULATION USING FDTD

This chapter presents three examples of electromagnetic simulation that illustrate the power and flexibility of the FDTD method. Section 5.1 describes a one-dimensional simulation of an optical pulse in a nonlinear material similar to material in a nonlinear optical fiber. Section 5.2 describes the use of the FDTD method in characterizing a two-dimensional electromagnetic structure. Section 5.3 contains a three-dimensional simulation of an *LC* loop, and describes how signal-processing techniques can be used in characterizing the loop.

5.1 NONLINEAR OPTICAL PULSE SIMULATION

As light is also an electromagnetic wave, it can also be simulated by FDTD. Light waves are at very high frequencies, in the range of terahertz (THz), or 10^{12} Hz. At optical frequencies, materials can have both dispersive and nonlinear properties. The dispersive properties imply that the dielectric properties are frequency dependent. Because a pulse in the time domain is composed of frequencies over a certain range, the different frequencies see different dielectric properties, and therefore, travel at different speeds (1, 2). For this reason, a pulse of light in an optical fiber cable, for example, will slowly broaden because of the dispersive nature of the properties of the optical fiber. However, it has been demonstrated that the nonlinearities of a material can lead to the formation of a soliton, a pulse that maintains its amplitude and width (3).

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

In this section, the simulation of a one-dimensional nonlinear optic pulse is described. The simulation demonstrates the propagation of a soliton. The formulation of the dispersive and nonlinear properties is once again made more efficient by the use of Z transforms (4).

We start with the normalized Maxwell's equations:

$$\frac{\partial D_x}{\partial t} = \frac{\partial H_y}{\partial z}, \tag{5.1a}$$

$$D_x = \varepsilon_\infty E_x + P_{\mathrm{L}} + P_{\mathrm{NL}}, \tag{5.1b}$$

$$\frac{\partial H_y}{\partial t} = \frac{\partial E_x}{\partial z}. \tag{5.1c}$$

Note that the relationship between the electric field and the flux density in Eq. (5.1b) contains two additional terms, the linear polarization P_{L} and the nonlinear polarization P_{NL}.

In the frequency domain, the linear polarization is given by

$$P_{\mathrm{L}}(\omega) = \frac{(\varepsilon_s - \varepsilon_\infty)}{1 + 2\delta_n \left(\frac{j\omega}{\omega_{\mathrm{L}}}\right) - \left(\frac{\omega}{\omega_{\mathrm{L}}}\right)^2} E(\omega). \tag{5.2}$$

It is desirable to transform this into the Z domain for implementation into FDTD. To do so, we first make the following change of variables:

$$\alpha_{\mathrm{L}} = \omega_{\mathrm{L}} \cdot \delta_{\mathrm{L}}, \tag{5.3a}$$

$$\beta_{\mathrm{L}} = \omega_{\mathrm{L}} \sqrt{1 - \delta_{\mathrm{L}}^2}, \tag{5.3b}$$

$$\gamma_{\mathrm{L}} = \frac{\omega_{\mathrm{L}} \cdot (\varepsilon_s - \varepsilon_\infty)}{\sqrt{1 - \delta_{\mathrm{L}}^2}}. \tag{5.3c}$$

Now Eq. (5.2) can be written as

$$P_{\mathrm{L}}(\omega) = \frac{\gamma_{\mathrm{L}} \beta_{\mathrm{L}}}{(\alpha_{\mathrm{L}}^2 + \beta_{\mathrm{L}}^2) + j\omega 2\alpha_{\mathrm{L}} - \omega^2} E(\omega). \tag{5.4}$$

From Table A.1, the Z transform of Eq. (5.4) is

$$P_{\mathrm{L}}(z) = \frac{\gamma_{\mathrm{L}} \mathrm{e}^{-\alpha_{\mathrm{L}} \cdot \Delta t} \cdot \sin(\beta_{\mathrm{L}} \cdot \Delta t) z^{-1}}{1 - 2\mathrm{e}^{-\alpha_{\mathrm{L}} \cdot \Delta t} \cdot \cos(\beta_{\mathrm{L}} \cdot \Delta t) z^{-1} + \mathrm{e}^{-2\alpha_{\mathrm{L}} \cdot \Delta t} z^{-2}} \Delta t \cdot E(z). \tag{5.5}$$

We define a new variable $S_{\mathrm{L}}(z)$ related to $P_{\mathrm{L}}(z)$:

$$P_{\mathrm{L}}(z) = z^{-1} S_{\mathrm{L}}(z).$$

The term $S_L(z)$ is calculated by

$$S_L(z) = c_1 \cdot z^{-1} S_L(z) - c_2 \cdot z^{-2} S_L(z) + c_3 E(z), \tag{5.6a}$$

where

$$c_1 = 2 \cdot e^{-\alpha_L \cdot \Delta t} \cdot \cos(\beta_L \cdot \Delta t); \tag{5.6b}$$

$$c_2 = e^{-2\alpha_L \cdot \Delta t}; \tag{5.6c}$$

$$c_3 = \gamma_L \cdot \Delta t \cdot e^{-\alpha_L \cdot \Delta t} \cdot \sin(\beta_L \cdot \Delta t). \tag{5.6d}$$

In the sampled time domain, the linear polarization is calculated by

$$P_L^n = S_L^{n-1} \tag{5.7a}$$

$$S_L^n = c_1 \cdot S_L^{n-1} - c_2 \cdot S_L^{n-2} + c_3 E^n. \tag{5.7b}$$

The nonlinear polarization is separated into two parts:

$$P_{NL} = P_K + P_R. \tag{5.8}$$

The part P_K is known as the *Kerr effect*, given in the time domain by

$$P_K(t) = \chi^{(3)} \alpha E^3(t). \tag{5.9a}$$

The part P_R is the Raman scattering, which in the time domain is given by

$$P_R(t) = \chi^{(3)}(1-\alpha)E(t) \cdot \int_0^t g_R(t-\tau)E^2(\tau)\mathrm{d}\tau. \tag{5.9b}$$

We will start with the Raman scattering. In the frequency domain, g_R in the integral in Eq. (5.9b) is

$$g_R(\omega) = \frac{1}{1 + 2\delta_{NL}\left(\frac{j\omega}{\omega_{NL}}\right) - \left(\frac{\omega}{\omega_{NL}}\right)^2}. \tag{5.10}$$

We define an integral

$$I_R(t) = \chi^{(3)}(1-\alpha)\int_0^t g_R(t-\tau)E^2(\tau)\mathrm{d}\tau, \tag{5.11}$$

which is the same as Eq. (5.9b), except that the term $E(t)$ in front of the integral has been omitted. We will now make a change of variables similar to that in Eq. (5.3):

$$\alpha_{\mathrm{R}} = \omega_{\mathrm{NL}} \cdot \delta_{\mathrm{NL}}, \tag{5.12a}$$

$$\beta_{\mathrm{R}} = \omega_{\mathrm{NL}} \cdot \sqrt{1 - \delta_{\mathrm{NL}}^2}, \tag{5.12b}$$

$$\gamma_{\mathrm{NL}} = \frac{\omega_{\mathrm{NL}} \cdot \chi^{(3)} \cdot (1 - \alpha)}{\sqrt{1 - \delta_{\mathrm{NL}}^2}}. \tag{5.12c}$$

Since g_{R} is also a second-order Lorentz term, the Z transform is similar to Eq. (5.4), so the Z transform of Eq. (5.11) is

$$I_{\mathrm{R}}(z) = \frac{\gamma_{\mathrm{R}} \mathrm{e}^{-\alpha_{\mathrm{R}} \cdot \Delta t} \cdot \sin(\beta_{\mathrm{R}} \cdot \Delta t) \cdot z^{-1}}{1 - 2\mathrm{e}^{-\alpha_{\mathrm{R}} \cdot \Delta t} \cdot \cos(\beta_{\mathrm{R}} \cdot \Delta t) z^{-1} + \mathrm{e}^{-2\alpha_{\mathrm{R}} \cdot \Delta t} z^{-2}} \cdot \Delta t \cdot E^2(z). \tag{5.13}$$

At this point, the $E^2(z)$ term requires some comment. Z transform theory does not, in general, cover nonlinear processes. However, if we consider $E^2(t)$ to be a time domain function, there is nothing to stop us from considering the Z transform of this function.

The z^{-1} in the numerator of Eq. (5.13) means we would have to store an $E^2(z)$ term to make the calculation; instead, we prefer to calculate $S_{\mathrm{R}}(z)$, which is related to $I_{\mathrm{R}}(z)$ by

$$I_{\mathrm{R}}(z) = z^{-1} S_{\mathrm{R}}(z).$$

$S_{\mathrm{R}}(z)$ is calculated similarly to $S_{\mathrm{L}}(z)$:

$$S_{\mathrm{R}}(z) = d_1 \cdot z^{-1} S_{\mathrm{R}}(z) - c_2 \cdot z^{-2} S_{\mathrm{R}}(z) + c_3 E^2(z), \tag{5.14}$$

where

$$d_1 = 2 \cdot \mathrm{e}^{-\alpha_{\mathrm{R}} \cdot \Delta t} \cdot \cos(\beta_{\mathrm{R}} \cdot \Delta t); \tag{5.15a}$$

$$d_2 = \mathrm{e}^{-2\alpha_{\mathrm{R}} \cdot \Delta t}; \tag{5.15b}$$

$$d_3 = \gamma_{\mathrm{R}} \cdot \Delta t \cdot \mathrm{e}^{-\alpha_{\mathrm{R}} \cdot \Delta t} \cdot \sin(\beta_{\mathrm{R}} \cdot \Delta t). \tag{5.15c}$$

The polarization due to the Raman scattering is calculated from

$$P_{\mathrm{R}}^n = E^n \cdot I_{\mathrm{R}}^n = E^n \cdot S_{\mathrm{R}}^{n-1} \tag{5.16a}$$

$$S_{\mathrm{R}}^n = d_1 \cdot S_{\mathrm{R}}^{n-1} - d_2 \cdot S_{\mathrm{R}}^{n-2} + d_3 (E^2)^n. \tag{5.16b}$$

The Kerr effect will require a somewhat different approach. Start by taking a first-order Taylor series expansion of $E^3(t)$ around the point t_{n-1} and evaluating it at t_n:

$$\begin{aligned} E^3(t_n) &\cong E^3(t_{n-1}) + \frac{\mathrm{d}}{\mathrm{d}t}(E^3(t_{n-1}))(t_n - t_{n-1}) \\ &\cong E^3(t_{n-1}) + 3 \cdot E^2(t_{n-1}) \left[\frac{E\left(t_n\right) - E(t_{n-1})}{t_n - t_{n-1}} \right] (t_n - t_{n-1}) \\ &= E^3(t_{n-1}) + 3E^2(t_{n-1})[E(t_n) - E(t_{n-1})] \\ &= 3E^2(t_{n-1}) \cdot E(t_n) - 2E^3(t_{n-1}). \end{aligned}$$

The times t_{n-1} and t_n correspond to times in the FDTD simulation, so we write

$$(E^n)^3 = 3(E^{n-1})^2 \cdot E^n - 2(E^{n-1})^3. \tag{5.17}$$

Substituting this into Eq. (5.9a), we obtain the Kerr polarization in the sampled time domain:

$$P_{\mathrm{K}}^n = \chi^{(3)}\alpha[3(E^{n-1})^2 \cdot E^n - 2(E^{n-1})^3]. \tag{5.18}$$

Now that we have all the polarization terms in a form suitable for FDTD, we can look back to Eq. (5.1b) and develop the calculation of E_x from D_x:

$$\begin{aligned} \varepsilon_\infty E_x^n &= D_x^n - P_{\mathrm{L}}^n - P_{\mathrm{R}}^n - P_{\mathrm{K}}^n \\ &= D_x^n - S_{\mathrm{L}}^{n-1} - S_{\mathrm{R}}^{n-1} E_x^n - \chi^{(3)}\alpha[3(E_x^{n-1})^2 \cdot E_x^n - 2(E_x^{n-1})^3]. \end{aligned}$$

Collect all the E_x^n terms on the left side:

$$\varepsilon_\infty E_x^n + S_{\mathrm{R}}^{n-1} E_x^n + \chi^{(3)}\alpha \cdot 3(E_x^{n-1})^2 \cdot E_x^n = D_x^n - S_{\mathrm{L}}^{n-1} + \chi^{(3)}\alpha \cdot 2(E_x^{n-1})^3.$$

E_x is calculated from D_x by the following equations:

$$E_x^n = \frac{D_x^n - S_{\mathrm{L}}^{n-1} + \chi^{(3)}\alpha \cdot 2(E_x^{n-1})^3}{\varepsilon_\infty + S_{\mathrm{R}}^{n-1} + \chi^{(3)}\alpha \cdot 3(E_x^{n-1})^2}, \tag{5.19a}$$

$$S_{\mathrm{L}}^n = c_1 \cdot S_{\mathrm{L}}^{n-1} - c_2 \cdot S_{\mathrm{L}}^{n-2} + c_3 E^n, \tag{5.19b}$$

$$S_{\mathrm{R}}^n = d_1 \cdot S_{\mathrm{R}}^{n-1} - d_2 \cdot S_{\mathrm{R}}^{n-2} + d_3 (E^2)^n. \tag{5.19c}$$

The accuracy of Eq. (5.19) was verified by an analytic formulation of the reflection coefficient from a nonlinear material (4).

Figure 5.1 shows the simulation of a pulse in a nonlinear dispersive material. The properties of this material are as follows: $\varepsilon_\infty = 2.25$, $\varepsilon_s = 5.25$, $\omega_{\mathrm{L}} = 2\pi \times 63.7$ THz, $\delta_{\mathrm{L}} = 0.00025$, $\chi^{(3)} = 0.07$, $\alpha = 0.7$,

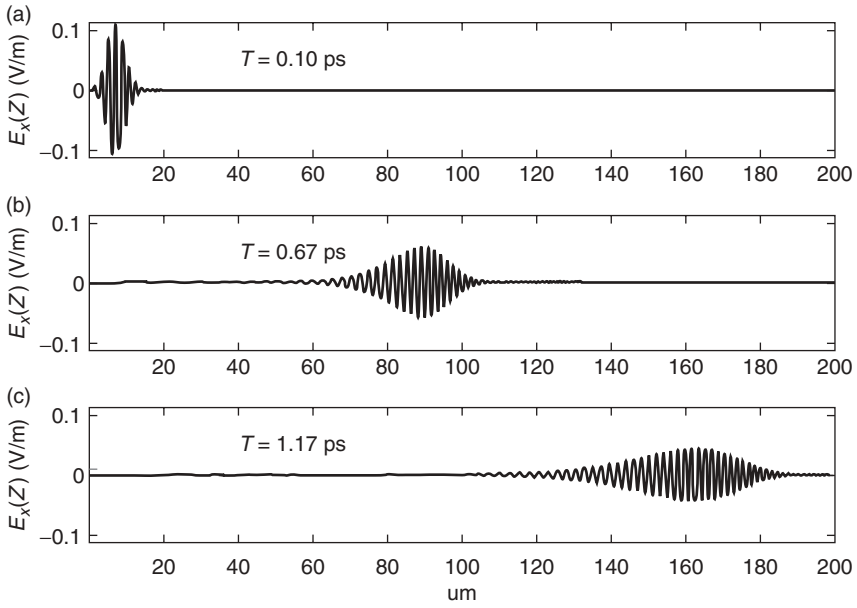

Figure 5.1 Propagation of a pulse in a nonlinear, dispersive medium. The magnitude of the pulse is not great enough to engage the nonlinearity. As the pulse propagates, dispersion causes the pulse to broaden. (a) $T = 0.10$, (b) $T = 0.67$, and (c) $T = 1.17$ ps.

$\omega_{\mathrm{NL}} = 2\pi \times 14.8$ THz, and $\delta_{\mathrm{NL}} = 0.336$. The pulse begins with a maximum amplitude of 0.1 V/m. This is not strong enough to adequately engage the nonlinear terms. Therefore, the material is predominantly dispersive, and we see the pulse spread as it propagates through the medium.

Figure 5.2a is a display of the time domain propagation of the pulse of Fig. 5.1 as it passes three different points in the problem space. It is instructive to look at the Fourier transforms of these waveforms, as shown in Fig. 5.2b. Although the time domain pulses at 1, 60, and 120 μm look very different, their Fourier magnitudes appear identical. It is the changes in the phase caused by the linear polarization that have led to the dispersion, or spreading, of the pulse shape.

Figure 5.3 shows the results of a simulation similar to the one in Fig. 5.1, except that the amplitude of the pulse begins at 1 V/m. This amplitude is strong enough to engage the nonlinearity. The result is the formation of a soliton. Notice that the waveforms at 0.67 and 1.17 ps have about the same shape and amplitude. Notice also a second pulse of smaller amplitude that has moved ahead of the main pulse. This is referred to as the *daughter soliton*, and is also characteristic of soliton propagation (3).

Figure 5.4a shows the time domain waveforms and the corresponding Fourier magnitudes, similar to Eq. (5.2). In Fig. 5.4b, we see that as the waveform propagates, there is a "redshift" in the Fourier spectrum as well as a sharpening of the spectrum, characteristic of soliton propagation (3).

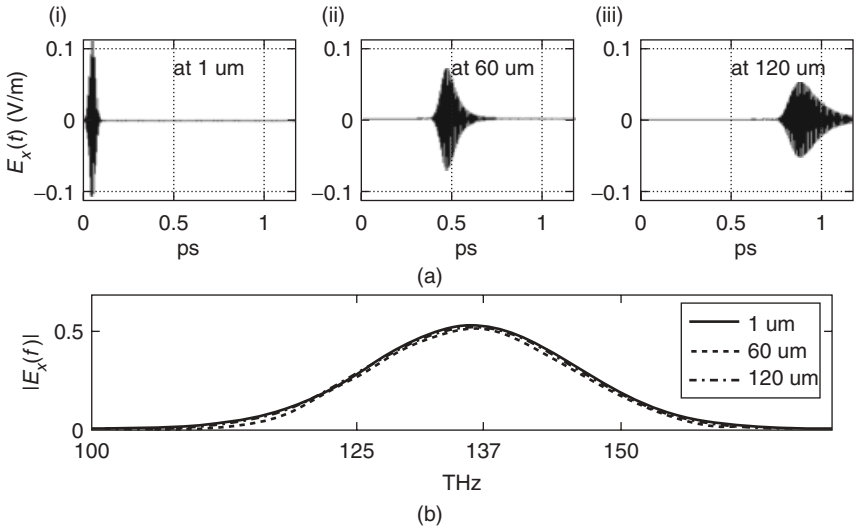

Figure 5.2 (a) The time domain E_x field at (i) 1, (ii) 60, and (iii) 120 μm. (b) The magnitude of the Fourier transforms of the time domain waveforms.

