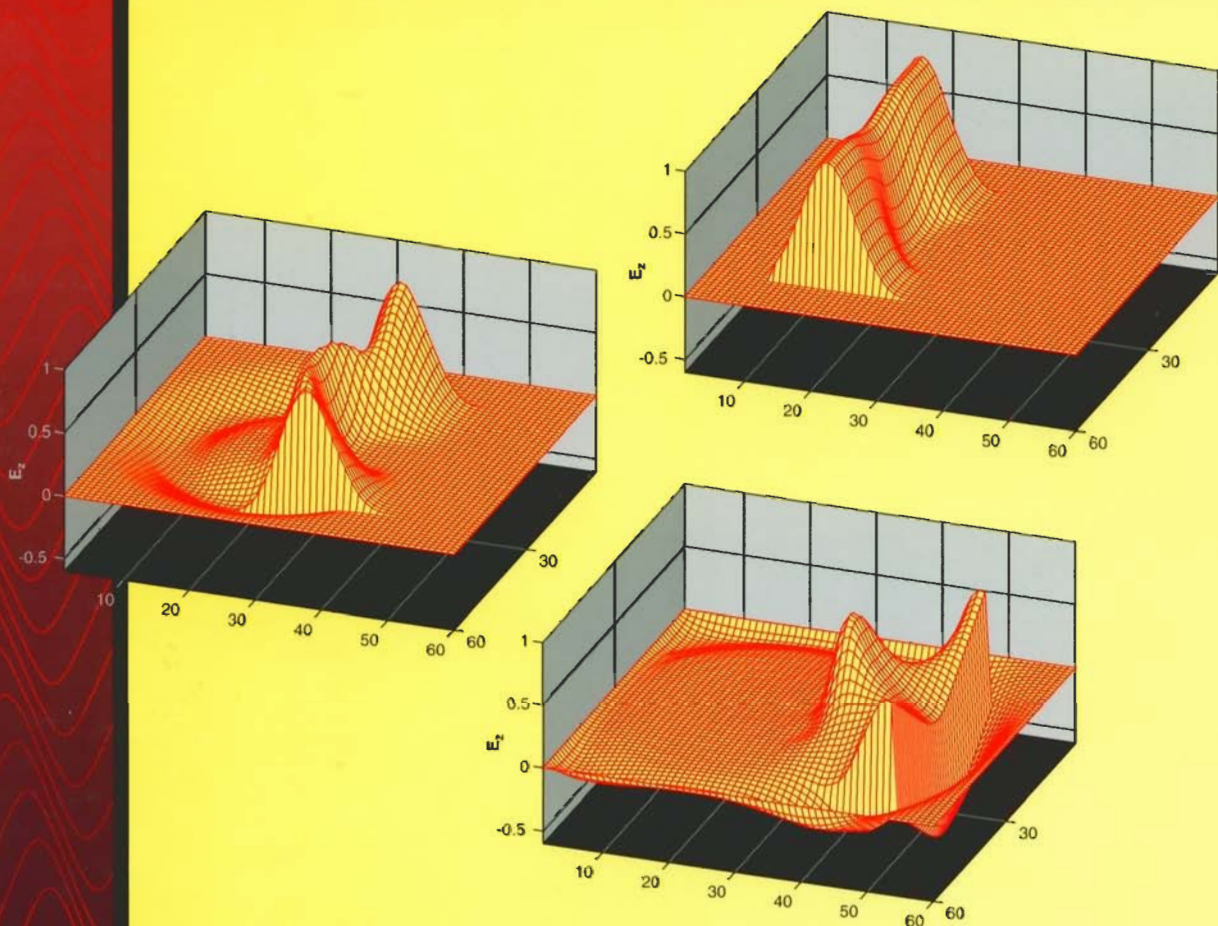


# ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

DENNIS M. SULLIVAN



IEEE PRESS SERIES ON RF AND MICROWAVE TECHNOLOGY  
Roger D. Pollard and Richard Booton, *Series Editors*

# ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

IEEE Press  
445 Hoes Lane, P.O. Box 1331  
Piscataway, NJ 08855-1331

**IEEE Press Editorial Board**

Robert J. Herrick, *Editor in Chief*

M. Akay	M. Eden	M. S. Newman
J. B. Anderson	M. E. El-Hawary	M. Padgett
P. M. Anderson	R. F. Hoyt	W. D. Reeve
J. E. Brewer	S. V. Kartalopoulos	G. Zobrist
	D. Kirk	

Kenneth Moore, *Director of IEEE Press*  
Catherine Faduska, *Senior Acquisitions Editor*  
Linda Matarazzo, *Associate Acquisitions Editor*  
Anthony VenGraitis, *Project Editor*  
Marilyn G. Catis, *Marketing Manager*

IEEE Microwave Theory and Techniques Society, *Sponsor*  
MTT-S Liaison to IEEE Press, Lawrence Dunleavy

Cover Design: William T. Donnelly, *WT Design*

**Technical Reviewers**

Roger Pollard, *University of Leeds, United Kingdom*  
Richard W. Ziolkowski, *University of Arizona, Tucson, AZ*  
Tatsuo Itoh, *UCLA*  
Raymond Luebbers, *Pennsylvania State University, University Park, PA*  
Melinda Piket-May, *University of Colorado at Boulder*  
Nihad I. Dib, *Jordan University of Science and Technology, Irbid, Jordan*

**Books of Related Interest from the IEEE Press**

*MAGNETIC RECORDING: The First 100 Years*

Eric D. Daniel, Denis C. Mee, Mark Clark

1999      Softcover      360 pp      IEEE Order No. PP5396      ISBN 0-7803-4709-9

*MAGNETIC HYSTERESIS*

Edward Della Torre

1999      Hardcover      240 pp      IEEE Order No. PC5766      ISBN 0-7803-4719-6

*MAGNETO-OPTICAL RECORDING MATERIALS*

Edited by Richard J. Gambino and Takao Suzuki

2000      Hardcover      424 pp      IEEE Order No. PC3582      ISBN 0-7803-1009-8

*EMC AND THE PRINTED CIRCUIT BOARD: Design, Theory, and Layout Made Simple*

Mark I. Montrose

1999      Hardcover      344 pp      IEEE Order No. PC5756      ISBN 0-7803-4703-X

*PRINTED CIRCUIT BOARD DESIGN TECHNIQUES FOR EMC COMPLIANCE: A Handbook for Designers*, Second Edition

Mark I. Montrose

2000      Hardcover      336 pp      IEEE Order No. PC5816      ISBN 0-7803-5376-5

# ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

**Dennis M. Sullivan**

*Electrical Engineering Department*

*University of Idaho*

IEEE Microwave Theory and Techniques Society, *Sponsor*



**IEEE  
PRESS**



IEEE Press Series on RF and Microwave Technology

Roger D. Pollard and Richard Booton, *Series Editors*

The Institute of Electrical and Electronics Engineers, Inc., New York

This book and other books may be purchased at a discount from the publisher when ordered in bulk quantities. Contact:

IEEE Press Marketing  
Attn: Special Sales  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331  
Fax: +1 732 981 9334

For more information about IEEE Press products, visit the IEEE Online Catalog & Store: <http://www.ieee.org/ieeestore>.