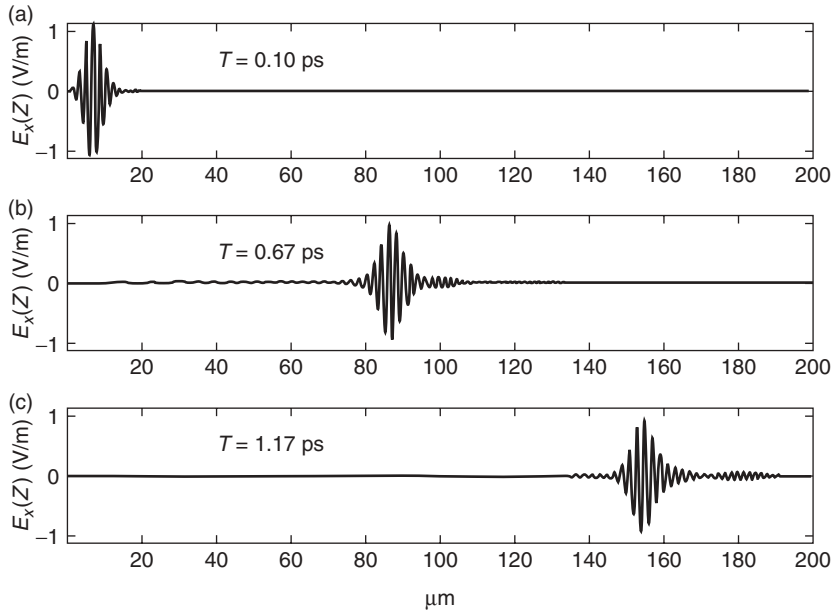

Figure 5.3 Propagation of a pulse in a nonlinear, dispersive medium. In contrast to Fig. 5.1, the magnitude of the pulse is great enough to engage the nonlinearity. The nonlinearity causes the formation of a soliton, which maintains its amplitude and width. Notice the daughter soliton that precedes the main pulse. (a) $T = 0.10$, (b) $T = 0.67$, and (c) $T = 1.17$ ps.

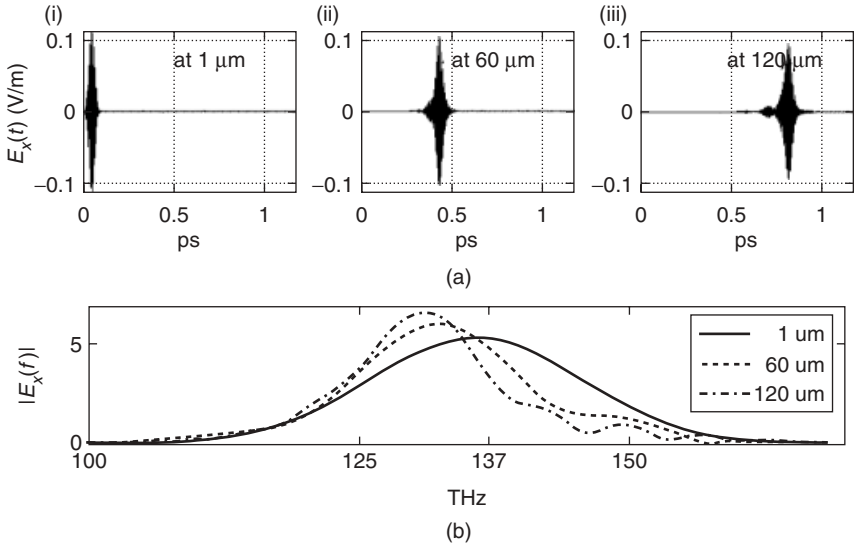

Figure 5.4 (a) The time domain E fields at (i) 1, (ii) 60, and (iii) 120 μm. (b) The magnitudes of the Fourier transforms show a shift caused by the nonlinearity.

This section has presented one-dimensional FDTD simulation of a pulse in nonlinear, dispersive media. This formulation has been used in the three-dimensional simulation of a nonlinear optical fiber (5).

PROBLEM SET 5.1

1. Using the program fd1d_nlp.c, find the smallest amplitude that will induce soliton propagation within the 200-μm problem space.

5.2 FINDING THE EIGENFUNCTIONS OF A TWO-DIMENSIONAL EM CAVITY

Eigenfunctions and eigenvalues often play a significant role in many branches of engineering and science. For instance, consider the two-dimensional cavity in Fig. 5.5. It has metal walls and is filled with air. We might be interested in knowing which electric fields will oscillate in this cavity. If the dimensions of the cavity are a in the x direction and b in the y direction, one such group of fields is given by (6)

$$E_z^{mn}(x, y) = \frac{1}{\sqrt{ab}} \sin\left(\frac{\pi mx}{a}\right) \sin\left(\frac{\pi ny}{b}\right), \tag{5.20}$$

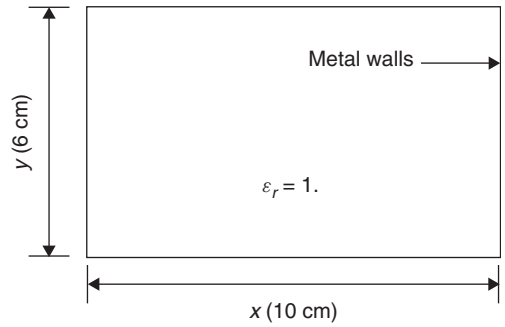

Figure 5.5 A two-dimensional air-filled cavity with metal walls.

where m and n are integers. The resulting E_z^{mn} will oscillate in time at a frequency given by

$$f_{mn} = \frac{c_0}{2}\sqrt{\left(\frac{m}{a}\right)^2 + \left(\frac{n}{b}\right)^2}, \tag{5.21}$$

where c_0 is the speed of light in vacuum.

We regard the E_z^{mn} of Eq. (5.20) to be the *eigenfunctions*, and the frequencies f_{mn} of Eq. (5.21) to be the *eigenfrequencies*. (We use the term *eigenfrequency* to be a little more specific than eigenvalue.) We can write each eigenfunction explicitly as a function of space and time:

$$E_z^{mn}(x, y, t) = \frac{1}{\sqrt{ab}} \sin\left(\frac{\pi m x}{a}\right) \sin\left(\frac{\pi n y}{b}\right) \mathrm{e}^{j2\pi f_{mn} t}.$$

The eigenfunctions of Eq. (5.20) have another very attractive property: they are *orthonormal*. This means the following:

$$\int_0^a \int_0^b E_z^{mn}(x, y) E_z^{lk}(x, y) \mathrm{d}y\mathrm{d}x = \begin{cases} 1 & \text{if } m = l \text{ and } n = k, \\ 0 & \text{otherwise.} \end{cases}$$

The orthonormal property means that any function inside the cavity can be written as a superposition of these orthonormal functions.

$$E_z(x, y) = \sum_{m,n} d_{mn} E_z^{mn}(x, y). \tag{5.22}$$

The d_{mn} are determined by taking the inner product of the function with the eigenfunctions.

$$d_{mn} = \int_0^a \int_0^b E_z(x, y) E_z^{mn}(x, y) \mathrm{d}y\mathrm{d}x. \tag{5.23}$$

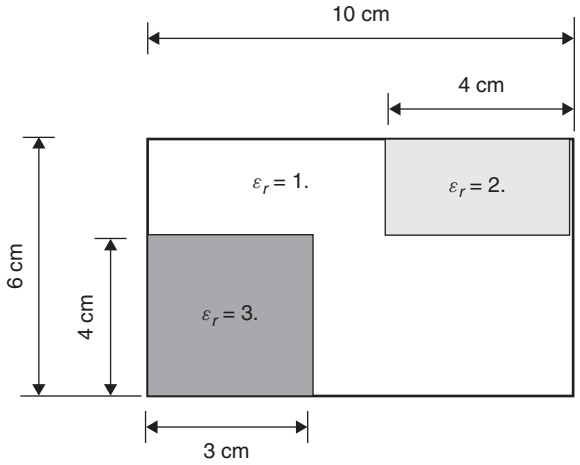

Figure 5.6 A two-dimensional cavity with metal walls, filled with two different dielectric materials and air.

This process of describing a function in terms of its eigenfunctions plays a significant role in engineering and science.

The problem comes when we do not know the eigenfunctions and corresponding eigenfrequencies. The determination of the eigenfunctions for Fig. 5.5 was quite straightforward. However, waveguides can be filled with dielectric materials to achieve various properties (7). Then the calculation is not so easy. In this section, a method is described to determine the eigenfunctions and eigenfrequencies of a two-dimensional cavity, even when the inside is fairly complicated. We use as an example the cavity described in Fig. 5.6. This cavity is of no known practical use, but presents a challenging structure.

We start by initializing a narrow Gaussian field E_z^0 in the cavity centered at the points $x_0 = 6$ cm, $y_0 = 2$ cm, as shown in Fig. 5.7. We can write E_z^0 as an expansion of the orthonormal eigenfunctions, even if we do not yet know what the eigenfunctions are.

$$E_z^0(x, y, t) = \sum_n d_n \phi_n(x, y) \mathrm{e}^{j2\pi f_n t}. \tag{5.24}$$

(For simplicity, the eigenfunctions are just labeled with one integer, n.)

First we look for the eigenfrequencies f_n. We start the FDTD simulation, and the E_z field expands out and interacts with the dielectric material and the metal walls, as shown in Fig. 5.7. However, while the simulation is running, we monitor the time domain E_z field at x_0, y_0, which mathematically will have the following form

$$E_z^0(x_0, y_0, t) = \sum_n d_n \phi_n(x_0, y_0) \mathrm{e}^{j2\pi f_n t}.$$

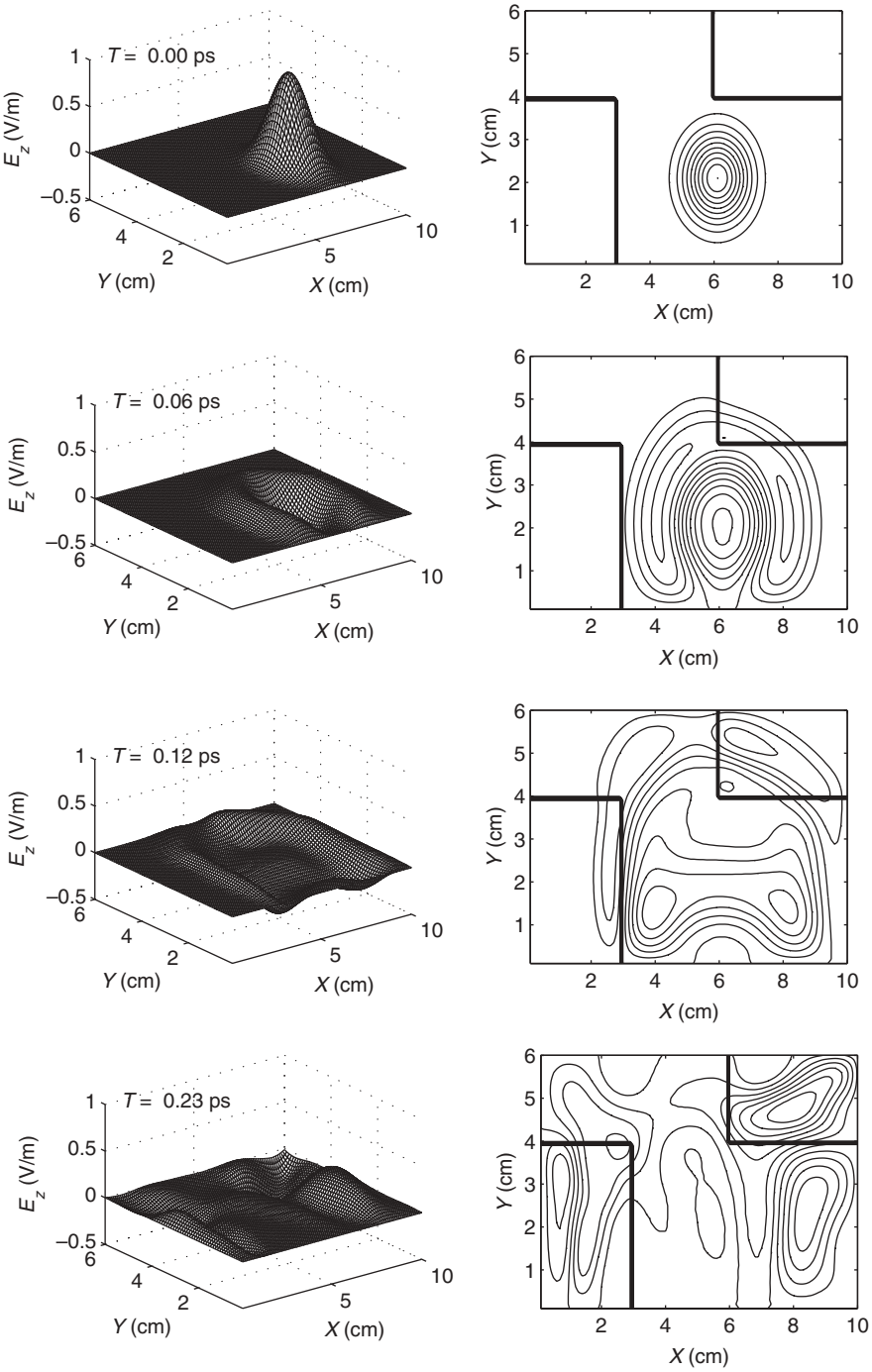

Figure 5.7 At $T = 0$, a pulse is initialized in the problem space, modeling the two-dimensional cavity of Fig. 5.6. As time progresses, the waveform spreads and interacts with the dielectric material and the metal walls. While this is happening, the time domain E_z at the origin of the pulse at $x_0 = 6$ cm, $y_0 = 2$ cm are saved.

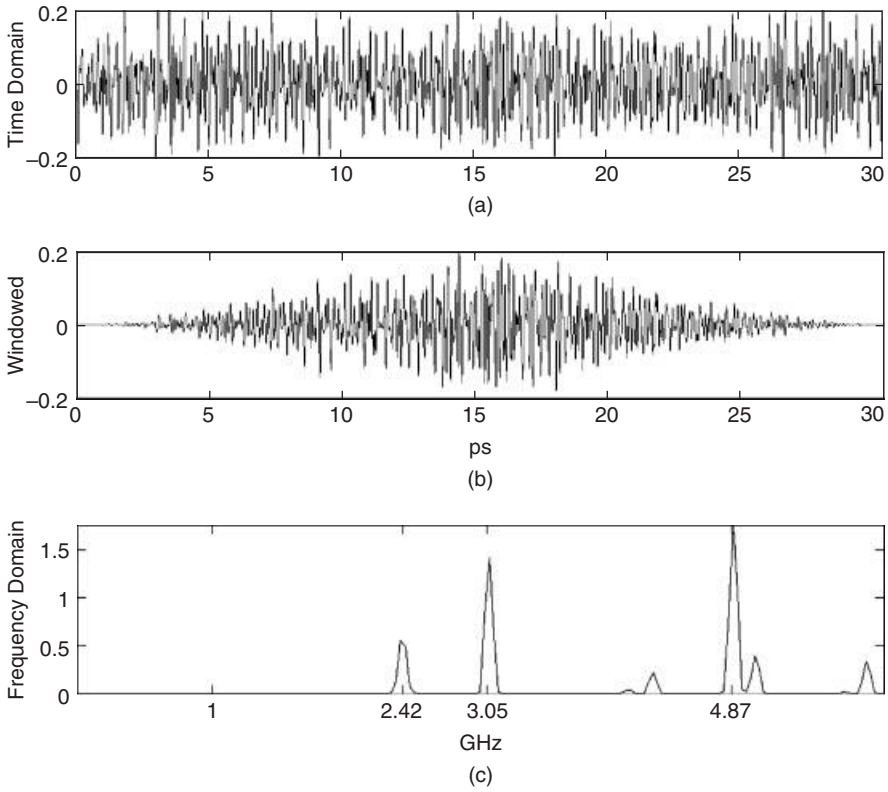

Figure 5.8 (a) The time domain data collected at x_0, y_0, (b) the same data multiplied by the Hanning window, and (c) the Fourier transform of the data from (b).

If we take the Fourier transform of this function, we obtain

$$E_z^0(x_0, y_0, f) = \sum_n d_n \phi_n(x_0, y_0)\delta(f - f_n),$$

that is, a group of delta functions in the frequency domain that reveal the eigenfrequencies! After 20,000 time steps, we obtain the time domain function shown in Fig. 5.8a. This stored data is multiplied by the Hanning window,

$$h(n) = \frac{1}{2} - \frac{1}{2}\cos\left(\frac{2\pi n}{N}\right),$$

where N is the total number of points, in this case, 20,000. The process of windowing removes artifacts when the Fourier transform is taken (8) (Fig. 5.8b). We take the Fourier transform of the windowed data by using the fast Fourier transform. The results are shown in Fig. 5.8c. We assume that the spikes at 2.43,

3.05, and 4.87 GHz correspond to eigenfrequencies. (It could be that the smaller peaks are also eigenfrequencies, but we will just use the biggest peaks for now.)

We now look at the first eigenfrequency f_1 to find the corresponding eigenfunction. Go back to our original function, Eq. (5.24) and take the Fourier Transform at the frequency, f_1:

$$\begin{aligned}\boldsymbol{F}\{E_z^0(x, y, t)\}_{f=f_1} &= \int_{-\infty}^{\infty} \left(\sum_n d_n \phi_n (x, y)\, \mathrm{e}^{j2\pi f_n t} \right) \mathrm{e}^{-jf_1 t} \mathrm{d}t \\ &= \sum_n d_n \phi_n (x, y) \int_{-\infty}^{\infty} \mathrm{e}^{j2\pi (f_n - f_i)t} \mathrm{d}t = d_1 \phi_1 (x, y). \quad (5.25)\end{aligned}$$

This last step results from the fact that

$$\int_{-\infty}^{\infty} \mathrm{e}^{j2\pi (f_n - f_i)t} \mathrm{d}t = \begin{cases} 1 & \text{for } n = i, \\ 0 & \text{otherwise.} \end{cases}$$

Equation (5.25) says that in order to determine $\phi_1(x, y)$ we have to take a Fourier transform at every point, x,y, and then evaluate it at one frequency. That seems like a prohibitive amount of computation. First, we are not going to integrate from $t = -\infty$, but begin at $t = 0$, because that is when an FDTD simulation begins. Second, we are not going to integrate to $t = \infty$, but to a time $T_{\max}$ that is adequate. Finally, we are going to approximate the integral as a summation:

$$\begin{aligned}\boldsymbol{F}\{E_z^0(x, y, t)\}_{f=f_1} &= \int_0^{T_{\max}} E_z^0(x, y, t) \mathrm{e}^{-j2\pi f_1 t} \mathrm{d}t \\ &= \sum_{m=1}^{M_{\max}} E_z^0(x, y, m \times \Delta t) \mathrm{e}^{-j2\pi f_1 m \times \Delta t}. \quad (5.26)\end{aligned}$$

But we have done this before. In Section 2.2, we showed that the discrete Fourier transform at one frequency can be determined by two additional lines of code in Eq. (2.14). Using this method, we can calculate the eigenfunctions corresponding to the three eigenfrequencies of Fig. 5.8c. These are shown in Fig. 5.9.

We can easily verify that these are indeed eigenfunctions of the two-dimensional problem space. We can use one of the eigenfunctions to initialize an FDTD simulation. If it remains stable and returns to its starting position in a time period dictated by

$$T_m = \frac{1}{f_m},$$

then it is an eigenfunction. In Fig. 5.10, the FDTD program has been initialized with E_z values corresponding to the eigenfunction at $f_3 = 4.87$ THz. As the

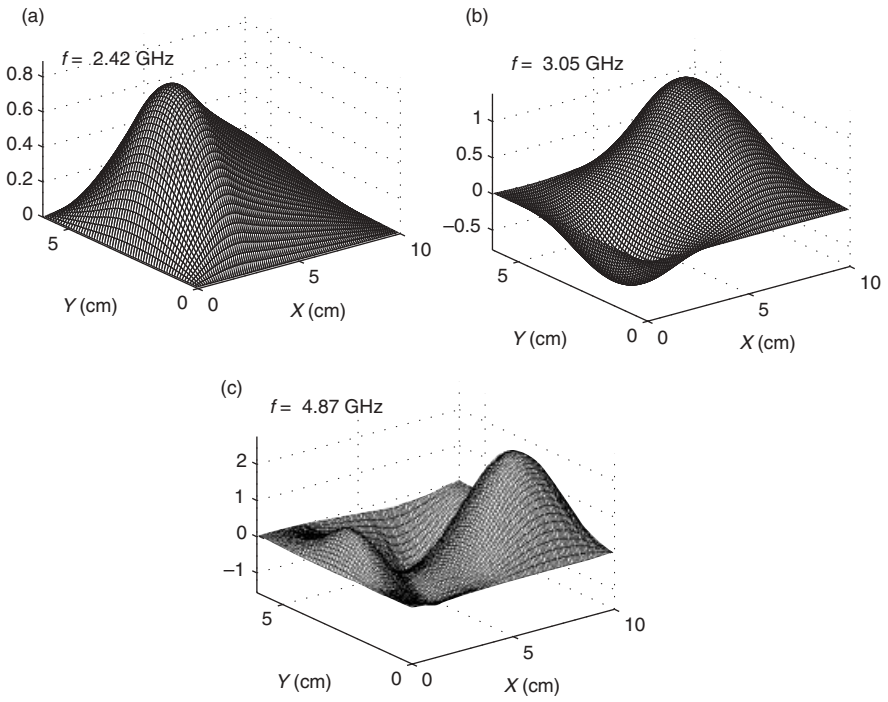

Figure 5.9 The eigenfunctions corresponding to the eigenfrequencies (f) at (a) 2.42, (b) 3.05, and (c) 4.87 GHz.

simulation proceeds, the E_z field oscillates, but by 0.2 ps, the time predicted by

$$T_3 = \frac{1}{4.87 \times 10^{12}\ \mathrm{s}^{-1}} = 0.205 \times 10^{12}\ \mathrm{s},$$

the simulation has returned to its original shape.