© 2000 by the Institute of Electrical and Electronics Engineers, Inc.  
3 Park Avenue, 17th Floor, New York, NY 10016-5997

*All rights reserved. No part of this book may be reproduced in any form, nor may it be stored in a retrieval system or transmitted in any form, without written permission from the publisher.*

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

**ISBN 0-7803-4747-1**  
**IEEE Order No. PC5400**

### **Library of Congress Cataloging-in-Publication Data**

Sullivan, Dennis M.

Electromagnetic simulation using the FDTD method / Dennis M. Sullivan ; IEEE Microwave Theory and Techniques Society, sponsor.

p. cm. -- (IEEE Press series on RF and microwave technology)

Includes bibliographical references and index.

ISBN 0-7803-4747-1

1. Electromagnetism--Computer simulation. 2. Finite differences. 3. Time-domain analysis. I. Title. II. Series.

QC760 .S92 2000  
537'.01'13--dc21

00-038922

To  
Sully and Jane

# Contents

## **GUIDE TO THE BOOK xi**

### **CHAPTER 1 ONE-DIMENSIONAL SIMULATION WITH THE FDTD METHOD 1**

- 1.1 One-Dimensional Free Space Formulation 1
- 1.2 Stability and the FDTD Method 4
- 1.3 The Absorbing Boundary Condition in One Dimension 4
- 1.4 Propagation in a Dielectric Medium 5
- 1.5 Simulating Different Sources 7
- 1.6 Determining Cell Size 8
- 1.7 Propagation in a Lossy Dielectric Medium 9
- Appendix 1.A 11
- References 11

### **CHAPTER 2 MORE ON ONE-DIMENSIONAL SIMULATION 19**

- 2.1 Reformulation Using the Flux Density 19
- 2.2 Calculating the Frequency Domain Output 21
- 2.3 Frequency-Dependent Media 23
  - 2.3.1 Auxiliary Differential Equation Method 26
- 2.4 Formulation Using Z Transforms 27
  - 2.4.1 Simulation of an Unmagnetized Plasma 28
- 2.5 Formulating a Lorentz Medium 31
  - 2.5.1 Simulation of Human Muscle Tissue 33
- References 35

<b>CHAPTER 3</b>	<b>TWO-DIMENSIONAL SIMULATION</b>	<b>49</b>
3.1	FDTD in Two Dimensions	49
3.2	The Perfectly Matched Layer (PML)	51
3.3	Total/Scattered Field Formulation	58
3.3.1	A Plane Wave Impinging on a Dielectric Cylinder	59
3.3.2	Fourier Analysis	61
	References	63
<b>CHAPTER 4</b>	<b>THREE-DIMENSIONAL SIMULATION</b>	<b>79</b>
4.1	Free Space Formulation	79
4.2	The PML in Three Dimensions	83
4.3	Total/Scattered Field Formulation in Three Dimensions	85
4.3.1	A Plane Wave Impinging on a Dielectric Sphere	85
	References	89
<b>CHAPTER 5</b>	<b>TWO APPLICATIONS USING FDTD</b>	<b>109</b>
5.1	Simulation of a Microstrip Antenna	109
5.1.1	Description of the Problem	109
5.1.2	Modeling the Materials	110
5.1.3	Source	111
5.1.4	Boundary Conditions	111
5.1.5	Calculating the $S_{11}$	112
5.2	Calculation of the Far Field of an Aperture Antenna	113
5.2.1	Formulating the Transformation from the Aperture	115
5.2.2	Verification of the Accuracy of the Transformation	118
5.2.3	FDTD Implementation of the Far Field Calculations	120
	References	121
<b>CHAPTER 6</b>	<b>USING FDTD FOR OTHER TYPES OF SIMULATION</b>	<b>133</b>
6.1	The Acoustic FDTD Formulation	133
6.2	Simulation of the Schroedinger Equation	136
6.2.1	Formulating the Schroedinger Equation into FDTD	137
6.2.2	Calculating the Expectation Values of the Observables	138
6.2.3	Simulation of an Electron Striking a Potential Barrier	139
	References	140
<b>APPENDIX</b>	<b>THE Z TRANSFORM</b>	<b>147</b>
A.1	Definition of the Z Transform	147
A.2	Convolution Using the Z Transform	148
A.2.1	Proof of the Convolution Theorem	149
A.2.2	Example: A Low-Pass Filter	150
A.3	Convolution of Sampled Signals	152
A.3.1	Simulation of a Two-Pole Digital Filter	152
A.3.2	Sum of Two Parallel Systems	154



A.4 Alternative Methods to Formulate the Z Transform	155
A.4.1 Backward Rectangular Approximation	156
A.4.2 Trapezoidal Approximation (Bilinear Transform)	157
A.5 Summary	158
References	159

**INDEX 161**

**LIST OF C PROGRAMS 163**

**ABOUT THE AUTHOR 165**

# Guide to the Book

## PURPOSE

This book has one purpose only: to enable the reader or student to learn how to 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 to 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:

<b>Dimension of Simulation</b>	<b>Type of Material</b>
One-dimensional	Free space
Two-dimensional	Dielectric material
Three-dimensional	Lossy dielectric material
	Frequency-dependent material

The first section of Chapter 1 is one-dimensional simulation in free space. From there it progresses 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. This is the approach taken throughout the book.

## 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, one 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 among different approaches. In most cases, the approach being taught is this author's preference. That 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 (Gaussian) units.** Maxwell's equations have been normalized by substituting

$$\tilde{\mathbf{E}} = \sqrt{\frac{\epsilon_0}{\mu_0}} \mathbf{E}.$$

This is a system called *Gaussian units*, which is frequently used by physicists. The reason for using it here is simplicity in the formulations. The  $E$  field 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. **The PML for boundary conditions.** The absorbing boundary conditions (ABCs) are an important topic in FDTD simulation. The ABCs prevent spurious reflections from the edge of the problem space. There are numerous approaches to this, but this book will use the perfectly matched layer (PML) for two- and three-dimensional simulation exclusively. (A simpler boundary condition will be used in one dimension just for convenience.) The reason is its effectiveness and versatility in working with different media.
3. **Maxwell's equations with 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:

$$\frac{\partial \mathbf{E}}{\partial t} = \frac{1}{\epsilon_0} \nabla \times \mathbf{H} - \frac{\sigma}{\epsilon_0} \mathbf{E} \quad (1)$$

$$\frac{\partial \mathbf{H}}{\partial t} = \frac{1}{\mu_0} \nabla \times \mathbf{E}. \quad (2)$$

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

$$\frac{\partial \mathbf{D}}{\partial t} = \nabla \times \mathbf{H} \quad (3)$$

$$\mathbf{D} = \epsilon \mathbf{E} \quad (4)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \mathbf{E}. \quad (5)$$

In this formulation, it is assumed that the materials being simulated are nonmagnetic; that is,  $\mathbf{H} = \frac{1}{\mu_0} \mathbf{B}$ . 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: Eqs. (3) and (5) remain the same regardless of the material; any complicated mathematics stemming from the material is 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. That being the case, the most direct approach to solving 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 graduate class on FDTD simulation, I begin the semester by teaching two topics in parallel: FDTD simulation and Z transforms. When we have progressed to the simulation of complicated dispersive materials, the students are ready to apply the Z transform theory. This had two distinct advantages over and above the simulation applications: (1) Electrical engineering students have had another application of Z transforms to strengthen their understanding of signal processing and filter theory; and (2) physics students and others now know and can use Z transforms, something that had not previously 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. An appendix of Z transform theory is provided.

## **PROGRAMMING EXERCISES**

The philosophy behind this book is that the student 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 of Chapter 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 3 describes how a simple absorbing boundary condition is implemented. The program `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 bold-face.

## **PROGRAMMING LANGUAGE**

The programs at the end of each chapter are written in the C programming language. The reasons for this are the almost universal availability of C compilers on UNIX workstations, and the fact that so many engineers and scientists know C. However, the reader should keep one fact in mind: most researchers who do large scientific programming use FORTRAN because FORTRAN was written for scientific programming. The structured style of C may have aesthetic appeal, but it typically runs slower than FORTRAN. This is particularly true of supercomputers.

Dennis M. Sullivan  
*Electrical Engineering Department*  
*University of Idaho*

# Acknowledgments

I am deeply indebted to Professor John Schneider of Washington State University for technical assistance, and to Ms. Judith C. Breedlove for editorial assistance.

Dennis M. Sullivan  
*Electrical Engineering Department*  
*University of Idaho*

# One-Dimensional Simulation with the FDTD Method

This chapter is a step-by-step introduction to the FDTD 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 the formulation for more complicated media

## 1.1 ONE-DIMENSIONAL FREE SPACE FORMULATION

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

$$\frac{\partial \mathbf{E}}{\partial t} = \frac{1}{\epsilon_0} \nabla \times \mathbf{H} \quad (1.1a)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \mathbf{E}. \quad (1.1b)$$

$\mathbf{E}$  and  $\mathbf{H}$  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

$$\frac{\partial E_x}{\partial t} = -\frac{1}{\epsilon_0} \frac{\partial H_y}{\partial z} \quad (1.2a)$$

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

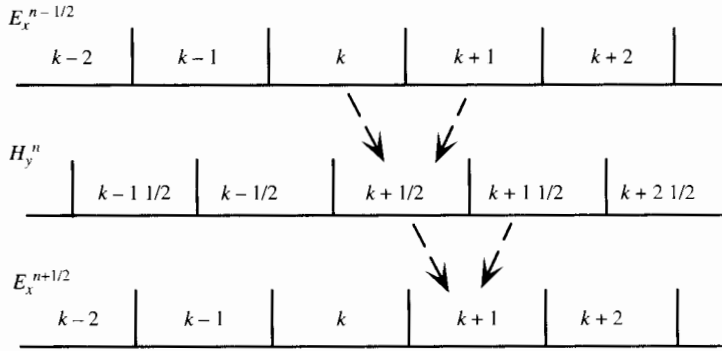
These are the equations of a plane wave with the electric field oriented in the  $x$  direction, the magnetic field oriented in the  $y$  direction, and traveling in the  $z$  direction.

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

$$\frac{E_x^{n+1/2}(k) - E_x^{n-1/2}(k)}{\Delta t} = -\frac{1}{\epsilon_0} \frac{H_y^n(k+1/2) - H_y^n(k-1/2)}{\Delta x} \quad (1.3a)$$

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

In these two equations, time is specified by the superscripts, i.e., “ $n$ ” actually means a time  $t = \Delta t \cdot n$ . 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, i.e., “ $k$ ” actually means the distance  $z = \Delta x \cdot k$ . (It might seem more sensible to use  $\Delta z$  as the incremental step, since in this case we are going in the  $z$  direction. However,  $\Delta x$  is so commonly used for a spatial increment that I will use  $\Delta x$ .) The formulation of Eq. (1.3a) and (1.3b) assumes that the  $\mathbf{E}$  and  $\mathbf{H}$  fields are interleaved in both space and time.  $\mathbf{H}$  uses the arguments  $k + 1/2$  and  $k - 1/2$  to indicate that the  $\mathbf{H}$  field values are assumed to be located between the  $\mathbf{E}$  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.



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

Eq. (1.3a) and (1.3b) can be rearranged in an iterative algorithm:

$$E_x^{n+1/2}(k) = E_x^{n-1/2}(k) - \frac{\Delta t}{\epsilon_0 \cdot \Delta x} [H_y^n(k + 1/2) - H_y^n(k - 1/2)] \quad (1.4a)$$

$$H_y^{n+1}(k + 1/2) = H_y^n(k + 1/2) - \frac{\Delta t}{\mu_0 \cdot \Delta x} [E_x^{n+1/2}(k + 1) - E_x^{n+1/2}(k)]. \quad (1.4b)$$

Notice that the calculations are interleaved in both 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 finite-difference time-domain (FDTD) method [1].

Eqs. (1.4a) and (1.4b) are very similar, but because  $\epsilon_0$  and  $\mu_0$  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]:

$$\tilde{E} = \sqrt{\frac{\epsilon_0}{\mu_0}} E. \quad (1.5)$$

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

$$\tilde{E}_x^{n+1/2}(k) = \tilde{E}_x^{n-1/2}(k) - \frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\Delta t}{\Delta x} [H_y^n(k + 1/2) - H_y^n(k - 1/2)] \quad (1.6a)$$

$$H_y^{n+1}(k + 1/2) = H_y^n(k + 1/2) - \frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\Delta t}{\Delta x} [\tilde{E}_x^{n+1/2}(k + 1) - \tilde{E}_x^{n+1/2}(k)] \quad (1.6b)$$

Once the cell size  $\Delta x$  is chosen, then the time step  $\Delta t$  is determined by

$$\Delta t = \frac{\Delta x}{2 \cdot c_0} \quad (1.7)$$

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

$$\frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\Delta t}{\Delta x} = c_0 \cdot \frac{\Delta x/2 \cdot c_0}{\Delta x} = \frac{1}{2} \quad (1.8)$$

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

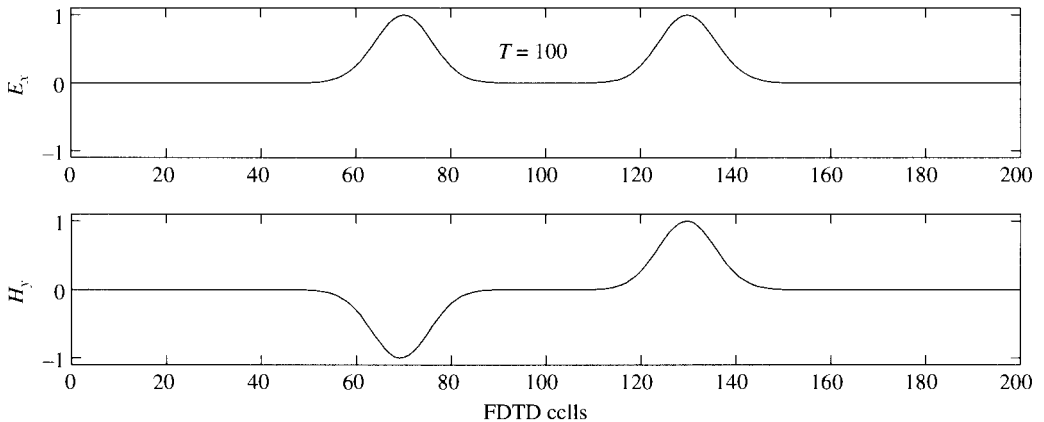
$$\text{ex}[k] = \text{ex}[k] + 0.5 * ( \text{hy}[k-1] - \text{hy}[k] ) \quad (1.9a)$$

$$\text{hy}[k] = \text{hy}[k] + 0.5 * ( \text{ex}[k] - \text{ex}[k+1] ) . \quad (1.9b)$$

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  $\text{ex}$  on the right side of the equal sign is the previous value at  $n - 1/2$ , and the  $\text{ex}$  on the left side is the new value,  $n + 1/2$ , 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 direction. The following things are worth noting about the program:

1. The  $E_x$  and  $H_y$  values are calculated by separate loops, and they 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.