The process described in this section has been used to find the eigenfrequencies and corresponding eigenfunctions of the Schrödinger equation in nanoscale quantum structures (9, 10). The same method can be used for three-dimensional structures (11). A systematic method of searching for the eigenfunctions and eigenfrequencies has also been described (12).

PROBLEM SET 5.2

1. Use the program fd2d_eigen.c to determine if one of the smaller peaks in Fig. 5.8c is also an eigenfunction.

2. Repeat the simulation of Fig. 5.7, but start the Gaussian pulse at a different point in the problem space, $x_0 = 4$ cm, $y_0 = 4$ cm. Do you find different eigenfrequencies? If so, explain why.

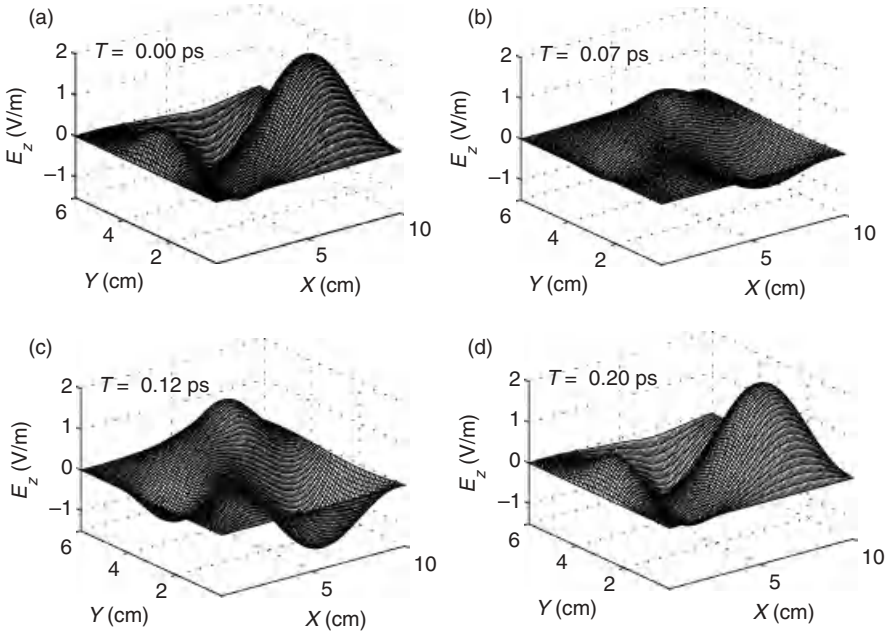

Figure 5.10 In the same FDTD program used to calculate the eigenfrequencies and eigenfunctions, the E_z field is initialized using the eigenfunction corresponding to the eigenfrequency at 4.87 THz. As the simulation proceeds in time, the E_z field oscillates, but it returns to its original state in the time dictated by the eigenfrequency. (a) $T = 0$, (b) $T = 0.07$, (c) $T = 0.12$, and (d) $T = 0.20$ ps.

5.3 SIMULATION OF RF COILS

In this section, the FDTD simulation of a radio frequency (RF) coil of the type shown in Fig. 5.11 is described. These coils are the type used in magnetic resonance imaging (MRI) (13). We will need to model wires with diameters much less than the size of a cell. For this, we use the thin-rod approximation (TRA) (14). It will be shown that FDTD can accurately model the inductance of a metal loop. Furthermore, capacitors in the loop can be modeled using the FDTD method that has already been described.

The basic FDTD equations remain the same as those previously described for three-dimensional simulation in Chapter 4. We will not be using frequency-dependent materials, so in a material with dielectric constant ε_r and conductivity σ, E_z is calculated from D_z as in Eq. (2.8):

$$E_z^n = \frac{D_z^n - I_z^{n-1}}{\varepsilon_r + \frac{\sigma \cdot \Delta t}{\varepsilon_0}} \tag{5.27a}$$

$$I_z^n = I_z^{n-1} + \frac{\sigma \cdot \Delta t}{\varepsilon_0}. \tag{5.27b}$$

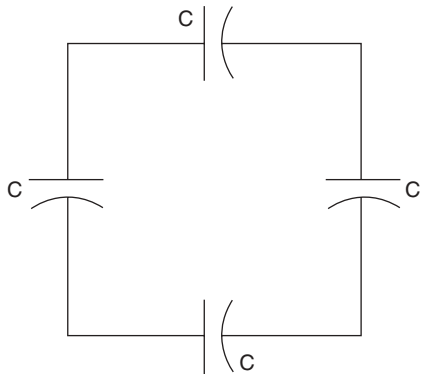

Figure 5.11 An *LC* resonant loop of the type used in MRI. The inductance is provided by the wires.

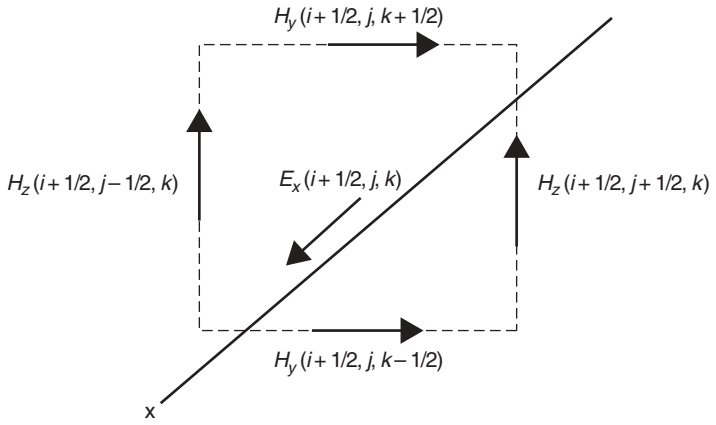

Figure 5.12 The calculation of the current through a wire using Ampere's law as approximated by FDTD in Eq. (5.28).

FDTD only models the E and H fields. Therefore, current flowing through a wire is modeled indirectly through Ampere's law:

$$\oint \boldsymbol{H} \cdot \mathrm{d}l = \boldsymbol{I}.$$

In FDTD, the current through a wire in the x direction is simulated by

$$I_x = \Delta x \cdot \left[\begin{matrix} H_z\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) - H_z\left(i+\frac{1}{2}, j-\frac{1}{2}, k\right) \\ -H_y\left(i+\frac{1}{2}, j, k+\frac{1}{2}\right) + H_y\left(i+\frac{1}{2}, j, k-\frac{1}{2}\right) \end{matrix} \right]. \tag{5.28}$$

This is illustrated in Fig. 5.12.

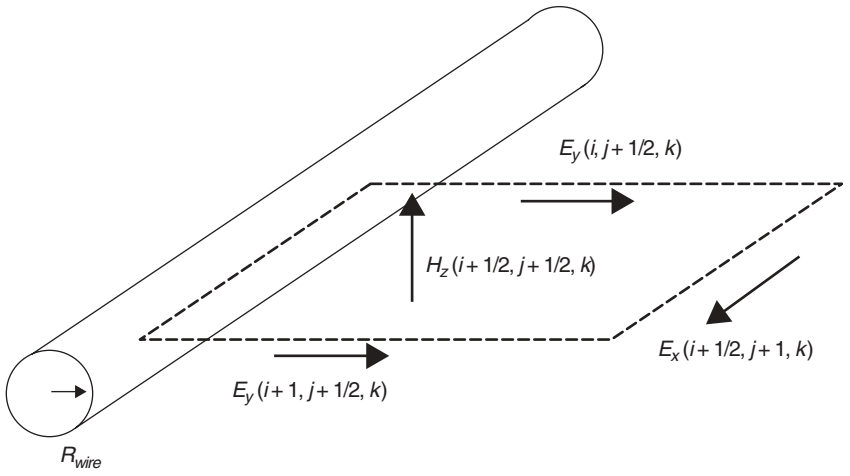

Figure 5.13 The calculation of H_z is modified by the thin-rod approximation (TRA) when the field is in close proximity to a wire.

This by itself does not account for different wire diameters. To simulate wires of different diameters, we employ the TRA (14). This is implemented by an alteration to the calculation of the H fields in close proximity to the wire, as illustrated in Fig. 5.13. For instance, the corresponding equation for H_z is

$$H_z^{n+1}\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) = H_z^n\left(i+\frac{1}{2}, j+\frac{1}{2}, k\right) + \begin{bmatrix} -E_y^{n+1/2}\left(i+1, j+\frac{1}{2}, k\right) + E_y^{n+1/2}\left(i, j+\frac{1}{2}, k\right) \\ -www \cdot E_x^{n+1/2}\left(i+\frac{1}{2}, j+1, k\right) \end{bmatrix}, \tag{5.29}$$

and the factor www is calculated by the TRA

$$www = \left(\frac{\pi}{2}\right)\frac{1}{\ln\left(\dfrac{1}{R_{\text{wire}}}\right)}. \tag{5.30}$$

R_{wire} is the radius of the wire, given as a fraction of the cell size. The details are available in the article by Nadobny et al. (14).

Voltage in FDTD is simulated by the E field of a cell times the length of the cell

$$V = E_x \Delta x. \tag{5.31}$$

In an LC loop, the capacitance is provided by individual capacitors, but the inductance is provided by the wire. Therefore, in order to evaluate the capability

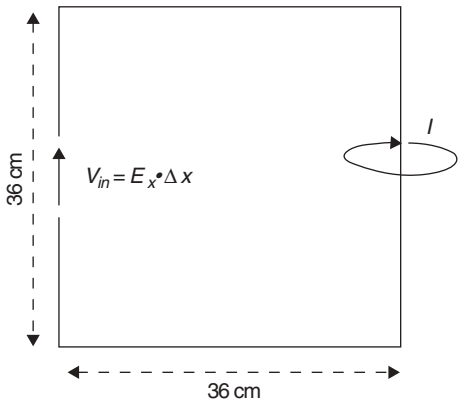

Figure 5.14 Diagram of an inductive rectangular loop. The input V_{in} is generated by an E_x field across one cell. The current is determined by Eq. (5.28) and Eq. (5.29).

of FDTD to correctly model the inductance of the wire, we begin with the simulation of a square wire loop without capacitors, as shown in Fig. 5.14.

The FDTD cells are $1\,\text{cm}^3$. One side of the 36 cm $\times$ 36 cm loop then requires 36 cells. The problem space in the plane of the loop is 60×60 cells. This allows room for a seven-cell PML on each boundary of the problem space. The problem space in the direction perpendicular to the plane is 40 cells. The input to the loop, V_{in} in Fig. 5.14, is a narrow Gaussian pulse. Figure 5.15 shows the H_z fields in the plane of the loop at four different times after 175, 245, 305, and 350 time steps. The time steps are 0.0167 ns. As H_z is one of the fields perpendicular to the wire, it may be regarded as an indication of the current flow.

The inductance is determined through the standard relationship between current and voltage of an inductor:

$$I(t) = \frac{1}{L}\int_0^t v(\tau)\mathrm{d}\tau. \tag{5.32}$$

The current is calculated by Eq. (5.28). The integral of the input voltage is very easily calculated in FDTD by

$$V_{\text{int}}^n = V_{\text{int}}^{n-1} + \Delta t \cdot V_{\text{in}}^n. \tag{5.33}$$

L can be calculated after a sufficient number of time steps n:

$$L = 377\frac{V_{\text{int}}^n}{I^n}. \tag{5.34}$$

The impedance of free space (377) is necessary because we are using the normalized units.

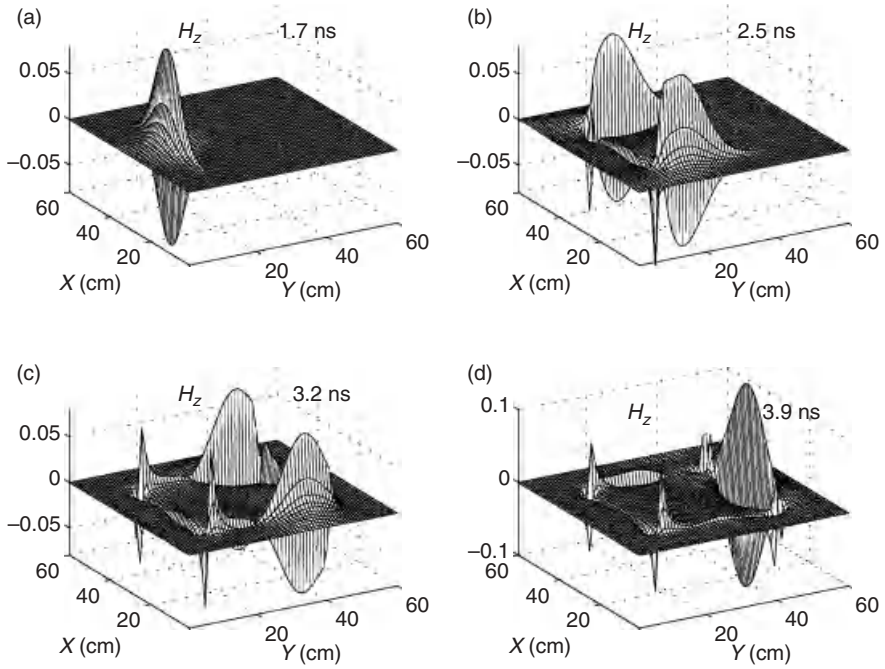

Figure 5.15 The H_z field in the plane of the loop at (a) 1.7, (b) 2.5, (c) 3.2, and (d) 3.9 ns after the input voltage has been applied.

As an example, we simulate the 36 cm $\times$ 36 cm loop of Fig. 5.14 having circular wires that are 0.01 mm in diameter. Equation (5.30) then gives

$$www = \left(\frac{\pi}{2}\right) \frac{1}{\ln\left(\frac{1}{0.005}\right)} = 0.2965.$$

The results are shown in Fig. 5.16. The ringing that appears in the plot of the current is simply due to the time that it takes the pulse to circle the loop. We need the final zero frequency, or direct current (dc), $I(t)$ value that results when the ringing stops. This could potentially take a long time. Instead, we average the last 1000 values of $I(t)$ weighted by the Hanning window, as shown in Fig. 5.17. After 2000 time steps, $I^n = 0.0615$ mA and $V^n_{\text{int}} = 0.0005$ V $-$ μs, and so from Eq. (5.34), $L = 3.07$ μH.

Fortunately, there is an analytic formula available to check the accuracy of this calculation (15). For a one-turn square loop with sides of length a and wire of diameter d, the inductance is given by

$$L_{\text{ana}} = 0.4\left[2d \cdot \ln\left(\frac{4a^2}{d}\right) - 2d\ln(1+\sqrt{2})d\right] + 0.4(2\sqrt{2}d + a - 4d). \quad (5.35)$$

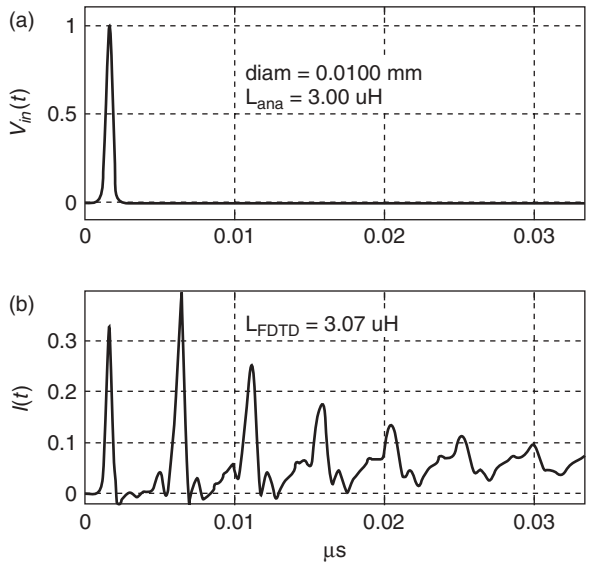

Figure 5.16 Simulation of the inductive loop illustrated in Fig. 5.14. The Gaussian voltage plot that is used as the excitation of the loop (a) and the time domain current as calculated by Eq. (5.28) (b).

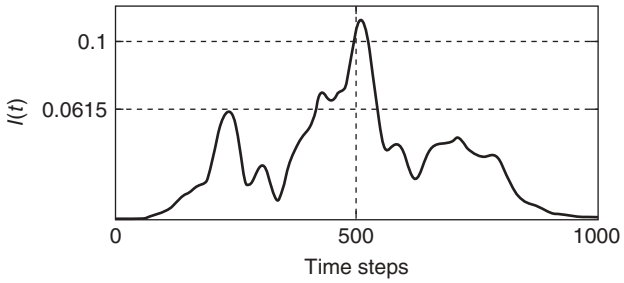

Figure 5.17 The last 1000 time steps of $I(t)$ are weighted by the Hanning window and averaged to get the dc value, rather than wait for all the ringing to stop.

Figure 5.18 shows a comparison of the FDTD-calculated values of inductance for the single-turn rectangular loop of Fig. 5.14 and the analytic values of Eq. (5.35). The comparison is very good over several orders of magnitude.

We now move to the simulation of a resonant *LC* loop, as illustrated in Fig. 5.11. The first task is to add the simulation of the four capacitors. The addition of lumped elements to an FDTD simulation is not new and there are different approaches (16–18). A straightforward approach is used here. A capacitor is added to an FDTD simulation by assuming that one cell in the FDTD grid is a parallel-plate capacitor filled with a material with the relative dielectric

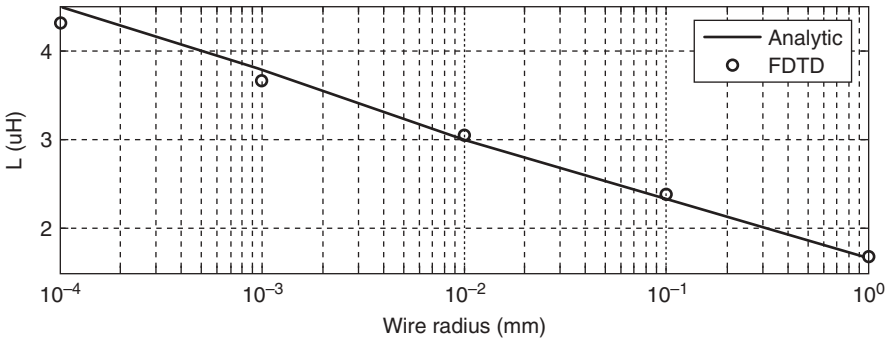

Figure 5.18 Comparison of the FDTD-calculated values of inductance and the analytic values calculated via Eq. (5.39).

constant ε_r. The capacitance is given by

$$C = \varepsilon_r \varepsilon_0 \frac{\text{Area}}{\text{Distance}} = \varepsilon_r \varepsilon_0 \frac{\Delta x^2}{\Delta x} = \varepsilon_r \varepsilon_0 \Delta x. \tag{5.36}$$

Therefore, to specify a capacitance C_1 at an FDTD cell, we set the relative dielectric constant to

$$\varepsilon_r = \frac{C_1}{\varepsilon_0 \Delta x},$$

and calculate the values of `gax` in Eq. (2.10) accordingly. In actuality, Fig. 5.11 is an LC circuit with four capacitors in series. Therefore, if a total capacitance of C_T is desired, C_1 must be set to $C_1 = 4C_T$.

Recall that the goal was to simulate a loop that was resonant at one frequency, say f_0. We know the inductance of the loop for a given wire radius, at least before we added the capacitors. So a good place to begin is

$$C_T = \frac{1}{L(2\pi f_0)^2}. \tag{5.37}$$

We begin with a wire diameter of 0.01 mm whose inductance we have calculated as 3.07 μH. If a resonant frequency of $f_0 = 100$ MHz is desired, then a capacitance of $C_T = 0.844$ pF is required. In fact, the resulting resonance is at about $f_0 = 95$ MHz. This should not be too surprising. We did nothing to allow for the fact that out of the 36 cells that make up each arm of the loop, one has been converted to capacitors. This is bound to have an effect on our original calculation of inductance. Furthermore, our estimation of the capacitance given in Eq. (5.37) must be regarded as a *first-order* approximation. It was found that $C_T = 0.75$ pF results in a resonant frequency of 100 MHz, as shown in Fig. 5.19. The H_z field at four different times during the simulation is shown in Fig. 5.20.

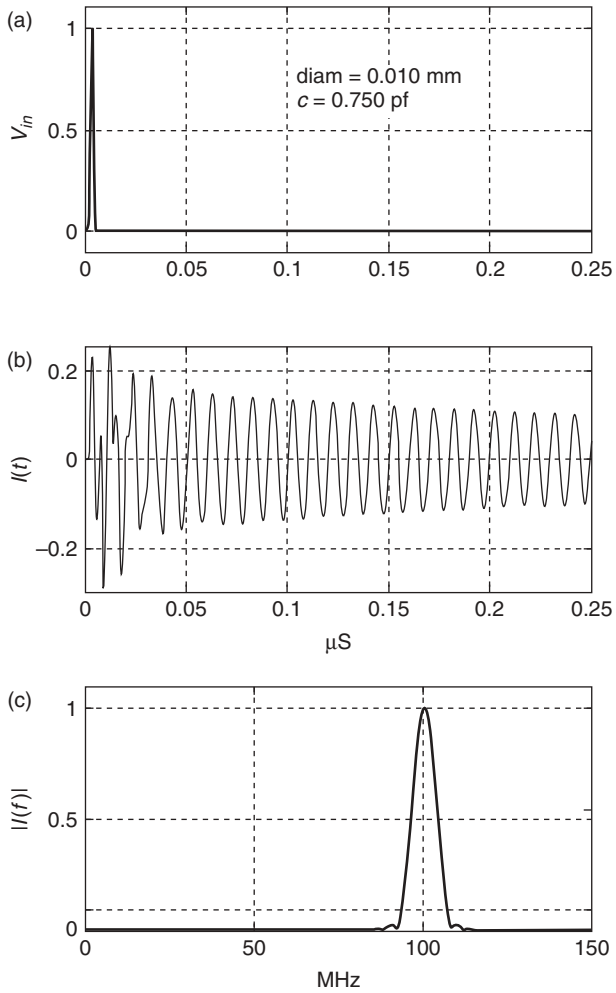

Figure 5.19 The impulse response of an LC coil with $L = 3\ \mu\text{H}$ and $C = 0.75\,\text{pF}$. The Gaussian excitation voltage (a), the time domain plot of the current (b), and the Fourier transform of the current, showing that the resonance is around 100 MHz (c).

The last parameter to be calculated is the resistance of the loop. This is necessary to calculate the quality factor Q of the loop. This will be determined simply by observing the attenuation of the current in the loop, the $I(t)$ plot in Fig. 5.19. In Fig. 5.21, the positive peaks of the current are shown over a limited range to better observe the attenuation. The amplitude decreases from 0.16 to 0.1 A over 0.2 μs. Assuming that the magnitude of the current attenuates as

$$|I(t)| = \mathrm{e}^{-\alpha t},$$

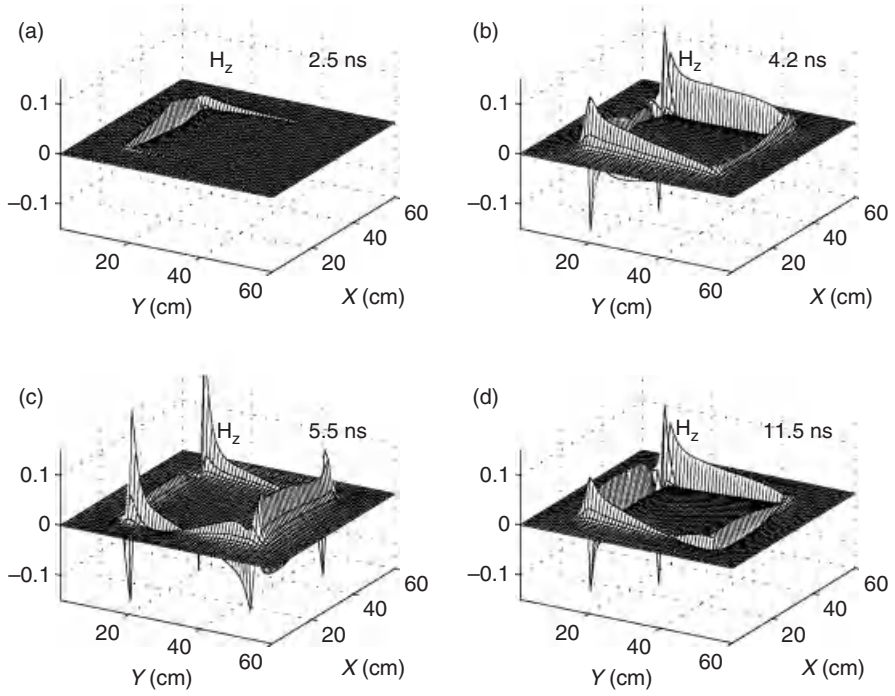

Figure 5.20 The H_z field in the plane of the LC loop at (a) 2.5, (b) 4.2, (c) 5.5, and (d) 11.3 ns after the input voltage has been applied.

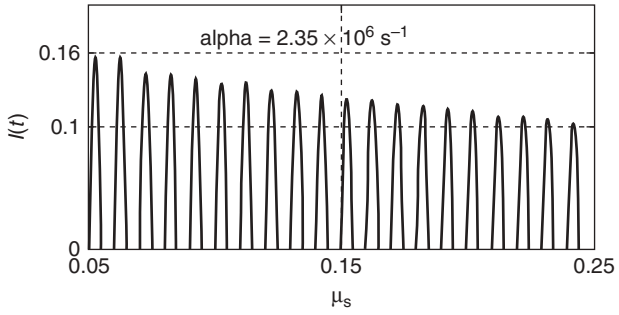

Figure 5.21 The attenuation constant α is determined by observing the attenuation over the given time period using Eq. (5.38).

the attenuation constant α can be determined by

$$\alpha = -\frac{\ln\left(\frac{0.16}{0.1}\right)}{0.2\ \mu\text{s}} = \frac{0.47}{0.2\ \mu\text{s}} = 2.35 \times 10^6 \text{s}^{-1}. \tag{5.38}$$

In a series *RLC* circuit (19),

$$R = 2\alpha L = 2(2.35 \times 10^6 \mathrm{s}^{-1})(3.07 \times 10^{-6}\mathrm{H}) = 14.4\ \Omega.$$

From this, the Q of the circuit is determined by

$$Q = \frac{1}{R}\sqrt{\frac{L}{C}} = \frac{1}{14.4}\sqrt{\frac{3.07 \times 10^{-6}}{0.75 \times 10^{-12}}} = 140.3. \tag{5.39}$$

Remember that the characteristics of the *LC* loop were determined for free space. It is very likely that resonant *LC* loops like the one shown in Fig. 5.11 will be in close proximity to a human body (13). The close proximity of a body will have a significant influence on the characteristics of the loop. In a previous paper by Sullivan et al. (20), the effects of the body were easily modeled into the FDTD simulation to demonstrate the effects on the characteristics of the loop. The energy deposition in the body due to the loop was also simulated. It has been demonstrated in the past that FDTD can be used to accurately model an electromagnetic source of energy and the resulting energy deposition (21). Even models of specific patients can be developed (22). This is particularly useful in hyperthermia cancer therapy simulation (23).

PROBLEM SET 5.3

1. Using the program fd3d_LCloop.f, add a dielectric material with the properties $\varepsilon_r = 30$, $\sigma = 3$, a distance of 5 cm from the plane of the loop. Using the same wire diameter and the same values of capacitance as in the given example, repeat the simulation to see if the dielectric material has caused a shift in the resonant frequency. Tune the value of C to return the resonant frequency to 100 MHz. Has the Q value changed significantly?

REFERENCES

1. R. M. Joseph, S. C. Hagness, and A. Taflove, Direct time integration of Maxwell's equations in linear dispersive media with absorption of scattering and propagation of femtosecond electromagnetic pulses, *Optic. Lett.*, vol. 17, 1991, pp. 1412–1414.
2. P. M. Goorjian and A. Talfove, Direct time integration of Maxwell's equations in non-linear dispersive media with absorption of scattering and propagation of femtosecond electromagnetic pulses, *Optic. Lett.*, vol. 17, 1992, pp. 180–181.
3. P. M. Goorjian, A. Taflove, R. M. Joseph, and S. C. Hagness, Computational modeling of femtosecond optical solitons from Maxwell's equations, *IEEE J. Quant. Electron.*, vol. 28, 1992, pp. 2415–2422.
4. D. M. Sullivan, Nonlinear FDTD formulations using Z transforms, *IEEE Trans. Microw. Theor. Tech.*, vol. 43, March 1995, pp. 676–682.

5. D. M. Sullivan, J. Jiu, and M. Kuzyk, Three-dimensional optical pulse simulation using the FDTD method, *IEEE Trans. Microw. Theor. Tech.*, vol. 48, 2000, pp. 1127–1133.
6. D. K. Cheng, *Fields and Wave Electromagnetics*, Menlow Park, CA: Addison-Wesley, 1992.
7. R. E. Collin, *Field Theory of Guided Waves*, New York, NY: IEEE Press, 1991.
8. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1973.
9. D. M. Sullivan and D. S. Citrin, Determination of the eigenfunctions of arbitrary nanostructures using time domain simulation, *J. Appl. Phys.*, vol. 91, 2002, p. 3219.
10. D. M. Sullivan, *Quantum Mechanics for Electrical Engineers*, New York, NY: IEEE Press, 2012.
11. D. M. Sullivan and D. S. Citrin, Determining quantum eigenfunctions in three-dimensional nanoscale structures, *J. Appl. Phys.*, vol. 97, 2005, p. 104305.
12. D. M. Sullivan and D. S. Citrin, Determining a complete three-dimensional set of quantum eigenfunctions for nanoscale structure analysis, *J. Appl. Phys.*, vol. 98, 2005, p. 084311.
13. J. Juanming, *Electromagnetic Analysis and Design in Magnetic Resonance Imaging*, Boca Raton, FL: CRC Press, 1999.
14. J. Nadobny, R. Pontalti, D. M. Sullivan, W. Wlodarczk, A. Vaccari, P. Deuflhard, and P. Wust, A thin-rod approximation for the improved modeling of bare and insulated cylindrical antennas using the FDTD method, *IEEE Trans. Antenn. Propag.*, vol. 51, August 2003, pp. 1780–1796.
15. F. E. Terman, *Radio Engineers' Handbook*, 1st Edition, London: McGraw-Hill, September 1950.
16. W. Sui, D. A. Christiansen, and D. H. Durney, Extending the two-dimensional FDTD method to hybrid electromagnetic systems with active and passive lumped elements, *IEEE Trans. Microw. Theor. Tech.*, vol. 40, April 1992, pp. 724–730.
17. J. A. Pereda, F. Alimenti, P. Mezzanotte, L. Roselli, and R. Sorrentino, A new algorithm for the incorporation of arbitrary linear lumped networks into FDTD simulators, *IEEE Trans. Microw. Theor. Tech.*, vol. 47, June 1999, pp. 943–949.
18. J. Lee, J. Lee, and H. Jung, Linear lumped loads in the FDTD method using piecewise linear recursive convolution method, *IEEE Microw. Wireless Compon. Lett.*, vol. 16, April 2006, pp. 158–160.
19. C. Paul, *Fundamentals of Electric Circuit Analysis*, New York, NY: John Wiley & Sons, Inc., 2001.
20. D. M. Sullivan, P. Wust, and J. Nadobny, Accurate FDTD simulation of RF coils for MRI using the thin-rod approximation, *IEEE Trans. Antenn. Propag.*, vol. 58, August 2003, pp. 1780–1796.
21. D. M. Sullivan, Three-dimensional computer simulation in deep regional hyperthermia using the FDTD method, *IEEE Trans. Microw. Theor. Tech.*, vol. 38, June 1990, pp. 943–949.
22. B. J. James and D. M. Sullivan, Direct use of CT scans for hyperthermia treatment planning, *IEEE Trans. Biomed. Eng.*, vol. 39, February 1992, pp. 845–851.