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



**PROBLEM SET 1.1**

1. Get the program `fd1d_1.1.c` running. What happens when the pulse hits the end of the array? Why?
2. Modify the program so it has two sources, one at  $k_C - 20$  and one at  $k_C + 20$  (Notice that  $k_C$  is the center of the problem space). What happens when the pulses meet? Explain this from basic EM theory.
3. Instead of  $E_x$  as the source, use  $H_y$  at  $k = k_C$  as the source. What difference does it make? Try a two-point magnetic source at  $k_C - 1$  and  $k_C$  such that  $hy[k_C-1] = -hy[k_C]$ . What does this look like? What does it correspond to physically?

**1.2 STABILITY AND THE FDTD METHOD**

Let us return to the discussion of how we determine the time step. An electromagnetic 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  $\Delta t = \Delta x/c_0$ . When we get to two-dimensional simulation, we have to allow for the propagation in the diagonal direction, which brings the time requirement to  $\Delta t = \Delta x/(\sqrt{2}c_0)$ . Obviously, three-dimensional simulation requires  $\Delta t = \Delta x/(\sqrt{3}c_0)$ . This is summarized by the well-known ‘‘Courant Condition’’ [3, 4]:

$$\Delta t \leq \frac{\Delta x}{\sqrt{n} \cdot c_0}, \quad (1.10)$$

where  $n$  is the dimension of the simulation. Unless otherwise specified, throughout this book we will determine  $\Delta t$  by

$$\Delta t = \frac{\Delta x}{2 \cdot c_0}. \quad (1.11)$$

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

**PROBLEM SET 1.2**

1. In `fd1d_1.1.c`, go to the two governing equations, 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 .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 to one side. However, we have an advantage because we know there are no sources outside the problem space. Therefore, the fields at the edge must be propagating outward. We will use these two facts to estimate the value at the end by using the value next to it [5].

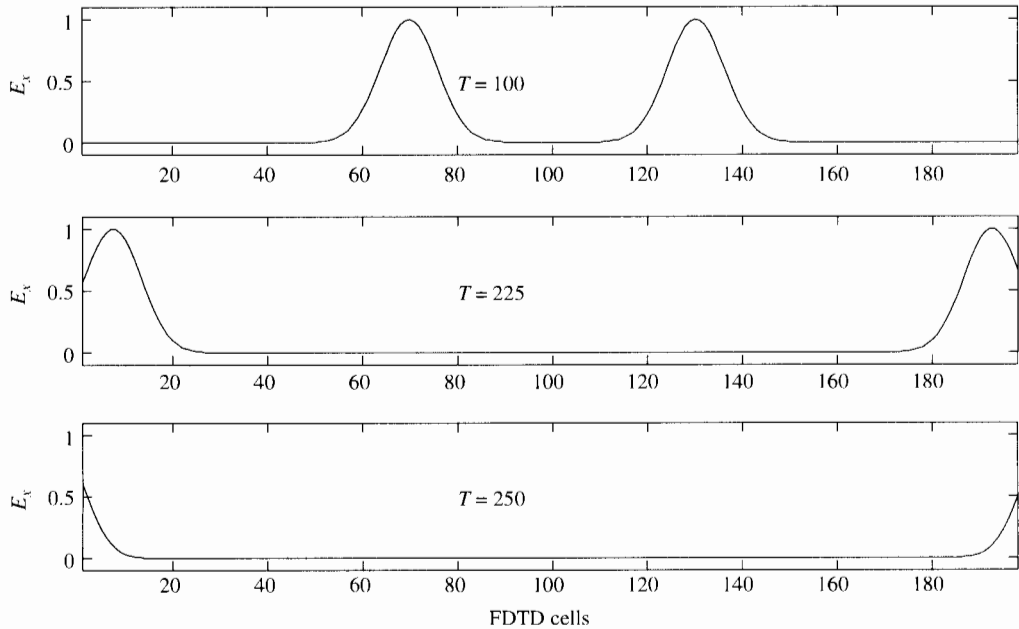
Suppose we are looking for a boundary condition at the end where  $k = 0$ . If a wave is going toward a boundary in free space, it is traveling at  $c_0$ , the speed of light. So in one time step of the FDTD algorithm, it travels

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

This equation basically explains that it takes two time steps for a wave front to cross one cell. So a common sense approach tells us that an acceptable boundary condition might be

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

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 printout labeled `fd1d_1.2.c` at the end of the chapter, the entire program hasn't been reproduced, only those parts which are different from `fd1d_1.1.c`. 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.



**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.

### 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 the boundary conditions are completely absorbing the pulse.

## 1.4 PROPAGATION IN A DIELECTRIC MEDIUM

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

$$\frac{\partial \mathbf{E}}{\partial t} = \frac{1}{\epsilon_r \epsilon_0} \nabla \times \mathbf{H} \quad (1.13a)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \mathbf{E}. \quad (1.13b)$$

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

$$\frac{\partial \tilde{E}_x(t)}{\partial t} = \frac{1}{\epsilon_r \sqrt{\epsilon_0 \mu_0}} \cdot \frac{\partial H_y(t)}{\partial z}$$

$$\frac{\partial H_y(t)}{\partial t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \cdot \frac{\partial \tilde{E}_x(t)}{\partial z},$$

and then go to the finite difference approximations

$$\frac{\tilde{E}_x^{n+1/2}(k) - \tilde{E}_x^{n-1/2}(k)}{\Delta t} = \frac{1}{\epsilon_r \sqrt{\epsilon_0 \mu_0}} \cdot \frac{H_y^n(k+1/2) - H_y^n(k-1/2)}{\Delta x} \quad (1.14a)$$

$$\frac{H_y^{n+1}(k+1/2) - H_y^n(k+1/2)}{\Delta t} = -\frac{1}{\mu_0} \frac{\tilde{E}_x^{n+1/2}(k+1) - \tilde{E}_x^{n+1/2}(k)}{\Delta x}. \quad (1.14b)$$

From the previous section

$$\frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\Delta t}{\Delta x} = \frac{1}{2},$$

so Eq. (1.14) becomes

$$\tilde{E}_x^{n+1/2}(k) = \tilde{E}_x^{n-1/2}(k) + \frac{1}{2} [H_y^n(k+1/2) - H_y^n(k-1/2)] \quad (1.15a)$$

$$H_y^{n+1}(k+1/2) = H_y^n(k+1/2) - \frac{1}{2} [\tilde{E}_x^{n+1/2}(k+1) - \tilde{E}_x^{n+1/2}(k)]. \quad (1.15b)$$