23. R. Ben-Yosef and D. M. Sullivan, Peripheral neuropathy and myonecrosis following hyperthermia and radiation therapy for recurrent prostatic cancer: correlation damage with predicted SAR pattern, *Int. J. Hyperther.*, vol. 8, 1992, pp. 173–185.

```
/* FD1D_nlp.c.  Nonlinear pulse */

# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE 20000

main ()
{
    float dx[KE],ex[KE],hy[KE];
    float ga[KE],gb[KE],gc[KE];
    float sx[KE],sxm1[KE],sxm2[KE];
    float snlx[KE],snlxm1[KE],snlxm2[KE];
    int n,m,k,kc,ke,kstart,kend,nsteps;
    float ddx,dt,T,epsz,epsilon,sigma;
    float t0,spread,Amp,pi,pulse;
    FILE *fp, *fopen();
    float ex_low_m1,ex_low_m2,ex_high_m1,ex_high_m2;
    float tau,chi1,del_exp;
    float freq_in;
    float mag,pos,ratio;
    float eps_inf,eps_s,wL,delta_L,alpha_L,beta_L,gamma_L;
    float c1,c2,c3;
    float chi_3, alpha_R, beta_R,delta_NL,wNL,gamma_R,alpha;
    float d1,d2,d3;
    float arg;
    int npml,iT;
    float etime1[100000],etime2[100000];
    float etime3[100000],stime1[100000];

    kc = KE/2;                 /* Center of the space */
    pi  = 3.14159;
    epsz = 8.8e-12;
    ddx = 0.01e-6;               /* Cell size */
    dt = ddx/6e8;              /* Time steps */
       printf(" %6.4f  %10.5e \n",ddx,dt);

    for ( k=0; k < KE; k++ ) {  /* Initialize to free space */
      ga[k] = 1.;
      gb[k] = 0.;
      gc[k] = 0.;
      dx[k] = 0.;
      ex[k] = 0.;
      hy[k] = 0.;
```

```
  sx[k] = 0.;
  sxm1[k] = 0.;
  sxm2[k] = 0.;
  snlx[k] = 0.;
  snlxm1[k] = 0.;
  snlxm2[k] = 0.;
}

/* The Lorentz dispersion */

 eps_inf = 2.25;
 eps_s   = 5.25;
 wL = 2*pi*63.7e12;
 delta_L = 0.00025;

 alpha_L = wL*delta_L;
 beta_L  = wL*sqrt(1 - pow(delta_L,2.0));
 gamma_L = wL*(eps_s - eps_inf)/sqrt(1. - pow(delta_L,2.));

 printf("alpha_L = %12.5e  beta_L = %12.5e ",alpha_L,beta_L);
 printf(" gamma_L = %12.5e \n", gamma_L);

 c1 = 2*exp(-alpha_L*dt)*cos(beta_L*dt);
 c2 = exp(-2*alpha_L*dt);
 c3 = gamma_L*dt*exp(-alpha_L*dt)*sin(beta_L*dt);

 printf("c1 = %8.6f  c2 = %8.6f  c3 = %8.6f \n",c1,c2,c3);

 /* Nonlinear polarization */

  alpha = .7;
  chi_3 = 0.07;

 printf("alpha = %6.3f  chi_3 = %6.3f ",alpha,chi_3);

 /* The Raman Scattering */

  delta_NL = 0.336;
  wNL = 2*pi*14.8e12;

 printf(" delta_NL = %6.3f wNL = %12.5e \n",delta_NL,wNL);

  alpha_R = wNL*delta_NL;
  beta_R  = wNL*sqrt(1. - pow(delta_NL,2.));
  gamma_R = wNL*chi_3*(1. - alpha);

 printf("alpha_R = %12.5e  beta_R = %12.5e ",alpha_R,beta_R);
 printf("  gamma_R = %12.5e \n", gamma_R);
```

```
       d1 = 2*exp(-alpha_R*dt)*cos(beta_R*dt);
       d2 = exp(-2*alpha_R*dt);
       d3 = gamma_R*dt*exp(-alpha_R*dt)*sin(beta_R*dt);

     printf("d1 = %8.6f  d2 = %8.6f  d3 = %12.5e \n",d1,d2,d3);

   /* These parameters specify the input pulse */

       spread = 1000;
       t0 = 3000;
       Amp = .7;
 /* Use Amp = .07 for dispersive, Amp= .7 for NL */
       freq_in = 137;
       printf("%f6.2  THz\n", freq_in);
       freq_in = freq_in*1.e12;
       arg = 2*pi*freq_in*dt;
       printf("%f \n", arg);

    T = 0;
    iT = 0;
    nsteps = 1;

    /* Main part of the program */

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", nsteps);
       printf("%d \n", nsteps);

    for ( n=1; n <=nsteps ; n++)
    {
       T = T + 1;
       iT = iT + 1;

       /* Calculate the Dx field */
       for ( k=0; k < KE; k++ )
       { dx[k] = dx[k] + 0.5*( hy[k-1] - hy[k] ) ; }

       /* Initialize with a pulse */

      pulse = Amp*cos(arg*(T-t0))*exp(-.5*(pow((T-t0)/spread,2.0)));
      dx[1] = eps_inf*pulse;    /* Start the pulse at the low end */

       /* Calculate Ex from Dx */
       pos = 0.;
       mag = 0.;
       sigma = 0.;
       for ( k=1; k < KE-1; k++ )
       {  ex[k] = (dx[k] - sx[k] + chi_3*alpha*2*pow(ex[k],3.) )
```

```
      /(eps_inf + snlx[k] + chi_3*alpha*3*pow(ex[k],2.));
              sx[k] = c1*sxm1[k] - c2*sxm2[k] + c3*ex[k];
              sxm2[k] = sxm1[k];
              sxm1[k] = sx[k];
          snlx[k] = d1*snlxm1[k] - d2*snlxm2[k] + d3*pow(ex[k],2.);
              snlxm2[k] = snlxm1[k];
              snlxm1[k] = snlx[k];
       }

      etime1[iT] = ex[100];
      etime2[iT] = ex[6000];
      etime3[iT] = ex[12000];
      stime1[iT] = sx[100];

     /* Calculate the Hy field */
      for ( k=0; k < KE-1; k++ )
      { hy[k] = hy[k] + .5*( ex[k] - ex[k+1] ) ; }
   }

 /* End of the main loop */

       /* Write the E field out to a file "Ex" */
      fp = fopen( "Ex","w");
      for ( k=0; k < KE; k++ )
      { fprintf( fp,"  %8.5f \n",ex[k]); }
      fclose(fp);

       /* Write the Etime1 to a file "Etime1" */
      fp = fopen( "Etime1","w");
      for ( n=0; n < iT; n++ )
      { fprintf( fp,"  %8.5f \n",etime1[n]); }
      fclose(fp);

       /* Write the Etime2 to a file "Etime2" */
      fp = fopen( "Etime2","w");
      for ( n=0; n < iT; n++ )
      { fprintf( fp,"  %8.5f \n",etime2[n]); }
      fclose(fp);

       /* Write the Etime3 to a file "Etime3" */
      fp = fopen( "Etime3","w");
      for ( n=0; n < iT; n++ )
      { fprintf( fp,"  %8.5f \n",etime3[n]); }
      fclose(fp);

       /* Write the time in (ps) to a file "Time" */
      fp = fopen( "Time","w");
      { fprintf( fp,"  %8.5f \n",1e12*dt*T); }
      fclose(fp);
```

```
       printf( "T = %5.1f pulse = %6.2f  ",T,pulse);
}
}

 /* Fd2d_eigen.c   2D program to find TM eigenfunctions */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>>

#define IE  102
#define JE  62

main ()
{
    float ga[IE][JE],dz[IE][JE],ez[IE][JE],hx[IE][JE],hy[IE][JE];
    float real_pt[IE][JE],imag_pt[IE][JE],amp[IE][JE],phase[IE][JE];
    float eps[IE][JE];
    float han[30000];
    int l,n,i,j,ic,jc,nsteps,Nhan,isource,jsource,III;
    float ddx,dt,T,epsz,pi,epsilon,sigma,eaf;
    float t0,spread,pulse;
    float xdist,ydist,dist;
    float freq,fc,arg,cosarg,sinarg;
    FILE *fp, *fp1, *fp2, *fopen();

    ic = IE/2;
    jc = JE/2;
    ddx = .001;                 /* Cell size */
    dt = ddx/6e8;               /* Time steps */
    epsz = 8.8e-12;
    pi=3.14159;

    for ( j=0; j < JE; j++ ) {
       for ( i=0; i < IE; i++ ) {
           dz[i][j] = 0.;
           ez[i][j] = 0.;
           hx[i][j] = 0.;
           hy[i][j] = 0.;
           ga[i][j]= 1.0  ;
           eps[i][j] = 1.;
           real_pt[i][j] = 0.;
           imag_pt[i][j] = 0.;
    }   }

    /* Specify the dielectric in the waveguide*/

    for ( j=0; j < 40; j++ ) {
       for ( i=0; i < 30; i++ ) {
```

```
          eps[i][j] = 3.0;
          ga[i][j] = 1./eps[i][j];
     }  }
    for ( i=60; i < IE; i++ ) {
        for ( j= 40; j < JE; j++ ) {
          eps[i][j] = 2.0;
          ga[i][j] = 1./eps[i][j];
     }  }

     /* Write the eps field out to a file "eps" */
       fp = fopen( "eps","w");
       for ( j=1; j < JE-1; j++ ) {
          for ( i=1; i < IE-1; i++ ) {
             fprintf( fp,"%6.3f ",eps[i][j]);
          }
             fprintf( fp," \n");
       }
       fclose(fp);

 /* Initialize the test function */

    spread = 7.0;
       for ( j=1; j < JE; j++ ) {
          for ( i=1; i < IE; i++ ) {
             isource = ic+10-i;
             jsource = jc-10-j;
             xdist = isource;
             ydist = jsource;
             dist = sqrt(pow(xdist,2.) + pow(ydist,2.));
             dz[i][j] = exp(-.5*(pow((dist/spread),2.)));
       }  }

  /* Use Nhan = 20000 to find the eigenfrequencies and
         Nhan = 5000 to find the eigenfunctions.        */
       printf( "N Hanning --> ");
       scanf("%d", &Nhan);

       for ( n=1; n < Nhan; n++ ) {
          han[n] = .5*(1 - cos(2*pi*n/Nhan));
       }

       printf( "fc (Ghz) --> ");
       scanf("%f", &fc);
       freq = fc;
       fc = fc*1e9;
       printf("  %f \n", fc);
       arg = 2*pi*dt*fc;
       printf(" arg =  %f \n", arg);
       printf(" 1/arg =  %f \n", 1/arg);
```

```
       fp = fopen( "freq","w");
             fprintf( fp,"%7.3f ",freq);
             fprintf( fp," \n");
       fclose(fp);

    t0 = 30.0;
    T = 0;
    nsteps = 1;

       fp1 = fopen( "Ez_time","w");

while ( nsteps > 0 ) {
       printf( "nsteps --> ");
       scanf("%d", &nsteps);
       printf("%d \n", nsteps);
    for ( n=1; n <=nsteps ; n++)  {
       T = T + 1;

/*  ----   Start of the Main FDTD loop ----  */

       /* Calculate the Dz field */
       for ( j=1; j < JE; j++ ) {
          for ( i=1; i < IE; i++ ) {
              dz[i][j] = dz[i][j]
             + .5*( hy[i][j] - hy[i-1][j] - hx[i][j]
 + hx[i][j-1]) ;
       }  }

       /* Calculate the Ez field */
       /* The field is truncated to simulate metal boundaries */
       for ( j=2; j < JE-1; j++ ) {
          for ( i=2; i < IE-1; i++ ) {
              ez[i][j] = ga[i][j]*dz[i][j] ;
       }  }

             printf( "%5.2f \n",ez[isource][jsource]);

             fprintf( fp1,"%6.3f ",ez[isource][jsource]);
             fprintf( fp1," \n");

        cosarg = cos(arg*T);
        sinarg = cos(arg*T);
       for ( j=2; j < JE-1; j++ ) {
          for ( i=2; i < IE-1; i++ ) {
              real_pt[i][j] = real_pt[i][j] + han[n]*cosarg*ez[i][j];
              imag_pt[i][j] = imag_pt[i][j] + han[n]*sinarg*ez[i][j];
       }  }
```

```
       /* Calculate the Hx field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i < IE-1; i++ ) {
             hx[i][j] = hx[i][j]  + .5*( ez[i][j] - ez[i][j+1] ) ;
       }  }

       /* Calculate the Hy field */
       for ( j=0; j < JE-1; j++ ) {
          for ( i=0; i <= IE-1; i++ ) {
             hy[i][j] = hy[i][j]  + .5*( ez[i+1][j] - ez[i][j] ) ;
     } } }

/*  ----  End of the main FDTD loop ---- */

   /*    for ( j=1; j < jc; j++ ) {
          printf( "%2d  ",j);
          for ( i=1; i < ic; i++ ) {
             printf( "%5.2f ",ez[2*i][2*j]);
          }
             printf( " \n");
       }
       */
     /* Write the E field out to a file "Ez"
      /* Find the magnitude of the constructed function*/
       fp = fopen( "Amp","w");
       fp2 = fopen( "Phase","w");
       for ( j=1; j < JE-1; j++ ) {
          for ( i=1; i < IE-1; i++ ) {
           amp[i][j] = .01*sqrt( pow(real_pt[i][j],2.)
                     + pow(imag_pt[i][j],2.) );
             phase[i][j] = atan2(imag_pt[i][j],real_pt[i][j]);
             fprintf( fp,"%6.3f ",amp[i][j]);
             fprintf( fp2,"%6.3f ",phase[i][j]);
          }
             fprintf( fp," \n");
             fprintf( fp2," \n");
       }
       fclose(fp);
       fclose(fp2);

       fp = fopen( "Amp_rl","w");
       for ( j=1; j < JE-1; j++ ) {
          for ( i=1; i < IE-1; i++ ) {
          amp[i][j] = amp[i][j]*cos(phase[i][j]);
             fprintf( fp,"%7.3f ",amp[i][j]);
          }
             fprintf( fp," \n");
       }
       fclose(fp);
```

```
            printf( "T = %5.0f \n",T);

            fp = fopen("Time","w");
               fprintf(fp,"%10.5f \n",1.e9*dt*T);
            fclose(fp);
               printf("T =  %10.5f ps\n",1e9*dt*T);

      printf("Initialize with the eigenfunction (1 = YES)
 --> ");
      scanf("%d", &III);
      printf("III = %d \n", III);

      if ( III >  0) {
      printf( "Replace E \n");
         for ( j=1; j < JE-1; j++ ) {
            for ( i=1; i < IE-1; i++ ) {
               ez[i][j] = amp[i][j];
               dz[i][j] = ez[i][j]/ga[i][j];
               hx[i][j] = 0.;
               hy[i][j] = 0.;
            }
         }
         T = 0;
      }
 }
      fclose(fp1);

      fp = fopen( "Ez","w");
      for ( j=1; j < JE-1; j++ ) {
         for ( i=1; i < IE-1; i++ ) {
            fprintf( fp,"%6.3f ",ez[i][j]);
         }
            fprintf( fp," \n");
      }
      fclose(fp);

/* Fd3d_LCloop.c  */
/* WARNING:  The chapter describes the loop in the XY plane.
   However, this program constructs a loop in the XZ plane. */

  /*  Specify the loop */

      loop_size = 36;
      lp2 = loop_size/2;

      printf( "wire_diam (mm) --> ");
      scanf("%f", &wire_diam);

      wire_radius = 0.5*(1.e-3*wire_diam/ddx);
      www = (pi/2.)/log(1./wire_radius);
```

```
       for ( k=kc-lp2; k <= kc+lp2-1; k++ ) {
          gaz[ic-lp2][jc][k] = 0.;
          gaz[ic+lp2][jc][k] = 0.;

          wyi[ic-lp2][jc][k] = www;
          wyi[ic-lp2-1][jc][k] = www;
          wxj[ic-lp2][jc][k] = www;
          wxj[ic-lp2][jc-1][k] = www;

          wyi[ic+lp2][jc][k] = www;
          wyi[ic+lp2-1][jc][k] = www;
          wxj[ic+lp2][jc][k] = www;
          wxj[ic+lp2][jc-1][k] = www;
       }

       for ( i=ic-lp2; i <= ic+lp2-1; i++ ) {
          gax[i][jc][kc-lp2] = 0.;
          gax[i][jc][kc+lp2] = 0.;

          wzj[i][jc][kc-lp2] = www;
          wzj[i][jc-1][kc-lp2] = www;
          wyk[i][jc][kc-lp2] = www;
          wyk[i][jc][kc-lp2-1] = www;

          wzj[i][jc][kc+lp2] = www;
          wzj[i][jc-1][kc+lp2] = www;
          wyk[i][jc][kc+lp2] = www;
          wyk[i][jc][kc+lp2-1] = www;
       }

    /* Add the capacitors */

       printf( " C total (pf)--> ");
       scanf("%f", &c_tot);

       eps_c = 4.*c_tot*1e-12/(epsz*ddx);

       gaz[ic-lp2][jc][kc] = 1/eps_c;
       gaz[ic+lp2][jc][kc] = 1/eps_c;
       gax[ic][jc][kc-lp2] = 1/eps_c;
       gax[ic][jc][kc+lp2] = 1/eps_c;


/*  ----   Start of the Main FDTD loop ----  */

       /* Calculate the Dx field */

      for ( i=1; i < IE; i++ ) {
         for ( j=1; j < JE; j++ ) {
            for ( k=1; k < KE; k++ ) {
```

```
            curl_h = ( hz[i][j][k] - hz[i][j-1][k]
                     - hy[i][j][k] + hy[i][j][k-1]) ;
            idx[i][j][k] = idx[i][j][k] + gi0[i]*curl_h;
            dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
          + gj2[j]*gk2[k]*.5*(curl_h + gi1[i]*idx[i][j][k]);
 }  }  }

  /* Calculate the Dy field */

  for ( i=1; i < IE; i++ ) {
     for ( j=1; j < JE; j++ ) {
        for ( k=1; k < KE; k++ ) {
           curl_h = ( hx[i][j][k] - hx[i][j][k-1]
                    - hz[i][j][k] + hz[i-1][j][k]) ;
           idy[i][j][k] = idy[i][j][k] + gj0[j]*curl_h;
           dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
           + gi2[i]*gk2[k]*.5*( curl_h + gj1[j]*idy[i][j][k]);
 }  }  }

  /* Calculate the Dz field */

  for ( i=1; i < IE; i++ ) {
     for ( j=1; j < JE; j++ ) {
        for ( k=0; k < KE; k++ ) {
           curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                    - hx[i][j][k] + hx[i][j-1][k]) ;
           idz[i][j][k] = idz[i][j][k] + gk0[k]*curl_h;
           dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
           + .5*gi2[i]*gj2[j]*( curl_h + gk1[k]*idz[i][j][k] );
 }  }  }

  /* Calculate the E from D field */
  for ( i=1; i < IE-1; i++ ) {
     for ( j=1; j < JE-1; j++ ) {
        for ( k=1; k < KE-1; k++ ) {
        ex[i][j][k] = gax[i][j][k]*dx[i][j][k] ;
        ey[i][j][k] = gay[i][j][k]*dy[i][j][k] ;
        ez[i][j][k] = gaz[i][j][k]*dz[i][j][k] ;
  }  }  }

/*  Source */

  i = ic-lp2;
  j = jc;
  k = kc+5;      /* Moved to accomodate the capacitor */


  pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
  ez[i][j][k] = pulse;
```

```
       Vtime[iT] = ez[i][j][k];

       k = kc;
       H_current = ( hy[i][j][k] - hy[i-1][j][k]
                   - hx[i][j][k] + hx[i][j-1][k]) ;
       Itime[iT] = H_current;

/* Calculate the Hx field */

      for ( i=0; i < IE; i++ ) {
         for ( j=0; j < JE-1; j++ ) {
            for ( k=0; k < KE-1; k++ ) {
            curl_e =              (  ey[i][j][k+1] - ey[i][j][k] )
                   + wxj[i][j][k]*( -ez[i][j+1][k] + ez[i][j][k] );
                ihx[i][j][k] = ihx[i][j][k]  + fi0[i]*curl_e;
                 hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
                + fj2[j]*fk2[k]*.5*( curl_e + fi1[i]*ihx[i][j][k] );
      }  }  }

       /* Calculate the Hy field */

       for ( i=0; i < IE-1; i++ ) {
          for ( j=0; j < JE; j++ ) {
             for ( k=0; k < KE-1; k++ ) {
             curl_e = wyi[i][j][k]*(  ez[i+1][j][k] - ez[i][j][k] )
                    + wyk[i][j][k]*( -ex[i][j][k+1] + ex[i][j][k] );
                 ihy[i][j][k] = ihy[i][j][k] + fj0[j]*curl_e ;
                 hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
                + fi2[i]*fk3[k]*.5*( curl_e + fj1[j]*ihy[i][j][k] );
      }  }  }

       /* Calculate the Hz field */

       for ( i=0; i < IE-1; i++ ) {
          for ( j=0; j < JE-1; j++ ) {
             for ( k=0; k < KE; k++ ) {
             curl_e = wzj[i][j][k]*(  ex[i][j+1][k] - ex[i][j][k] )
                    +              ( -ey[i+1][j][k] + ey[i][j][k] );
                ihz[i][j][k] = ihz[i][j][k] + fk0[k]*curl_e;
                 hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
                + .5*fi2[i]*fj2[j]*( curl_e + fk1[k]*ihz[i][j][k] );
      }  }  }
    }
/*  ----  End of the main FDTD loop ---- */

   /* Write out Vtime */

        fp = fopen( "Vtime","w");
```

```
   for ( i=0; i < iT; i++ ) {
     fprintf( fp,"%8.5f \n",Vtime[i]);
   }
fclose(fp);

/* Write out Itime */

fp = fopen( "Itime","w");
   for ( i=0; i < iT; i++ ) {
     fprintf( fp,"%10.7f \n",Itime[i]);
   }
fclose(fp);
```

6

QUANTUM SIMULATION

The Schrödinger equation (6.1) is at the heart of quantum mechanics. This chapter provides a brief introduction to quantum simulation using FDTD. The same basic methods used to simulate Maxwell's equations can be used to simulate the Schrödinger equation (6.2).

6.1 SIMULATION OF THE ONE-DIMENSIONAL, TIME-DEPENDENT SCHRÖDINGER EQUATION

The time-dependent Schrödinger equation is

$$i\hbar\frac{\partial\psi(x,t)}{\partial t} = -\frac{\hbar^2}{2m_e}\nabla^2\psi(x,t) + V(x)\psi(x,t). \tag{6.1}$$

The parameter $\psi(x,t)$ is a state variable. It has no direct physical meaning, but all relevant physical parameters can be determined from it. In general, $\psi(x,t)$ is a function of both space and time, and its values are complex. $V(x)$ is the potential and has the unit of energy, which we usually describe in electronvolts (eV) for our applications ($1\,\text{eV} = 1.6 \times 10^{-19}\text{J}$). $\hbar$ is Planck's constant ($\hbar = 1.054 \times 10^{-34}\text{J s} = 6.58 \times 10^{-16}\text{eV s}$). m_e is the mass of the particle being represented by the Schrödinger equation. For this discussion, we will assume the mass of an

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

electron, which is $m_e = 9.109 \times 10^{-31}$kg.We begin by rewriting the Schrödinger equation in one dimension as

$$\frac{\partial \psi}{\partial t} = i\frac{\hbar}{2m_e}\frac{\partial^2 \psi(x,t)}{\partial x^2} - \frac{i}{\hbar}V(x)\psi(x,t). \tag{6.2}$$

To avoid using complex numbers, we will split $\psi(x,t)$ into its real and imaginary components:

$$\psi(x,t) = \psi_{\text{real}}(x,t) + i \cdot \psi_{\text{imag}}(x,t).$$

Inserting this into Eq. (6.2) and separating into the real and imaginary parts results in two coupled equations:

$$\frac{\partial \psi_{\text{real}}(x,t)}{\partial t} = -\frac{\hbar}{2m_e}\frac{\partial^2 \psi_{\text{imag}}(x,t)}{\partial x^2} + \frac{1}{\hbar}V(x)\psi_{\text{imag}}(x,t) \tag{6.3a}$$

$$\frac{\partial \psi_{\text{imag}}(x,t)}{\partial t} = \frac{\hbar}{2m_e}\frac{\partial^2 \psi_{\text{real}}(x,t)}{\partial x^2} - \frac{1}{\hbar}V(x)\psi_{\text{real}}(x,t). \tag{6.3b}$$

To enter these equations into a computer, we will use the finite-difference approximations. The time derivative is approximated by

$$\frac{\partial \psi_{\text{real}}(x,t)}{\partial t} \cong \frac{\psi_{\text{real}}(x,(m+1)\cdot\Delta t) - \psi_{\text{real}}(x,(m)\cdot\Delta t)}{\Delta t}, \tag{6.4a}$$

where Δt is the time step. The Laplacian is approximated by

$$\frac{\partial^2 \psi_{\text{imag}}(x,t)}{\partial x^2}$$
$$\cong \frac{\psi_{\text{imag}}(\Delta x\cdot(n+1), m\cdot\Delta t) - 2\psi_{\text{imag}}(\Delta x\cdot n, m\cdot\Delta t) + \psi_{\text{imag}}(\Delta x\cdot(n-1), m\cdot\Delta t)}{(\Delta x)^2}, \tag{6.4b}$$

where Δx is the size of the cells being used for the simulation. For simplicity, we will use the following notation

$$\psi(n\cdot\Delta x, m\cdot\Delta t) = \psi^m(n), \tag{6.5}$$

where the superscript m indicates the time in units of time steps ($t = m\cdot\Delta t$) and n indicates the position in units of cells ($x = n\cdot\Delta x$). Now Eq. (6.3a) can be written as

$$\frac{\psi_{\text{real}}^{m+1}(n) - \psi_{\text{real}}^{m}(n)}{\Delta t} = -\frac{\hbar}{2m}\frac{\psi_{\text{imag}}^{m+1/2}(n+1) - 2\psi_{\text{imag}}^{m+1/2}(n) + \psi_{\text{imag}}^{m+1/2}(n-1)}{(\Delta x)^2} + \frac{1}{\hbar}V(n)\psi_{\text{imag}}^{m+1/2}(n),$$

which we can rewrite as

$$\psi_{\text{real}}^{m+1}(n) = \psi_{\text{real}}^{m}(n) - \frac{\hbar}{2m_{\text{e}}}\frac{\Delta t}{(\Delta x)^2}[\psi_{\text{imag}}^{m+1/2}(n+1) - 2\psi_{\text{imag}}^{m+1/2}(n) + \psi_{\text{imag}}^{m+1/2}(n-1)] + \frac{\Delta t}{\hbar}V(n)\psi_{\text{imag}}^{m+1/2}(n). \tag{6.6a}$$

A similar procedure converts the imaginary part to the same form

$$\psi_{\text{imag}}^{m+3/2}(n) = \psi_{\text{imag}}^{m+1/2}(n) + \frac{\hbar}{2m_{\text{e}}}\frac{\Delta t}{(\Delta x)^2}[\psi_{\text{real}}^{m+1}(n+1) - 2\psi_{\text{real}}^{m+1}(n) + \psi_{\text{real}}^{m+1}(n-1)] - \frac{\Delta t}{\hbar}V(n)\psi_{\text{real}}^{m+1}(n). \tag{6.6b}$$

According to Eq. (6.6a) and (6.6b), we can get the value of ψ at time $(m+1)\Delta t$ from the previous value and the surrounding values. Notice that the real values of ψ in Eq. (6.6a) are calculated at integer values of m, while the imaginary values of ψ are calculated at the half integer values of m. This is the leapfrogging technique between the real and imaginary terms similar to the E and H fields in FDTD EM simulation.