From these we can get the computer equations

$$\text{ex}[k] = \text{ex}[k] + \text{cb}[k] * (\text{hy}[k-1] - \text{hy}[k]) \quad (1.16a)$$

$$\text{hy}[k] = \text{hy}[k] + 0.5 * (\text{ex}[k] - \text{ex}[k+1]), \quad (1.16b)$$

where

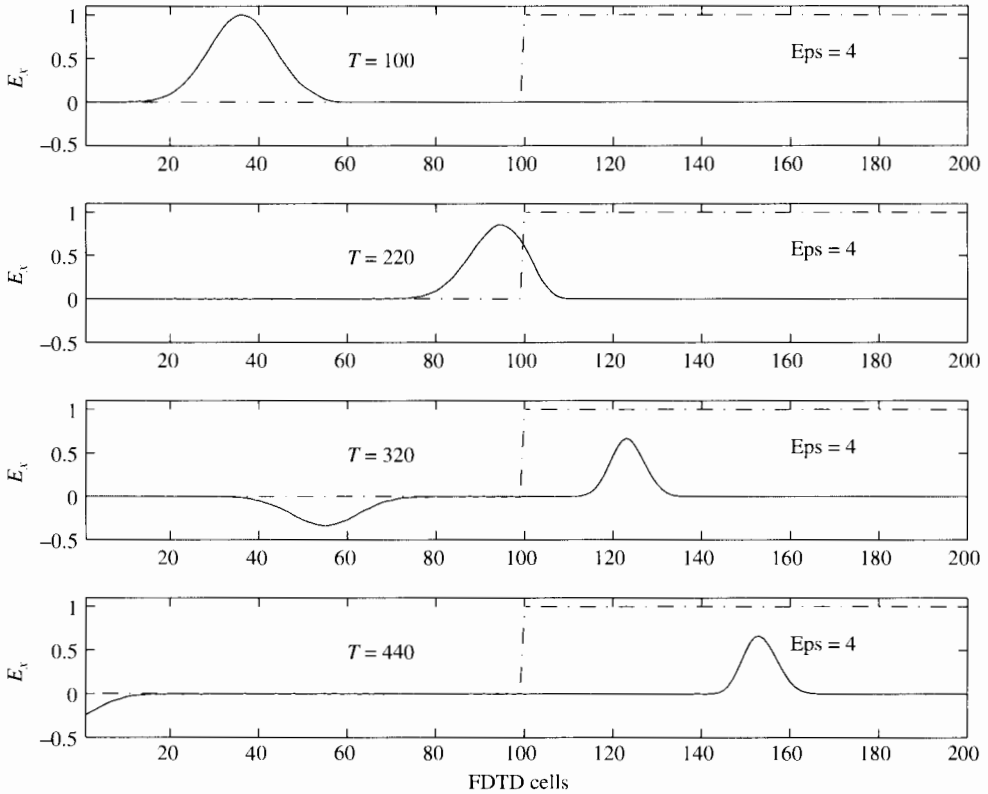
$$\text{cb}[k] = .5/\text{epsilon} \quad (1.17)$$

over those values of  $k$  which 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 a portion of the pulse propagates into the medium and a portion is reflected, in keeping with basic EM theory [6].

## 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. (See the appendix at the end of this chapter.)
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?



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

## 1.5 SIMULATING DIFFERENT SOURCES

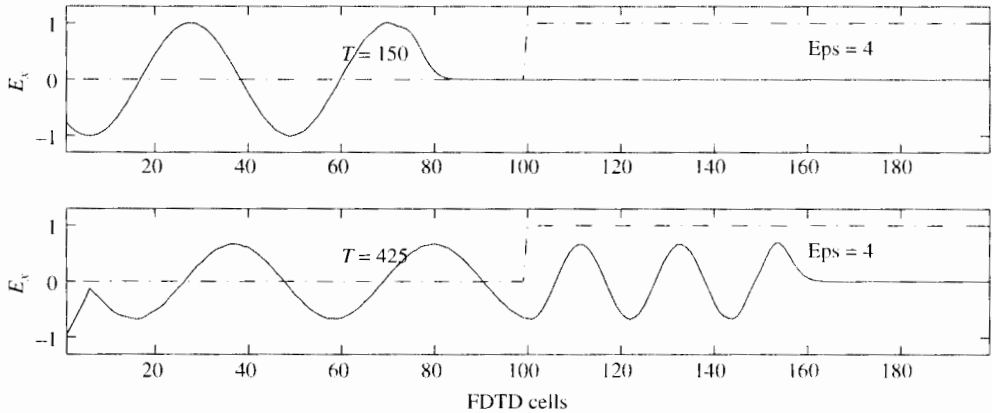
In the first two programs, a source is assigned a value to  $E_x$ ; this is referred to as a *hard source*. In `fd1d_1.3.c`, however, a value is added to  $E_x$  at a certain point; this is called a *soft source*. The reason is that with a hard source, a propagating pulse will see that value and be reflected,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

Up 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. This 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!

Note that in `fd1d_1.4.c` the cell size  $\text{d}x$  and the time step  $\text{d}t$  are specified explicitly. We do this because we need  $\text{d}t$  in the calculation of *pulse*. The cell size  $\text{d}x$  is only specified because it is needed to calculate  $\text{d}t$  from Eq. (1.7).



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

### PROBLEM SET 1.5

1. Modify your program `fd1d_1.3.c` to simulate the sinusoidal source (see `fd1d_1.4.c`).
2. Keep increasing your incident frequency from 700 MHz upward at intervals of 300 MHz. What happens?
3. A type of propagating wave function that is of great interest in areas such as optics is the “wave packet,” which is a sinusoidal function in a Gaussian envelope. 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 at 400 MHz. In free space, EM energy will propagate at the wavelength

$$\lambda_0 = \frac{c_0}{400 \text{ MHz}} = \frac{3 \times 10^8 \text{ m/sec}}{4 \times 10^8 \text{ sec}^{-1}} = .75 \text{ m.} \quad (1.18)$$

If we were only simulating free space we could choose

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

However, if we are simulating EM propagation in biological tissues, for instance, we must look at the wavelengths 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

$$\lambda_m = \frac{c_0/\sqrt{50}}{400 \text{ MHz}} = \frac{.424 \times 10^8 \text{ m/sec}}{4 \times 10^8 \text{ sec}^{-1}} = 10.6 \text{ cm,} \quad (1.19)$$

and we would probably select a cell size of one centimeter.

### PROBLEM SET 1.6

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

## 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  $\epsilon_r$ . 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:

$$\epsilon \frac{\partial \mathbf{E}}{\partial t} = \nabla \times \mathbf{H} - \mathbf{J} \quad (1.20a)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \mathbf{E}. \quad (1.20b)$$

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

$$\mathbf{J} = \sigma \cdot \mathbf{E},$$

where  $\sigma$  is the conductivity. Putting this into Eq. (1.20a) and dividing through by the dielectric constant we get

$$\frac{\partial \mathbf{E}}{\partial t} = \frac{1}{\epsilon_0 \epsilon_r} \nabla \times \mathbf{H} - \frac{\sigma}{\epsilon_0 \epsilon_r} \mathbf{E}.$$

We now revert to our simple one-dimensional equation:

$$\frac{\partial E_x(t)}{\partial t} = -\frac{1}{\epsilon_r \epsilon_0} \cdot \frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\epsilon_r \epsilon_0} E_x(t),$$

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

$$\frac{\partial \tilde{E}_x(t)}{\partial t} = -\frac{1}{\epsilon_r \sqrt{\epsilon_0 \mu_0}} \cdot \frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\epsilon_r \epsilon_0} \tilde{E}_x(t) \quad (1.21a)$$

$$\frac{\partial H_y(t)}{\partial t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \cdot \frac{\partial \tilde{E}_x(t)}{\partial z}. \quad (1.21b)$$