We will take the cell size as $\Delta x = 0.2$ nm. We still have to choose Δt. Look at Eq. (6.6). We will define a new parameter to combine all the terms in front of the parentheses

$$ra \equiv \frac{\hbar}{2m_{\text{e}}}\frac{\Delta t}{(\Delta x)^2}. \tag{6.7}$$

To maintain stability, this term must be small, no greater than approximately 0.15. All of the terms in Eq. (6.7) have been specified except Δt. Actually, Δt must also be small enough so that the term $\Delta t/\hbar[V(n)]$ is also fairly small. We still set ra = 1/8 and calculate Δt from Eq. (6.7).

Figure 6.1 illustrates the simulation of an electron in a potential of $V(x) = 0$ eV in close proximity to an area with a potential of $V(x) = 0.2$ eV. A jump in potential like this is the type of interface a particle might encounter at the junction of two semiconductors that are doped differently (3). The particle is traveling to the right in the positive x direction. This is indicated by the fact that the imaginary part of $\psi(x)$, the dashed line, leads the real part, the solid line. The simulation is initialized at time $T = 0$. The amplitude of $\psi(x)$ is determined by *normalization*,

$$\int_{-\infty}^{\infty} \psi^*(x)\psi(x)\mathrm{d}x = 1. \tag{6.8}$$

This basically states that the probability of the particle being somewhere is 1.

Notice that the kinetic energy is 0.365 eV. The particle has no potential energy, because it is at zero potential. After 200 iterations, which represents a time of

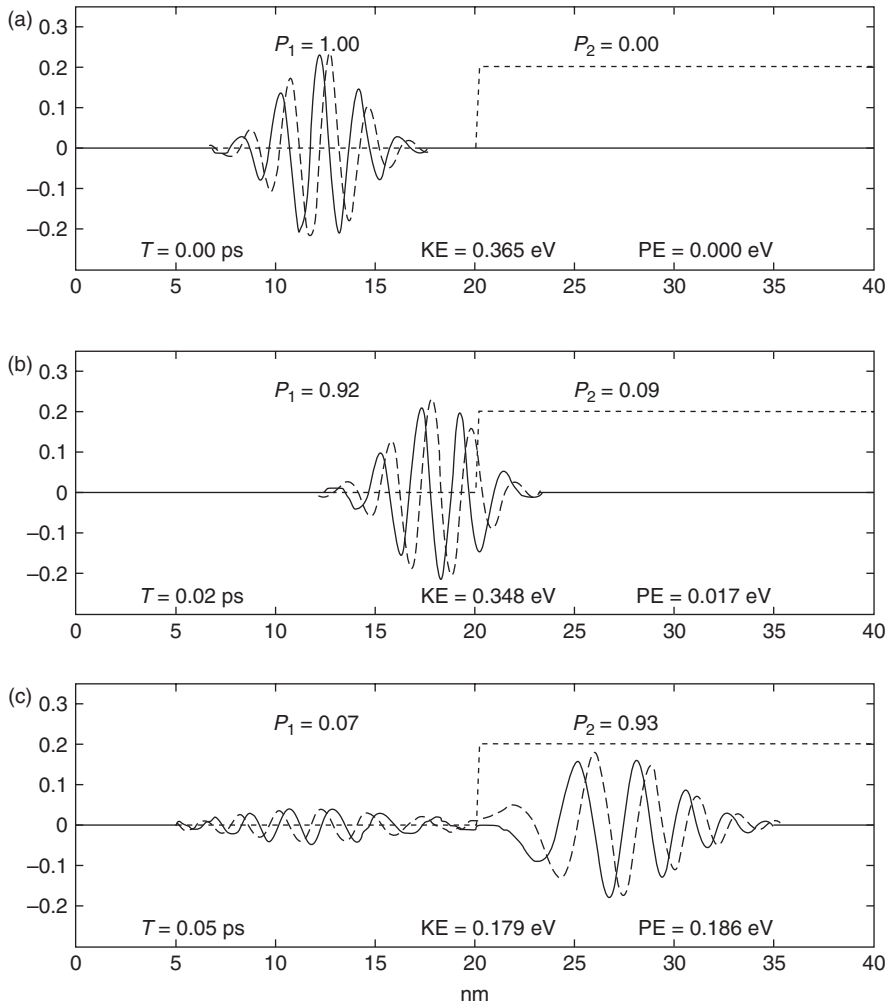

Figure 6.1 (a,b,c) A particle propagating in free space. The solid line represents the real part of ψ and the dashed line represents the imaginary part.

0.02 ps, we see the electron has moved about 5 nm and has begun to interact with the potential. By $T = 0.5$ ps, part of the particle has entered the potential and part of it is moving in the negative direction. This does not mean that the particle has split into two, but that there exists a probability of the particle being reflected by the potential, as well as a probability that it is transmitted into the potential. The respective probabilities are determined by

$$P_1 = \int_{0\,\mathrm{nm}}^{20\,\mathrm{nm}} \psi^*(x)\psi(x)\mathrm{d}x$$

$$P_2 = \int_{20\,\text{nm}}^{40\,\text{nm}} \psi^*(x)\psi(x)\mathrm{d}x.$$

Note also that some of the kinetic energy has been exchanged for potential energy because part of the waveform is in the area with the potential. The total energy remains the same because no energy has been put into or taken out of the system.

Figure 6.1 indicates that the particle being simulated has 0.365 eV of kinetic energy. In quantum mechanics, momentum is related to wavelength through Planck's constant

$$p = \frac{h}{\lambda}. \tag{6.9}$$

(h is another version of Planck's constant. The versions are related by $h = 2\pi\hbar$.) We can calculate kinetic energy by

$$\text{KE} = \frac{1}{2}m_\text{e}v^2 = \frac{p^2}{2m_\text{e}} = \frac{1}{2m_\text{e}}\left(\frac{h}{\lambda}\right)^2. \tag{6.10}$$

In Fig. 6.1, the wavelength appears to be 2 nm. The mass of an electron in free space is 9.1×10^{-31} kg.

$$\begin{aligned}\text{KE} &= \frac{1}{2(9.1\times10^{-31}\text{kg})}\left(\frac{6.625\times10^{-34}\text{Js}}{2\times10^{-9}\text{m}}\right)^2 \\ &= \frac{1}{2(9.1\times10^{-31}\text{kg})}(1.656\times10^{-24}\text{Js/m})^2 = \frac{4.39\times10^{-50}\text{J}^2\text{s}^2}{18.2\times10^{-31}\text{kg m}^2} \\ &= 6.03\times10^{-20}\text{J}\left(\frac{1\ \text{eV}}{1.6\times10^{-19}\text{J}}\right) = 0.379\ \text{eV}.\end{aligned}$$

The FDTD program actually calculates the kinetic energy as the expectation value of the kinetic energy operator. This kinetic energy operator is

$$\text{KE} = -\frac{\hbar^2}{2m_\text{e}}\frac{\partial^2}{\partial x^2}.$$

The expectation value of the kinetic energy operator is

$$\langle\text{KE}\rangle = \int_{-\infty}^{\infty}\psi^*(x)\left[-\frac{\hbar^2}{2m_\text{e}}\frac{\partial^2}{\partial x^2}\psi\,(x)\right]\mathrm{d}x.$$

The potential energy is calculated according to the fraction of the energy that is at the higher potential

$$\text{PE} = \int_{20\,\text{nm}}^{40\,\text{nm}}\psi^*(x)\psi(x)V(x)\mathrm{d}x.$$

We know that the fraction of the particle transmitted into the potential of 0.2 eV is 0.93, so

$$\text{PE} = 0.93 \times 0.2 \text{ eV} = 0.186 \text{ eV}.$$

Details about the expectation values of operators are given in Reference 1. A description of how they are calculated in FDTD is provided in Reference 2.

It is interesting to stop and consider what this simulation tells us. A particle starts by moving left to right with a kinetic energy of 0.365 eV. After the collision with the 0.2-eV potential, there is a 7% chance that the particle is reflected. If that is the case, the reflected particle must also have 0.365 eV of kinetic energy. There is a 93% chance that the particle is transmitted into the potential. If that is the case, this transmitted particle will have a potential of 0.2 eV because it will be located at that potential. It will also have a kinetic energy of 0.365 − 0.2 eV = 0.165 eV because energy must be conserved.

More details on FDTD simulation of the Schrödinger equation are given in the book by Sullivan (4).

PROBLEM SET 6.1

1. Using the program se1d.c, repeat the results of Fig. 6.1. Change the potential to 4 eV. What happens?

6.2 TUNNELING

Tunneling is a purely quantum mechanical phenomenon. It is the ability of a particle such as an electron to pass through an energy barrier that would be impossible in classical physics (Fig. 6.2). Once again we have initialized a waveform representing that of an electron with a kinetic energy of 0.365 eV around 10 nm. Between 18 and 22 nm is a barrier with a potential of 0.4 eV. By 0.3 ps, the particle has reached the barrier and part of the waveform is in the barrier. Notice that it attenuates very rapidly. This is characteristic of a particle in a potential of higher energy (1). However, by 0.5 ps, most of the waveform has been reflected from the barrier, but about 19% has managed to tunnel through. Notice that the portion of the waveform that has tunneled through is at the same wavelength as the original waveform, which means it must have the same kinetic energy. Tunneling plays a significant role in many modern electronic devices (5–8).

PROBLEM SET 6.2

1. Still using the program se1d.c, repeat the results of the tunneling in Fig. 6.2. Make the barrier half as wide. What is the difference?

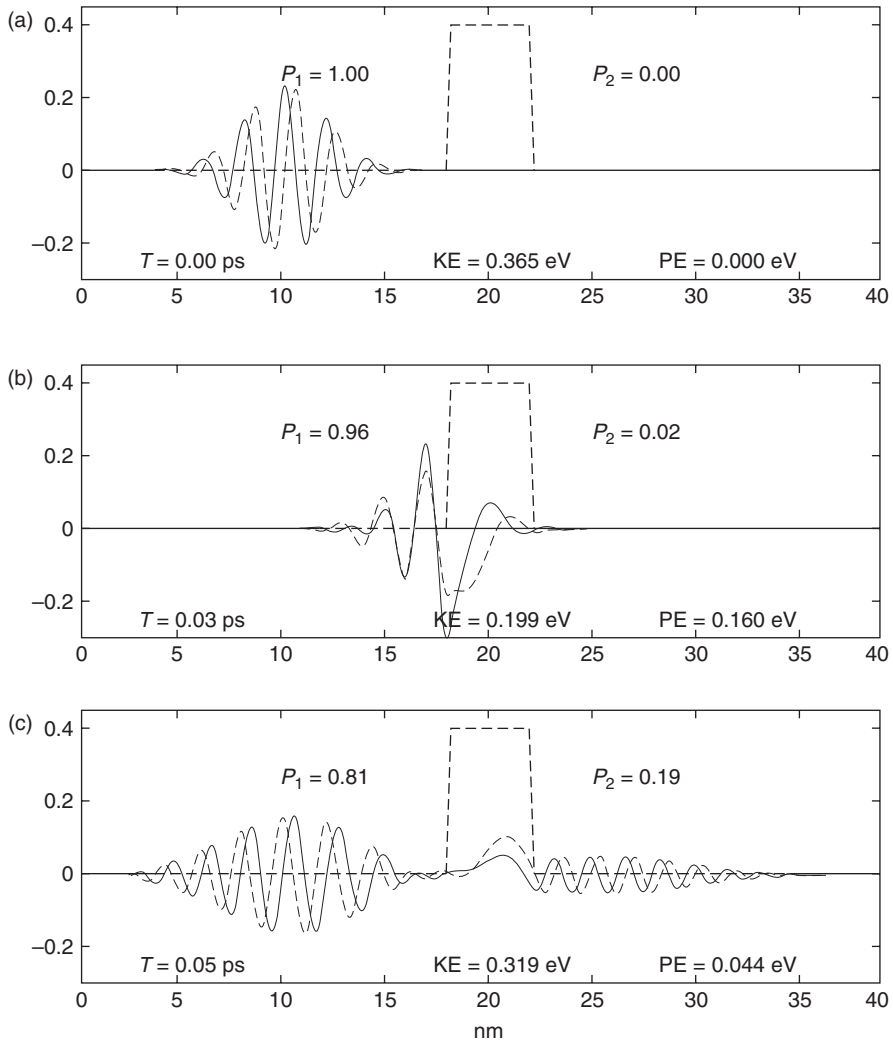

Figure 6.2 (a,b,c) A particle tunneling through a barrier.

6.3 WHY SEMICONDUCTORS HAVE ENERGY BANDS

Semiconductors are crystals. This means that the atoms that compose the semiconductors are in highly specific patterns. An example is given in Fig. 6.3a. A particle traveling in a certain direction in this semiconductor is bound to see a periodic change in the potential due to coulomb interactions with the atoms of the crystal. So if a particle was traveling in the direction shown by the arrow in Fig. 6.3a, it may see a periodic potential like the one shown if Fig. 6.3b.

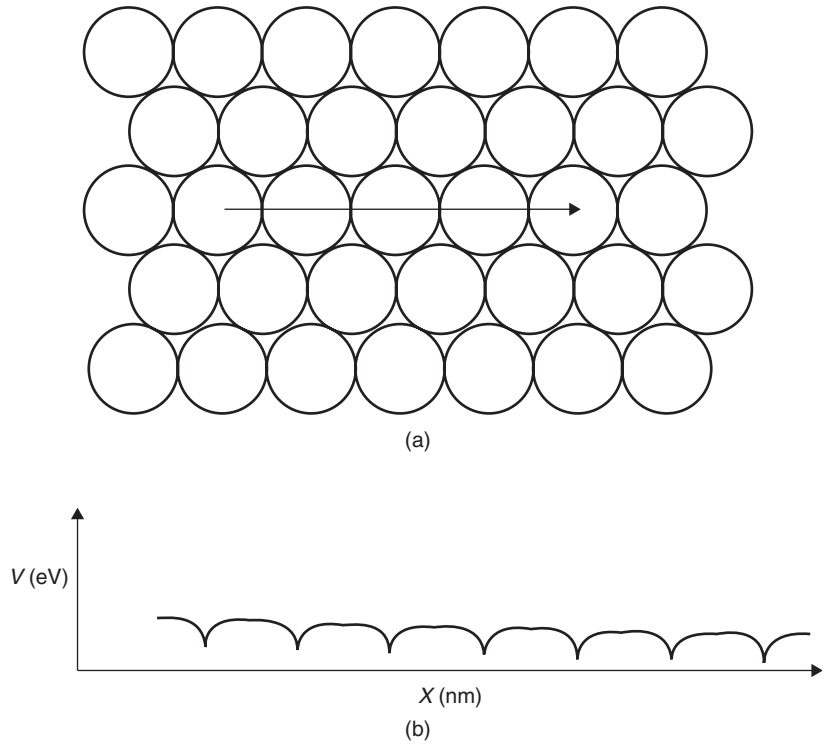

Figure 6.3 (a) An illustration of how atoms might be arranged in a crystal. (b) The potential seen by a particle moving along the direction indicated by the arrow in (a).

Figure 6.4 shows the waveform of an electron in a periodic potential. This potential consists of spikes of amplitude −0.1 eV spaced at intervals of 2 nm. This is intended to mimic the periodic potential in a semiconductor crystal. The waveform has a center wavelength of 2.5 nm and a kinetic energy of 0.239 eV. Classically, one would expect that this particle would just skip over this periodic potential. As shown in Fig. 6.4b at 0.08 ps, this is nearly what happens.

Figure 6.5 shows a similar simulation for a particle with a wavelength of 2 nm and a higher kinetic energy of 0.371 eV. However, when this particle starts to move in the potential, it is clearly leaving part of the waveform behind. Obviously, this particle cannot propagate over any substantial distance. To get a feel of what is happening, consider Fig. 6.6.

Figure 6.6 is a diagram of one interval in the periodic potential that we are using to model a semiconductor. Any wavelength related to the interval d by the equation

$$\lambda_n = \left(\frac{d}{2}\right) n, \quad n = 1, 2, 3, \ldots \tag{6.11}$$

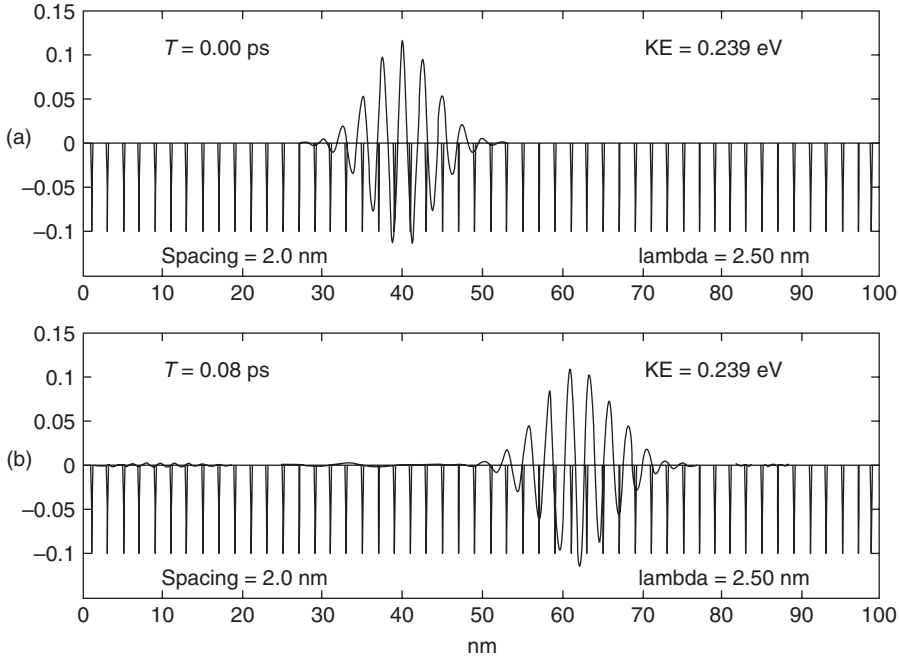

Figure 6.4 (a,b) A particle with wavelength of 2.5 nm is initialized in a periodic lattice. It apparently propagates with no interference from the lattice.

will tend to stay between the potential spikes of this one interval. This will happen at every interval. In other words, the waveform will lose some amplitude as it goes through the interval.

We learned earlier that kinetic energy is related to wavelength by Eq. (6.10).So if Eq. (6.11) gives the forbidden wavelengths, then the forbidden energies are

$$E_n = \frac{h^2}{2m_e\left[\left(\frac{d}{2}\right)n\right]^2} = \frac{2h^2}{m_e d^2}\frac{1}{n^2}. \tag{6.12}$$

Any electron trying to propagate through the lattice must have a kinetic energy between the forbidden energies given by Eq. (6.12) (1, 5). Classically, these allowed energy bands are often calculated by the Kronig–Penney model (3).

More detailed analysis of the use of FDTD simulation to determine forbidden and allowed bands is provided in the book by Sullivan (4).

PROBLEM SET 6.3

1. Equation (6.11) indicates that the smallest forbidden wavelength is $\lambda_1 = d/2$. If $d = 2$ nm, as in Fig. 6.5, to what energy does $\lambda_1 = d/2$ correspond? (Hint: you can deduct this from Fig. 6.5 without using a lot of math.)

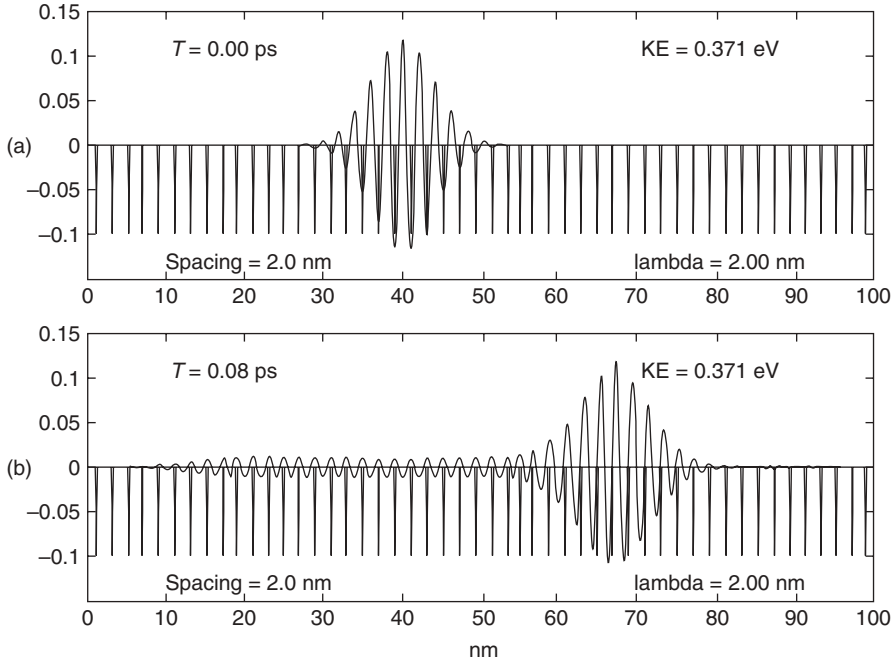

Figure 6.5 (a,b) A particle with a wavelength of 2 nm is initialized in a periodic lattice. Apparently, it cannot propagate because it is leaving part of the waveform behind.

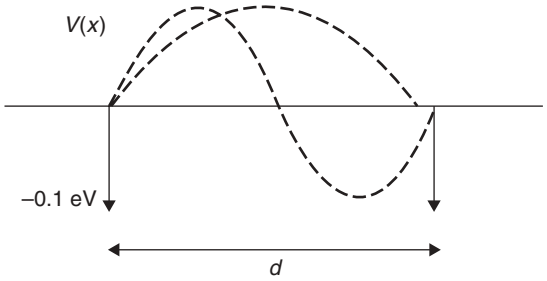

Figure 6.6 One interval in the periodic potential. Any waveform with an integer half-wavelength will tend to stay between the potential spikes.

2. Using the program se_lat.c, repeat the simulation of Fig. 6.3 using $\lambda_1 = 4$ nm. What happens? Change to $\lambda = 6$ nm. What happens?

REFERENCES

1. D. J. Griffiths, *Introduction to Quantum Mechanics*, Englewood Cliffs, NJ: Prentice Hall, 1994.

2. D. M. Sullivan and D. S. Citrin, Time-domain simulation of two electrons in a quantum dot, *J. Appl. Phys.*, vol. 89, pp. 3841–3846, April 2001.
3. D. A. Neamen, *Semiconductor Physics and Devices—Basic Principles*, 3rd Edition, New York, NY: McGraw-Hill, 2003.
4. D. M. Sullivan, *Quantum Mechanics for Electrical Engineers*, New, York, NY: IEEE Press, 2012.
5. S. Datta, *Quantum Transport—Atom to Transistor*, Cambridge, UK: Cambridge University Press, 2005.
6. G. W. Neudec and R.F. Pierret, *Advanced Semiconductor Fundamentals*, Englewood Cliffs, NJ: Prentice Hall, 2003.
7. S. M. Sze, *Semiconductor Devices—Physics and Technology*, New York, NY: John Wiley & Sons, Inc., 2002.
8. J. Singh, *Semiconductor Devices—Basic Principles*, New York, NY: John Wiley & Sons, Inc., 2001.

```
/* Se1d.c.  1D FDTD Schroedinger simulation */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define KE  200

main ()
{
    float psi_rl[KE],psi_im[KE];
    float Vpot,vp[KE];
    int n,k,kc,ke,kstart,kcenter,NSTEPS,n_pml;
    float pi,melec,hbar,eV2J,J2eV;
    float lambda,sigma,ptot1,ptot2;
    float ddx,dt,ra,T;
    float pulse,xn,xxn;
    FILE *fp, *fopen();
    float lap_rl,lap_im,ke_rl,ke_im,kine,PE;
    float gam_rl[KE],gam_im[KE],sig[KE],RR,sig0;
    float del_rl[KE],del_im[KE],drl,dim,denom;
    int npml,redge,ledge;

     pi = 3.14159;
     melec = 9.2e-31;   /* Mass of an electron */
     hbar = 1.055e-34;  /* Plank's constant    */
     eV2J = 1.6e-19;    /* Convert eV to J     */
     J2eV = 1./1.6e-19; /* Convert J to eV     */

     ddx = .2e-9;     /* Cell size */
     ra = 1./8.;
        printf( "ra = %e \n",ra);
     dt = .25*(melec/hbar)*pow(ddx,2.);
```

```
        printf( " dt = %e \n",dt);

    /* Initialize */
       for ( k=0; k < KE; k++ )
       { psi_rl[k] = 0.;
         psi_im[k] = 0.;
         gam_rl[k] = 1.;
         gam_im[k] = 0.;
         del_rl[k] = 0.;
         del_im[k] = 0.;
         sig[k] = 0.;
         vp[k] = 0.;
       }

   /* ZPML */

      sig0 = .002;
      sig0 = .02;
      npml = 20;

        /* Left side */
       ledge = npml;
       for (k=0; k < ledge; k++) {
          sig[k] = sig0*pow((ledge-k),2.);
          drl = 1. + 0.707*sig[k];
          dim =  -0.707*sig[k] - pow(sig[k],2.);
          denom = pow(drl,2.) + pow(dim,2.);
          gam_rl[k] = drl/denom;
          gam_im[k] = dim/denom;
      printf("%2d %12.5e    %8.5f %8.5f
         \n",k,sig[k],gam_rl[k],gam_im[k]);
       }

        /* Right side */
       redge = KE - npml;
       for (k=redge; k < KE; k++) {
          sig[k] = sig0*pow((k - redge),2.);
          drl = 1. + 0.707*sig[k];
          dim =  -0.707*sig[k] - pow(sig[k],2.);
          denom = pow(drl,2.) + pow(dim,2.);
          gam_rl[k] = drl/denom;
          gam_im[k] = dim/denom;
          printf("%2d %12.5e    %8.5f %8.5f
\n",k,sig[k],gam_rl[k],gam_im[k]);
       }

       /* Add the Potential */

       kstart = 100;
```

```
       /* Dielectric step */
       Vpot = 0.2;
   /*  for ( k=kstart; k < KE; k++ )
       { vp[k] = Vpot*1.602e-19 ;  }*/ /* Convert eV to Joules */

       /* Tunneling Barrier */
       Vpot = 0.4;
       for ( k=kstart-10; k < kstart+10; k++ )
       { vp[k] = Vpot*1.602e-19 ;  }  /* Convert eV to Joules */

      /* Initialize the pulse */

   /*  printf( "Initialize to pulse at  --> ");
       scanf("%d", &kcenter); */
       kcenter = 50;
        printf( "kcenter = %3d \n",kcenter);
    /* printf( "lambda,sigma --> ");
       scanf("%f ",&lambda );
       scanf("%f", &sigma); */
       lambda = 10;
       sigma = 10;
        ptot1 = 0.;
       for ( k=1; k < kstart; k++ )
       { psi_rl[k] = cos(2*pi*(k-kcenter)/lambda)
                   * exp( -.5*pow( (k-kcenter)/sigma,2.) );
         psi_im[k] = sin(2*pi*(k-kcenter)/lambda)
                   * exp( -.5*pow( (k-kcenter)/sigma,2.) );
         ptot1 = ptot1 + pow(psi_rl[k],2.) + pow(psi_im[k],2.);
       }

        printf( " ptot1= %f \n",ptot1);

       /* Normalize the waveform */
       for ( k=1; k < kstart; k++ )
       { psi_rl[k] = psi_rl[k]/sqrt(ptot1);
         psi_im[k] = psi_im[k]/sqrt(ptot1); }

    kc = KE/2;
    T = 0;
    NSTEPS = 1;

while ( NSTEPS > 0 ) {
       printf( "NSTEPS --> ");
       scanf("%d", &NSTEPS);
       printf("%d \n", NSTEPS);
       n= 0;

    for ( n=1; n <=NSTEPS ; n++)
```

```
    {
       T = T + 1;

    /*  Main FDTD Loop */

       /* Calculate the Real part */

       for ( k=1; k < KE-1; k++ )
       {  del_im[k] = psi_im[k+1] - 2.*psi_im[k] + psi_im[k-1];
          psi_rl[k] = psi_rl[k]
             + ra*( -gam_rl[k]*del_im[k] - gam_im[k]*del_rl[k] )
             + (dt/hbar)*vp[k]*psi_im[k] ;
       }

       /* Calculate the imaginary part */

       for ( k=1; k < KE-1; k++ )
       {  del_rl[k] =  psi_rl[k+1] - 2.*psi_rl[k] + psi_rl[k-1] ;
          psi_im[k] = psi_im[k]
             + ra*( gam_rl[k]*del_rl[k] - gam_im[k]*del_im[k] )
             - (dt/hbar)*vp[k]*psi_rl[k] ;
     }  }

   /* End of the Main FDTD Loop */

       ptot1 = 0.;
       for ( k=1; k < kstart; k++ )
       {  ptot1 = ptot1 + pow(psi_rl[k],2.)
+ pow(psi_im[k],2.); }

       ptot2 = 0.;
       for ( k=kstart; k < KE; k++ )
       {  ptot2 = ptot2 + pow(psi_rl[k],2.)
+ pow(psi_im[k],2.); }

       /* Print out the real and imaginary parts */
       for ( k=0; k < KE; k++ )
       { printf( "%3d   %9.5f  %9.5f\n",k,psi_rl[k],psi_im[k]); }

       /* Write the real part to a file "prl" */
       fp = fopen( "prl","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %8.4f \n",psi_rl[k]); }
       fclose(fp);

       /* Write the imaginary part to a file "pim" */
       fp = fopen( "pim","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %8.4f \n",psi_im[k]); }
```

```
       fclose(fp);

     /* Calculate the KE and PE */

      ke_rl = 0.;
      ke_im = 0.;
      PE = 0.;

      for ( k=0; k < KE; k++ ) {lap_rl = psi_rl[k+1] -
                     2.*psi_rl[k] + psi_rl[k-1] ; /* Laplacian */
        lap_im = psi_im[k+1] - 2.*psi_im[k] + psi_im[k-1] ;
        ke_rl = ke_rl + psi_rl[k]*lap_rl + psi_im[k]*lap_im;
        ke_im = ke_im + psi_rl[k]*lap_im - psi_im[k]*lap_rl;
        PE = PE + vp[k]*(pow(psi_rl[k],2.) + pow(psi_im[k],2.));
      }
        kine = .5*(hbar/melec)*(hbar/pow(ddx,2.))
             *sqrt(pow(ke_rl,2.) + pow(ke_im,2.));
        printf( " ptot1 = %6.3f  \n",ptot1);
        printf( " ke = %10.6f eV \n",J2eV*kine); /* convert J to
eV */
        printf( " pe = %10.6f eV \n",J2eV*PE);
        printf( " E  = %10.6f eV \n",J2eV*(PE+kine));

        fp = fopen("E","w");
        fprintf( fp,"  %8.4f %8.4f %6.2f %7.4f %7.4f \n",
        J2eV*kine,J2eV*PE,1e12*T*dt,ptot1,ptot2);
        fclose(fp);

       printf( "T = %5.0f\n",T);
}
}

/* Se_lat.c.  Particle in a lattice.*/

   /* The crystal lattice */

        nspace = 20;     /* Spacing of the spikes.   */
        V0 = -.1;       /* Magnitude of the spikes. */

        for ( k=0; k < 50; k++ ) {
           k_space = nspace*k + 10;
           vp[k_space] = V0*eV2J ;
        }

       kstart = 400;

        /* Initialize the pulse */
        sigma = 40;
        printf( "lambda --> ");
```

```
       scanf("%f", &lambda );
       ptot = 0.;
       for ( k=1; k < KE-1; k++ )
     {  psi_rl[k] = cos(2*pi*(k-kstart)/lambda)
                  * exp( -.5*pow( (k-kstart)/sigma,2.) );
        psi_im[k] = sin(2*pi*(k-kstart)/lambda)
                  * exp( -.5*pow( (k-kstart)/sigma,2.) ) ;
        ptot = ptot + pow(psi_rl[k],2.) + pow(psi_im[k],2.);
     }

   T = 0;
    NSTEPS = 1;

while ( NSTEPS > 0 ) {
       printf( "NSTEPS --> ");
       scanf("%d", &NSTEPS);
       printf("%d \n", NSTEPS);
       n= 0;

    for ( n=1; n <=NSTEPS ; n++)
    {
       T = T + 1;

    /*  Main FDTD Loop */

          /* Real part */

    psi_rl[0]=psi_rl[0]-ra*(psi_im[KE-2]-2.*psi_im[0]+ psi_im[1])
                 + (dt/hbar)*vp[0]*psi_im[0];
       psi_rl[KE-1] = psi_rl[0];

       for ( k=1; k < KE-1; k++ )
       { psi_rl[k] = psi_rl[k]
                - ra*( psi_im[k+1] - 2.*psi_im[k] + psi_im[k-1] )
                + (dt/hbar)*vp[k]*psi_im[k] ;
       }

          /* Imaginary part */

          psi_im[0]=psi_im[0]
                +ra*(psi_rl[KE-2]-2.*psi_rl[0] + psi_rl[1])
              - (dt/hbar)*vp[0]*psi_rl[0];
       psi_im[KE-1] = psi_im[0];

       for ( k=1; k < KE-1; k++ )
       { psi_im[k] = psi_im[k]
                + ra*( psi_rl[k+1] - 2.*psi_rl[k] + psi_rl[k-1] )
                - (dt/hbar)*vp[k]*psi_rl[k] ;
     }
```

```
}

  /* End of the Main FDTD Loop */

       /* Write the real part to a file "prl" */
       fp = fopen( "prl","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %7.3f \n",psi_rl[k]); }
       fclose(fp);

       /* Write the imaginary part to a file "pim" */
       fp = fopen( "pim","w");
       for ( k=0; k < KE; k++ )
       { fprintf( fp,"  %6.2f \n",psi_im[k]); }
       fclose(fp);

   /* Calculate the KE and PE */

      ke_rl = 0.;
      ke_im = 0.;
      PE = 0.;
      ptot = 0.;
      for ( k=0; k < KE-2; k++ ) {
         ptot = ptot + pow(psi_rl[k],2.) + pow(psi_im[k],2.);
      }
        printf( " ptot = %6.3f  \n",ptot);

      for ( k=1; k < KE-2; k++ ) {
   lap_rl = psi_rl[k+1] - 2.*psi_rl[k] + psi_rl[k-1] ;
lap_im = psi_im[k+1] - 2.*psi_im[k] + psi_im[k-1] ;
        ke_rl = ke_rl + psi_rl[k]*lap_rl + psi_im[k]*lap_im;
        ke_im = ke_im + psi_rl[k]*lap_im - psi_im[k]*lap_rl;
        PE = PE + vp[k]*(pow(psi_rl[k],2.) + pow(psi_im[k],2.));
      }
        kine = .5*(hbar/melec)*(hbar/pow(ddx,2.))
             *sqrt(pow(ke_rl,2.) + pow(ke_im,2.));
        printf( " ke = %7.4f eV \n",J2eV*kine);
        printf( " pe = %7.4f eV \n",J2eV*PE);

        fp = fopen("E","w");
        fprintf(fp," %8.3f %8.4f %6.2f %5.3f %5.2f %5.2f\n",
        J2eV*kine,J2eV*PE,1e12*T*dt,ptot,lambda*ddx*1e9,nspace*dd
x*1e9);
        fclose(fp);

       printf( "T = %5.0f\n",T);
}

}
```

APPENDIX A

THE Z TRANSFORM

One of the most useful techniques in engineering or scientific analysis is transforming a problem from the time domain to the frequency domain (1–3). Using a Fourier or Laplace transform, differential equations are changed to algebraic equations and convolution integrals are changed to multiplication. When using discrete time domain functions, that is, those only defined at points on a specific interval, we use Z transforms (4, 5).

Many engineers, particularly electrical engineers, are familiar with the Z transform, which is generally used when dealing with digital signals. While using it for FDTD simulation, we have to remember that we are implicitly sampling continuous time signals at a specific interval Δt. This presents some differences in the use of Z transforms, particularly in the convolution theorem where an extra Δt is present. We begin with a little mathematical background that will explain these differences.

A.1 THE SAMPLED TIME DOMAIN AND THE Z TRANSFORM

The Dirac delta function $\delta(t)$ has its value at one point only, at $t = 0$, and its amplitude is such that

$$\int_{-\infty}^{\infty} \delta(t)\mathrm{d}t = \int_{0^-}^{0^+} \delta(t)\mathrm{d}t = 1, \tag{A.1}$$

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.

that is, integrating over $\delta(t)$ returns the value 1. Similarly, $\delta(t - t_0)$ is defined only at t_0 and integrating it through t_0 produces the value 1. This brings us to the *sifting theorem* given by

$$\int_{-\infty}^{\infty} f(t)\delta(t - t_0)\mathrm{d}t = f(t_0). \tag{A.2}$$

Note that the Fourier transform of a function multiplied by a delta function is

$$\int_{-\infty}^{\infty} f(t)\delta(t - t_0)\mathrm{e}^{-j\omega t}\mathrm{d}t = f(t_0)\mathrm{e}^{-j\omega t_0}.$$

If we have a continuous, causal signal $x(t)$ and we sample it at an interval Δt, we get the following *discrete time* signal

$$x[n] = \sum_{n=0}^{\infty} x(t)\delta(t - n \cdot \Delta t). \tag{A.3}$$

The Fourier transform of this discrete time function is

$$X(\omega) = x[0] + x[1]\mathrm{e}^{-j\omega\cdot\Delta t} + x[2]\mathrm{e}^{-j2\omega\cdot\Delta t} + \cdots \tag{A.4}$$

It is useful to define a parameter

$$z = \mathrm{e}^{j\omega\cdot\Delta t}. \tag{A.5}$$

The quantity $\omega \cdot \Delta t$ is radian frequency times time, so $\omega \cdot \Delta t$ is an angle. As long as the time interval Δt is chosen to be much smaller than any frequency ω associated with the problem, then $0 < \omega \cdot \Delta t < 2\pi$. The parameter z is a complex parameter with magnitude 1 and angle $\omega \cdot \Delta t$.