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

$$\frac{\tilde{E}_x^{n+1/2}(k) - \tilde{E}_x^{n-1/2}(k)}{\Delta t} = -\frac{1}{\epsilon_r \sqrt{\epsilon_0 \mu_0}} \cdot \frac{H_y^n(k+1/2) - H_y^n(k-1/2)}{\Delta x} - \frac{\sigma}{\epsilon_r \epsilon_0} \frac{\tilde{E}_x^{n+1/2}(k) + \tilde{E}_x^{n-1/2}(k)}{2}. \quad (1.22)$$

Notice that the last term in Eq. (1.21a) is approximated as the average across two time steps in Eq. (1.22). From the previous section

$$\frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\Delta t}{\Delta x} = \frac{1}{2}.$$

so Eq. (1.22) becomes

$$\begin{aligned} \tilde{E}_x^{n+1/2}(k) \left[ 1 + \frac{\Delta t \cdot \sigma}{2\epsilon_r \epsilon_0} \right] &= \tilde{E}_x^{n-1/2}(k) \left[ 1 - \frac{\Delta t \cdot \sigma}{2\epsilon_r \epsilon_0} \right] \\ &\quad - \frac{1/2}{\epsilon_r} [H_y^n(k+1/2) - H_y^n(k-1/2)] \end{aligned}$$

or

$$\tilde{E}_x^{n+1/2}(k) = \frac{\left( 1 - \frac{\Delta t \cdot \sigma}{2\epsilon_r \epsilon_0} \right)}{\left( 1 + \frac{\Delta t \cdot \sigma}{2\epsilon_r \epsilon_0} \right)} \tilde{E}_x^{n-1/2}(k) - \frac{1/2}{\epsilon_r \cdot \left( 1 + \frac{\Delta t \cdot \sigma}{2\epsilon_r \epsilon_0} \right)} [H_y^n(k+1/2) - H_y^n(k-1/2)].$$

From these we can get the computer equations

$$\text{ex}[k] = \text{ca}[k] * \text{ex}[k] + \text{cb}[k] * (\text{hy}[k-1] - \text{hy}[k]) \quad (1.23a)$$

$$\text{hy}[k] = \text{hy}[k] + 0.5 * (\text{ex}[k] - \text{ex}[k+1]), \quad (1.23b)$$

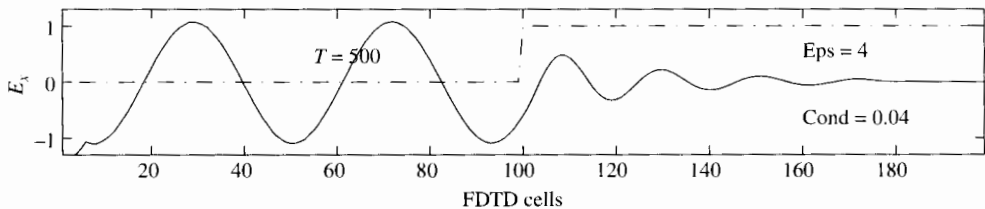
where

$$\text{eaf} = \text{dt} * \text{sigma} / (2 * \text{epsz} * \text{epsilon}) \quad (1.24a)$$

$$\text{ca}[k] = (1. - \text{eaf}) / (1. + \text{eaf}) \quad (1.24b)$$

$$\text{cb}[k] = 0.5 / (\text{epsilon} * (1. + \text{eaf})). \quad (1.24c)$$

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. This is illustrated in Fig. 1.6. Notice that the waveform in the medium is absorbed before it hits the boundary, so we don't have to worry about absorbing boundary conditions.



**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.

### 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, i.e., it is the correct amplitude going into the slab, and then it attenuates at the proper rate. You can best do this by writing a little program that calculates the parameters given in the appendix of this chapter.
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 a 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 didn't want to specify dielectric parameters, how else could you simulate metal in an FDTD program?

### APPENDIX 1.A

When a plane wave traveling in medium 1 strikes a medium 2, the fraction that is reflected is given by the reflection coefficient  $\Gamma$ , and the fraction that is transmitted into medium 2 is given by the transmission coefficient  $\tau$ . These are determined by the intrinsic impedances  $\eta_1$  and  $\eta_2$  of the respective media [6, p. 398]:

$$\Gamma = \frac{E_{ref}}{E_{inc}} = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1} \quad (1.A.1)$$

$$\tau = \frac{E_{trans}}{E_{inc}} = \frac{2\eta_2}{\eta_2 + \eta_1}. \quad (1.A.2)$$

The impedances are given by

$$\eta = \sqrt{\frac{\mu}{\epsilon_0 \epsilon_r^*}} \quad (1.A.3)$$

where  $\epsilon_r^*$  is the complex relative dielectric constant

$$\epsilon_r^* = \epsilon_r + \frac{\sigma}{j\omega\epsilon_0}.$$

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

$$\Gamma = \frac{\frac{1}{\sqrt{\epsilon_2^*}} - \frac{1}{\sqrt{\epsilon_1^*}}}{\frac{1}{\sqrt{\epsilon_2^*}} + \frac{1}{\sqrt{\epsilon_1^*}}} = \frac{\sqrt{\epsilon_1^*} - \sqrt{\epsilon_2^*}}{\sqrt{\epsilon_1^*} + \sqrt{\epsilon_2^*}} \quad (1.A.4)$$

$$\tau = \frac{2/\sqrt{\epsilon_2^*}}{\frac{1}{\sqrt{\epsilon_2^*}} + \frac{1}{\sqrt{\epsilon_1^*}}} = \frac{2 \cdot \sqrt{\epsilon_1^*}}{\sqrt{\epsilon_1^*} + \sqrt{\epsilon_2^*}}. \quad (1.A.5)$$

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

$$E_x = E_0 e^{-\alpha z} e^{-j\beta z},$$

where  $E_0$  is the amplitude at  $z = 0$ . The parameters  $\alpha$  and  $\beta$  are determined by the dielectric constant  $\epsilon_r$ , the conductivity  $\sigma$ , and the radian frequency  $\omega = 2\pi f$  of the propagating wave, via the following two formulas [6, p. 420]:

$$\alpha = \frac{\omega}{c_0} \sqrt{\frac{\epsilon_r}{2}} \left[ \sqrt{1 + \left( \frac{\sigma}{\omega\epsilon_0\epsilon_r} \right)^2} - 1 \right]^{1/2} \quad (\text{Np/m}) \quad (1.A.6)$$

$$\beta = \frac{\omega}{c_0} \sqrt{\frac{\epsilon_r}{2}} \left[ \sqrt{1 + \left( \frac{\sigma}{\omega\epsilon_0\epsilon_r} \right)^2} + 1 \right]^{1/2} \quad (\text{rad/m}). \quad (1.A.7)$$

### REFERENCES

- [1] K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antennas and Propagat.*, vol. 17, 1966, pp. 585–589.
- [2] A. Taflov and M. Brodwin, Numerical solution of steady state electromagnetic scattering problems using the time-dependent Maxwell's equations, *IEEE Trans. Microwave Theory Tech.*, vol. 23, 1975, pp. 623–730.



- [3] A. Taflové, *Computation Electrodynamics: The Finite-Difference Time-Domain Method*. 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. Compat.*, vol. 23, 1981, pp. 377–384.
- [6] D. K. Cheng, *Field and Wave Electromagnetics*, Menlo Park, CA: Addison-Wesley, 1992.

```

/* FD1D_1.1.c. 1D FDTD simulation in free space */

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

#define KE 200 /* KE is the number of cells to be used */

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

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

    kc = KE/2; /* Center of the problem space */
    t0 = 40.0; /* Center of the incident pulse */
    spread = 12; /* Width of the incident pulse */
    T = 0;
    NSTEPS = 1;

    while ( NSTEPS > 0 ) {
        printf( "NSTEPS --> "); /* NSTEPS is the number of times the */
        scanf("%d", &NSTEPS); /* main loop has executed */
        printf("%d \n", NSTEPS);
        n= 0;

        for ( n=1; n <=NSTEPS ; n++)
        {
            T = T + 1; /* T keeps track of the total number */
            /* of times the main loop is executed */

            /* 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`cn",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];
        int n,k,kc,ke,kstart,nsteps;
        float ddx,dt,T,epsz,epsilon,sigma,eaf;
        float cb[KE];
    .
    .
    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 */

float ddx, dt;
float freq_in;

    ddx = .01;                /* Set the cell size to 1 cm */
    dt = ddx/(2*3e8)         /* Calculate the time step */

/* These parameters specify the input pulse */
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 sinusoidal source at cell 5 p0
*/

        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. 1D FDTD simulation of a lossy dielectric medium */

float ca[KE],cb[KE];

    epsz = 8.85419e-12;

for ( k=1; k <= KE; k++ ) { /* Initialize to free space */
    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 in order 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 very apparent in this chapter how closely signal processing is linked to time-domain EM simulation. This will be most obvious in section 2.4, where we use Z transforms to simulate complicated media.

### 2.1 REFORMULATION USING THE FLUX DENSITY

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

$$\frac{\partial \mathbf{D}}{\partial t} = \nabla \times \mathbf{H} \quad (2.1a)$$

$$\mathbf{D}(\omega) = \epsilon_0 \cdot \epsilon_r^*(\omega) \cdot \mathbf{E}(\omega) \quad (2.1b)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \mathbf{E}, \quad (2.1c)$$

where  $\mathbf{D}$  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

$$\tilde{\mathbf{E}} = \sqrt{\frac{\epsilon_0}{\mu_0}} \cdot \mathbf{E} \quad (2.2a)$$

$$\tilde{\mathbf{D}} = \sqrt{\frac{1}{\epsilon_0 \cdot \mu_0}} \cdot \mathbf{D}, \quad (2.2b)$$



which leads to

$$\frac{\partial \tilde{\mathbf{D}}}{\partial t} = \frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \mathbf{H} \quad (2.3a)$$

$$\tilde{\mathbf{D}}(\omega) = \varepsilon_r^*(\omega) \cdot \tilde{\mathbf{E}}(\omega) \quad (2.3b)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\sqrt{\varepsilon_0 \mu_0}} \nabla \times \tilde{\mathbf{E}}. \quad (2.3c)$$

We saw in Chapter 1 that this form of Eqs. (2.3a) and (2.3c) will lead to the very simple finite difference equations, Eq. (1.4a) and (1.4b). 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 get it from the frequency domain to the time domain. We will assume we are dealing with a lossy dielectric medium of the form

$$\varepsilon_r^*(\omega) = \varepsilon_r + \frac{\sigma}{j\omega\varepsilon_0} \quad (2.4)$$

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

$$D(\omega) = \varepsilon_r \cdot E(\omega) + \frac{\sigma}{j\omega\varepsilon_0} E(\omega). \quad (2.5)$$

Taking the first term into the time domain is not a problem because it is 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

$$D(t) = \varepsilon_r \cdot E(t) + \frac{\sigma}{\varepsilon_0} \int_0^t E(t') \cdot dt'.$$

We will want to go to the sampled time domain, so the integral will be approximated as a summation over the time steps  $\Delta t$ :

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

Note that  $E$  and  $D$  are specified at time  $t = n \cdot \Delta t$ . There is one problem remaining: looking back at Eq. (2.3b), we see that we have to solve for  $E^n$  given the value  $D^n$ . Unfortunately, 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:

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

Now we can calculate  $E^n$  from

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

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

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

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

$$E^n = \frac{D^n - I^{n-1}}{\epsilon_r + \frac{\sigma \cdot \Delta t}{\epsilon_0}} \quad (2.8a)$$

$$I^n = I^{n-1} + \frac{\sigma \cdot \Delta t}{\epsilon_0} E^n. \quad (2.8b)$$

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 value of  $E^n$  from 0 to  $n$ ! Now the entire FDTD formulation is

$$dx[k] = dx[k] + .5 * ( hy[k-1] - hy[k] ) \quad (2.9a)$$

$$ex[k] = gax[k] * ( dx[k] - ix[k] ) \quad (2.9b)$$

$$ix[k] = ix[k] + gbx[k] * ex[k] \quad (2.9c)$$

$$hy[k] = hy[k] + .5 * ( ex[k] - ex[k+1] ) \quad (2.9d)$$

where

$$gax[k] = 1 / (epsilon + (sigma*dt/epsz)) \quad (2.10a)$$

$$gbx[k] = sigma*dt/epsz. \quad (2.10b)$$

The important point is this: all of the information regarding the media is contained in Eqs. (2.9b) and (2.9c). For free space,  $gax = 1$  and  $gbx = 0$ .; for lossy material,  $gax$  and  $gbx$  are calculated according to Eqs. (2.10a) and (2.10b). In calculating  $ex[k]$  at the point  $k$ , it uses only values of  $dx[k]$  and previous values of  $ex[k]$  in the time domain. Equations (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'll 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