The Z transform of a discrete signal is defined as

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n} = x[0] + x[1]z^{-1} + x[2]z^{-2} + \cdots \tag{A.6}$$

This is obviously the same as Eq. (A.4) using Eq. (A.5).

For example, let us start with the time domain function

$$x(t) = \mathrm{e}^{-\alpha t}u(t) \quad \alpha \geq 0.$$

If this function is sampled at an interval Δt, then it becomes

$$x[n] = (\mathrm{e}^{-\alpha\cdot\Delta t})^n u[n] \quad n = 0, 1, 2, \ldots$$

The Z transform of this function is

$$X(z) = \sum_{n=0}^{\infty} x[n]z^{-n} = \sum_{n=0}^{\infty} (e^{-\alpha\cdot\Delta t})^n z^{-n} = \sum_{n=0}^{\infty} (e^{-\alpha\cdot\Delta t} z^{-1})^n. \tag{A.7}$$

The following summation series will prove useful (6):

$$\sum_{n=0}^{\infty} q^n = \frac{1}{1-q} \quad \text{if} \quad |q| < 1. \tag{A.8}$$

In Eq. (A.7) as long as $e^{-\alpha\cdot\Delta t} < 1$, the absolute values of the terms in parentheses will also be less than 1. Using Eq. (A.8), Eq. (A.7) becomes

$$X(z) = \sum_{n=0}^{\infty} (e^{-\alpha\cdot\Delta t} z^{-1})^n = \frac{1}{1 - e^{-\alpha\cdot\Delta t} z^{-1}}. \tag{A.9}$$

Note that in the limit as α goes to 0, $e^{-\alpha\cdot\Delta t}$ goes to 1, so

$$Z\{u[n]\} = \frac{1}{1 - z^{-1}}. \tag{A.10}$$

Another important Z transform is that of the decaying sinusoid:

$$f[n] = e^{-\alpha\cdot\Delta t\cdot n} \sin(\beta\cdot\Delta t\cdot n)u[n] = \left(e^{-\alpha\cdot\Delta t\cdot n} \frac{e^{j\beta\cdot\Delta t\cdot n} - e^{-j\beta\cdot\Delta t\cdot n}}{2j}\right) u[n]$$

$$= \frac{1}{2j}(e^{-(\alpha - j\beta)\cdot\Delta t\cdot n} - e^{-(\alpha + j\beta)\cdot\Delta t\cdot n})u[n].$$

Extrapolating from Eq. (A.9), we get,

$$F(z) = \frac{1}{2j}\left(\frac{1}{1 - e^{-\alpha\cdot\Delta t} e^{j\beta\cdot\Delta t}\cdot\Delta z^{-1}} - \frac{1}{1 - e^{-\alpha\cdot\Delta t} e^{-j\beta\cdot\Delta t}\cdot\Delta z^{-1}}\right)$$

$$= \frac{e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t)z^{-1}}{1 - 2e^{-\alpha\cdot\Delta t}\cos(\beta\cdot\Delta t)z^{-1} + e^{-2\alpha\cdot\Delta t} z^{-2}}. \tag{A.11}$$

Finally, we want to know the Z transform of the Dirac delta function that was defined in Eq. (A.1). The discrete time version of the delta function is $\delta[n]$. It maintains the value 1 over an interval Δt, unlike Eq. (A.1), which implies that $\delta(t)$ is infinitely narrow. Therefore, we say that the Z transform of this function is

$$Z\{\delta[n]\} = \frac{1}{\Delta t}. \tag{A.12}$$

The $1/\Delta t$ term is necessary for mathematical consistency.

Table A.1 summarizes Eq. (A.9), Eq. (A.10), Eq. (A.11), and Eq. (A.12). These are four of the most commonly used Z transforms.

TABLE A.1 Some Z Transforms

Time Domain	Frequency Domain	Sampled Time Domain	Z Domain
$\delta(t)$	1	$\delta[n]$	$\frac{1}{\Delta t}$
$u(t)$	$\frac{1}{j\omega}$	$u[n]$	$\frac{1}{1-z^{-1}}$
$e^{-\alpha t}u(t)$	$\frac{1}{j\omega+\alpha}$	$e^{-\alpha\cdot\Delta t\cdot n}u[n]$	$\frac{1}{1-e^{-\alpha\cdot\Delta t}z^{-1}}$
$e^{-\alpha t}\sin(\beta t)u(t)$	$\frac{\beta}{(\alpha^2+\beta^2)+j\omega 2\alpha-\omega^2}$	$e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t\cdot n)$	$\frac{e^{-\alpha\cdot\Delta t}\sin(\beta\cdot\Delta t)z^{-1}}{1-2e^{-\alpha\cdot\Delta t}\cos(\beta\cdot\Delta t)z^{-1}+e^{-2\alpha\cdot\Delta t}z^{-2}}$

A.1.1 Delay Property

We know that the Z transform of a function $x[n]$ is given by $X(z)$ in Eq. (A.7). It is also important to know the Z transform of that same function delayed by m time steps. This is determined directly from the definition:

$$Z\{x[n-m]\} = \sum_{n=m}^{\infty} x[n-m]z^{-n}.$$

The index on the summation has been changed because we assume $x[n]$ is causal, values of $x[n-m]$ are zero until $n=m$. We change the variables to $i=n-m$, which gives

$$Z\{x[n-m]\} = \sum_{i=0}^{\infty} x[i]z^{-(i+m)} = z^{-m}\sum_{i=0}^{\infty} x[i]z^{-i} = z^{-m}X(z). \tag{A.13}$$

A.1.2 Convolution Property

Often, we are given information about a system in the frequency domain. For instance, if the transfer function of a system is $H(\omega)$ and the Fourier transform of the input is $X(\omega)$, the output in the frequency domain is

$$Y(\omega) = H(\omega)X(\omega). \tag{A.14}$$

If H and X are both causal functions, the above multiplication in the frequency domain becomes a convolution in the time domain (1):

$$y(t) = \int_0^t h(t-\tau)x(\tau)\mathrm{d}\tau.$$

If we replace $x(t)$ with the sampled time domain function of Eq. (A.3), then

$$y(t) = \int_0^t h(t-\tau)\sum_{n=0}^{\infty} x(\tau)\delta(\tau - n\cdot\Delta t)\mathrm{d}\tau.$$

We know that delta functions only have values at the interval Δt, so we can use the sifting theorem to write the integral as a summation times Δt,

$$y(t) = \Delta t\sum_{n=0}^{\infty} h(t - n\cdot\Delta t)x(n\cdot\Delta t),$$

giving the discrete time function

$$y[m] = \Delta t\sum_{n=0}^{\infty} h[m-n]x[n].$$

We assume that t has been replaced by $\Delta t\cdot m$ and τ has been replaced by $\Delta t\times n$. Taking the Z transform,

$$Y(z) = \sum_{m=0}^{\infty} y[m]z^{-m} = \Delta t\cdot\sum_{m=0}^{\infty} z^{-m}\sum_{n=0}^{n} h[m-n]x[n],$$

and making the following change of variables

$$i = m - n, \quad m = i + n,$$

we get

$$\begin{aligned} Y(z) &= \Delta t\cdot\sum_{i=-n}^{\infty} z^{-i}z^{-n}\sum_{n=0}^{n} h[i]x[n] \\ &= \Delta t\cdot\sum_{i=-n}^{\infty} h[i]z^{-i}\sum_{n=0}^{n} x[n]z^{-n} = \Delta t\cdot H(z)X(z). \end{aligned} \tag{A.15}$$

The summation over i was truncated to begin at zero because $h[i]$ is a causal function. The extra factor Δt in Eq. (A.15) is the difference in this version of the Z transform convolution and what is usually seen in an introductory text containing Z transforms.

Note that if we calculate the convolution of a discrete time function $x[n]$ with the delta function $\delta[n]$ in the Z domain

$$Y(z) = \Delta t\cdot X(z)\cdot\frac{1}{\Delta t} = X(z),$$

we get back the original function, as we should. These properties are summarized in Table A.2.

TABLE A.2 Properties of Fourier Transforms

Property	Time Domain	Z Domain
Definition	$x[n]$	$X(\omega)$
Time shift	$x[n-m]$	$z^{-m}X(\omega)$
Convolution	$y[m] = \Delta t \sum_{n=0}^{\infty} h[m-n]x[n]$	$\Delta t \cdot H(\omega)X(\omega)$

A.2 EXAMPLES

As an example, suppose we are to develop a computer program to calculate the convolution of the function $x(t)$ with a low pass filter given by

$$H(\omega) = \frac{\omega_1}{j\omega + \omega_1}. \tag{A.16}$$

If $y(t)$ is the output, then

$$Y(\omega) = \frac{\omega_1}{j\omega + \omega_1} X(\omega)$$

is the frequency domain. This filter has a cutoff frequency of $\omega_1 = 2\pi \times 10^3$ rad/s, so a sampling time of 0.01 ms should be adequate. Looking at Table A.1, the function H in the Z domain is

$$H(z) = \frac{\omega_1}{1 - e^{-\omega_1 \cdot \Delta t} z^{-1}}.$$

(Note that $\omega_1 \cdot \Delta t = 0.0628$, so $e^{-\omega_1 \cdot \Delta t} = 0.9391$.) The Z transform of the input $x(t)$ is $X(z)$, and that of the output is $Y(z)$. Therefore, by the convolution theorem, the Z transform of the output is

$$Y(z) = \Delta t \cdot \frac{\omega_1}{1 - e^{-\omega_1 \cdot \Delta t} z^{-1}} X(z). \tag{A.17}$$

We can write this as

$$(1 - e^{-\omega_1 \cdot \Delta t} z^{-1}) Y(z) = \omega_1 \cdot \Delta t \cdot X(z)$$

or

$$Y(z) = e^{-\omega_1 \cdot \Delta t} z^{-1} Y(z) + \omega_1 \cdot \Delta t \cdot X(z).$$

If we go to the sampled time domain, the above Z domain equation becomes

$$y[n] = e^{-\omega_1 \cdot \Delta t} \cdot y[n-1] + \omega_1 \cdot \Delta t \cdot x[n]. \tag{A.18}$$

This is calculated by the following C computer code:

```
y[1] = omega*dt*x[n];
for n=2; n< NN; n++) {
    y[n] = exp(-omega*dt)*y[n-1] + omega*dt*x[n]
}
```

The response to a step function is shown in Fig. A.1. Analytically, the step response to a filter of the form in Eq. (A.16) should be

$$y(t) = (1 - e^{-\omega_1 t})u(t).$$

We can see that $y[n]$ in Fig. A.1 levels off at about 1.03. The accuracy can be improved by a smaller sampling rate (See problem A.1).

Suppose our low pass filter were a two-pole low pass filter with the following transfer function:

$$H(\omega) = \frac{\beta}{(\alpha^2 + \beta^2) + j\alpha\omega - \omega^2}.$$

The output for an input $X(\omega)$ in the Z domain is

$$Y(z) = \Delta t \cdot \frac{e^{-\alpha \cdot \Delta t} \sin(\beta \cdot \Delta t) z^{-1}}{1 - 2e^{-\alpha \cdot \Delta t} \cos(\beta \cdot \Delta t) z^{-1} + e^{-2\alpha \cdot \Delta t} z^{-2}} X(z).$$

The corresponding sampled time domain function is

$$\begin{aligned} y[n] = {} & 2e^{-\alpha \cdot \Delta t} \cos(\beta \cdot \Delta t) \cdot y[n-1] \\ & - e^{-2\alpha \cdot \Delta t} y[n-2] + e^{-\alpha \cdot \Delta t} \sin(\beta \cdot \Delta t) \cdot \Delta t \cdot x[n-1]. \end{aligned}$$

In these examples, we proceeded directly from the frequency domain to the Z domain because the one- and two-pole low pass filters are terms listed in

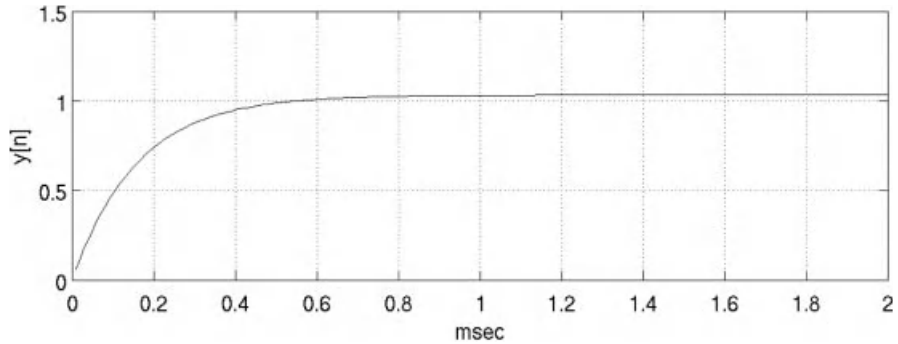

Figure A.1 The response of Eq. (A.18) to a step function input, that is, $x[n] = u[n]$.

Table A.1. If the frequency domain function is a higher order function that is not in the table, it may be possible to break the function into terms that are in the table by the method of partial fraction expansion (1).

A.3 APPROXIMATIONS IN GOING FROM THE FOURIER TO THE Z DOMAIN

In going from the frequency to the Z domain, there are times when it is difficult to break the frequency domain function into separate terms that can be found in a table like Table A.1. There are methods to directly go from ω to z, but they are approximations.

It can be shown (1) that if

$$F\{f(t)\} = F(\omega)$$

then

$$F\left\{\frac{\mathrm{d}f(t)}{\mathrm{d}t}\right\} = j\omega F(\omega).$$

A derivative can be approximated by

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \cong \frac{f(t) - f(t-\Delta t)}{\Delta t},$$

as long as Δt is small compared to how fast $f(t)$ is changing. As these may be thought of as two discrete points, we can take the Z transform

$$Z\left\{\frac{f(t) - f(t-\Delta t)}{\Delta t}\right\} = \frac{F(z) - z^{-1}F(z)}{\Delta t} = \frac{1 - z^{-1}}{\Delta t}F(z).$$

So at least as an approximation, we can say that $j\omega$ in the frequency domain becomes $(1 - z^{-1})/\Delta t$ in the Z domain. This is known as the backward rectangular approximation (4).

As an example, look back at the low pass filter in Eq. (A.16). Applying the backward linear approximation gives

$$H(z) = \frac{10^3}{\dfrac{1 - z^{-1}}{\Delta t} + 10^3} = \frac{10^3 \cdot \Delta t}{1 + 10^3 \cdot \Delta t - z^{-1}} = \frac{10^3 \dfrac{\Delta t}{(1 + 10^3 \cdot \Delta t)}}{1 - (1 + 10^3 \cdot \Delta t)^{-1} z^{-1}}.$$

If we use this to calculate the convolution of $x[n]$ with $h[n]$, we get the Z domain function similar to Eq. (A.17), which leads to the following sampled time domain function similar to Eq. (A.18):

$$y[n] = \frac{1}{1 + 10^3 \cdot \Delta t} y[n-1] + \frac{10^3 \cdot \Delta t}{1 + 10^3 \cdot \Delta t} x[n]. \tag{A.19}$$

Notice that as long as $\Delta t \ll 10^3$,

$$\frac{10^3 \cdot \Delta t}{1 + 10^3 \cdot \Delta t} \cong 10^3 \cdot \Delta t$$

and

$$\frac{1}{1 + 10^3 \cdot \Delta t} \cong \mathrm{e}^{-10^3 \cdot \Delta t}.$$

Therefore, Eq. (A.19) is almost the same as Eq. (A.18).

The following transform is the equivalent of using a trapezoidal approximation to a derivative (4):

$$j\omega = \frac{2}{\Delta t} \frac{1 - z^{-1}}{1 + z^{-1}}. \tag{A.20}$$

Since it takes a linear approximation over two time steps, it is more accurate than the backward rectangular approximation. Again, starting with Eq. (A.16), by using Eq. (A.20) we get

$$\begin{aligned} H(z) &= \frac{10^3}{\dfrac{2}{\Delta t}\dfrac{1 - z^{-1}}{1 + z^{-1}} + 10^3} = \frac{10^3 \cdot \Delta t(1 + z^{-1})}{2(1 - z^{-1}) + 10^3 \cdot \Delta t(1 + z^{-1})} \\ &= \frac{10^3 \cdot \Delta t(2 + 10^3 \cdot \Delta t)^{-1}(1 + z^{-1})}{1 - \left(\dfrac{1 - 10^3 \cdot \dfrac{\Delta t}{2}}{1 + 10^3 \cdot \dfrac{\Delta t}{2}}\right) z^{-1}} \cong \frac{10^3 \cdot \Delta t}{2} \frac{(1 + z^{-1})}{1 - (1 - 10^3 \cdot \Delta t) z^{-1}}. \end{aligned} \tag{A.21}$$

The FDTD equation comparable to Eq. (A.19) is

$$y[n] = (1 - 10^3 \cdot \Delta t) y[n - 1] + \frac{10^3 \cdot \Delta t}{2}(x[n] - x[n - 1]),$$

which basically says that the input $x[n]$ is averaged over two time steps.

PROBLEM SET A

1. Write a program that can duplicate the results of Fig. (A.1). Change the time step from 0.01 to 0.001 ms. Is it more accurate?

2. Rewrite the program from problem A.1 using the backward linear approximation that results in Eq. (A.19). Do this for both time steps of 0.1 and 0.01 ms.

3. Rewrite the program from problem A.2 using the bilateral transform of Eq. (A.21). How large a time step can you use and still get fairly accurate results?

REFERENCES

1. Z. Gajic, *Linear Dynamic Systems and Signals*, Upper Saddle River, NJ: Prentice Hall, 2003.
2. C. L. Phillips, J. M. Parr, and E. A. Riskin, *Signals, System, and Transforms*, Upper Saddle River, NJ: Prentice Hall, 2008.
3. P. D. Chan and J. I. Moliner, *Fundamentals of Signals and Systems*, Cambridge, UK: Cambridge Press, 2006.
4. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Englewood Cliffs, NJ: Prentice Hall, 1975.
5. S. K. Mitra, *Digital Signal Processing—A Computer-Based Approach*, 4th Edition, New York, NY: McGraw-Hill, 2001.
6. E. Kreyszic, *Advanced Engineering Mathematics*, 6th Edition, New York, NY: John Wiley & Sons, Inc., 1988.
7. D. M. Sullivan, A frequency-dependent FDTD method using Z transforms, *IEEE Trans. Antenn. Propag.*, vol. 40, October 1992, pp. 1223–1230.
8. D. M. Sullivan, Z transform theory and the FDTD method, *IEEE Trans. Antenn. Propag.*, vol. 44, October 1999, pp. 28–34.

INDEX

(Note: Page numbers in *italics* refer to Figures; those in **bold** to Tables.)

Electromagnetic Simulation Using the FDTD Method, Second Edition. Dennis M. Sullivan.