Up 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 we observe that a steady state has been 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. System theory tells us that 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 slab. 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 very 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 the Fourier transform of the data could be taken, presumably using a fast Fourier transform algorithm. This presents a logistical nightmare.

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

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

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

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

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

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

which may be implemented in computer code by

$$\text{real\_pt}[m,k] = \text{real\_pt}[m,k] + \text{ex}[k] \cdot \cos(2\pi \cdot \text{freq}(m) \cdot \text{dt} \cdot n) \quad (2.14a)$$

$$\text{imag\_pt}[m,k] = \text{imag\_pt}[m,k] + \text{ex}[k] \cdot \sin(2\pi \cdot \text{freq}(m) \cdot \text{dt} \cdot n). \quad (2.14b)$$

For every point  $k$ , in the region of interest, we require only two buffers for every frequency of interest  $f_m$ . At any point  $k$ , from the real part of  $E(f_i)$ ,  $\text{real\_pt}[m,k]$ , and the imaginary part  $\text{imag\_pt}[m,k]$ , we can determine the amplitude and phase at the frequency  $f_m$ :

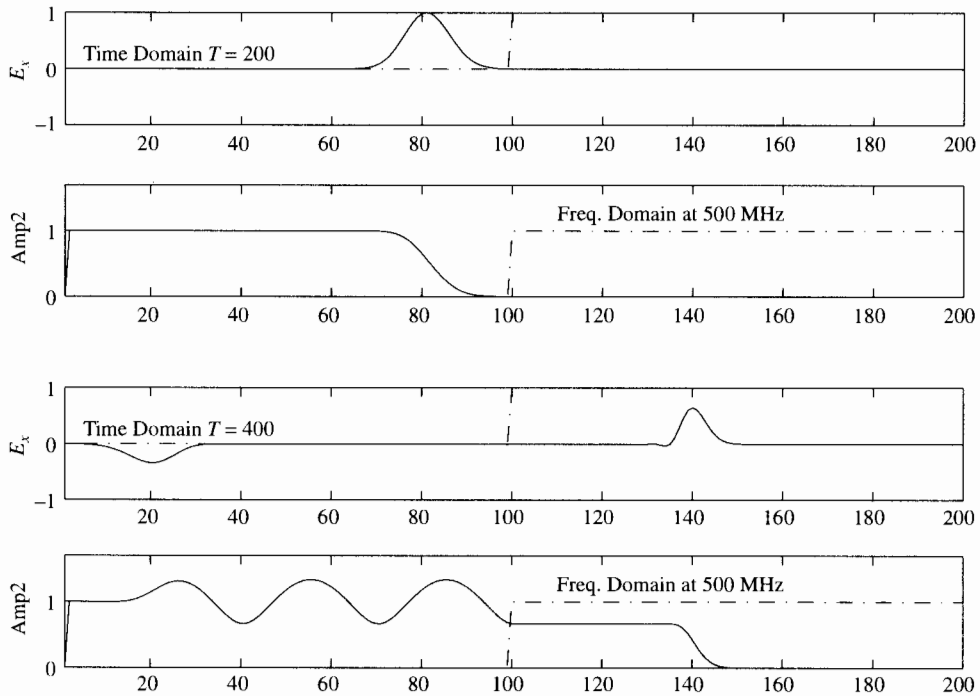
$$\text{amp}[m,k] = \text{sqrt}(\text{pow}(\text{real\_pt}[m,k], 2.) + \text{pow}(\text{imag\_pt}[m,k], 2.)) \quad (2.15a)$$

$$\text{phase}[m,k] = \text{atan2}(\text{imag\_pt}[m,k], \text{real\_pt}[m,k]). \quad (2.15b)$$

Note that there is an amplitude and phase associated with every frequency at each cell [1, 2]. The program `fd1d_2.2.c` at the end of the chapter calculates the 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 displayed. At  $T = 200$ , before the pulse has hit the medium, the frequency response is 1 through that part of the space where the pulse has traveled. After 400 time steps, the pulse has hit the medium, and some of it 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) at the end of Chapter 1,

$$\tau = \frac{\sqrt{4} \cdot 1}{1 + \sqrt{4}} = .667,$$

which is the Fourier amplitude in the medium. The Fourier amplitude outside the medium varies between  $1 - .333$  and  $1 + .333$ . This is in keeping with the pattern formed by the standing wave that is created from a sinusoidal signal, whose reflected wave is interacting with the original incident wave.



**Figure 2.1** Simulation of a pulse striking a dielectric medium with  $\epsilon_r = 4$ . The top figure is 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. After 400 time steps, the pulse has struck the medium, and part of it has been transmitted and part reflected. The Fourier amplitude in the medium is 0.667, which is the percentage that has been transmitted.

## 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.

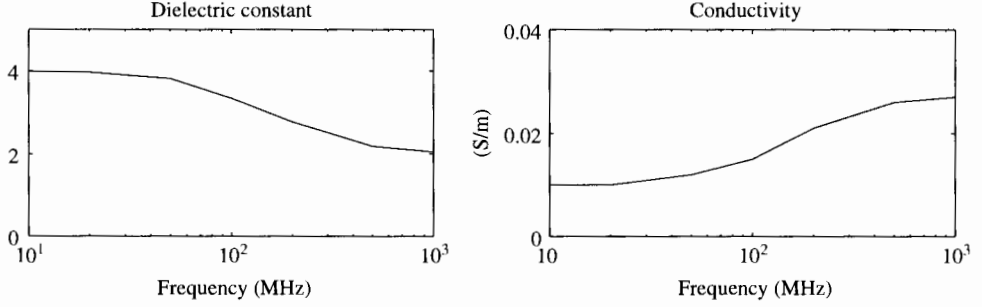
## 2.3 FREQUENCY-DEPENDENT MEDIA

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 was 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 of 10 to 1000 MHz as shown in Fig. 2.2. A material like this can be adequately represented by the following formulation:

$$\epsilon_r^*(\omega) = \epsilon_r + \frac{\sigma}{j\omega\epsilon_0} + \frac{\chi_1}{1 + j\omega t_0}. \quad (2.16)$$

This is referred to as the Debye formulation. In this formulation, there is a dielectric constant  $\epsilon_r$  and a conductivity  $\sigma$ , but there is also a frequency-dependent term. The following parameters



**Figure 2.2** Relative dielectric constant and conductivity as functions of frequency for a Debye medium with the following properties:  $\epsilon_r = 2$ ,  $\sigma = .1$ ,  $\chi = 2$ ,  $\tau = .001 \mu\text{sec}$ .

represent the medium of Fig. 2.2:

$$\epsilon_r = 2, \quad \sigma = .01, \quad \chi_1 = 2, \quad t_0 = .001 \mu \text{ sec}.$$

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

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

The inverse Fourier transform of the Debye term is  $(\chi_1/t_0)e^{-(t/t_0)}u(t)$ , where  $u(t)$  is the Heavyside, or rectangular function, which is 0 for  $t < 0$  and 1 thereafter. (Remember we said that implicit to FDTD is the fact that all functions are causal, i.e., zero for  $t < 0$ , because our computer programs initialize the field values to zero.) Equation (2.17) in the frequency domain becomes the convolution

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

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

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

Notice that

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

Substituting this value into Eq. (2.18) above gives

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

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

$$\begin{aligned} D^n &= \epsilon_r \cdot E^n + I^n + S^n \\ &= \epsilon_r \cdot E^n + \left[ \frac{\sigma \cdot \Delta t}{\epsilon_0} \cdot E^n + I^{n-1} \right] + \left[ \chi_1 \cdot \frac{\Delta t}{t_0} \cdot E^n + e^{-t/t_0} \cdot S^{n-1} \right], \end{aligned} \quad (2.20)$$

and solving for  $E^n$

$$E^n = \frac{D^n - I^{n-1} - e^{-\Delta t/t_0} S^{n-1}}{\epsilon_r + \frac{\sigma \cdot \Delta t}{\epsilon_0} + \chi_1 \cdot \frac{\Delta t}{t_0}} \quad (2.21a)$$

$$I^n = I^{n-1} + \frac{\sigma \cdot \Delta t}{\epsilon_0} \cdot E^n \quad (2.21b)$$

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

This formulation is implemented by the following computer code:

$$dx[k] = dx[k] + .5 * ( hy[k-1] - hy[k] ) \quad (2.22a)$$

$$ex[k] = gax[k] * ( dx[k] - ix[k] - del\_exp * sx[k] ) \quad (2.22b)$$

$$ix[k] = ix[k] + gbx[k] * ex[k] \quad (2.22c)$$

$$sx[k] = del\_exp * sx[k] + gbc[k] * ex[k] \quad (2.22d)$$

$$hy[k] = hy[k] + .5 * ( ex[k] - ex[k+1] ) \quad (2.22e)$$

where

$$gax[k] = 1 / ( epsr + ( sigma * dt / epsz ) + ( chi1 * dt / t0 ) ) \quad (2.23a)$$

$$gbx[k] = sigma * dt / epsz \quad (2.23b)$$

$$gbc[k] = chi1 * dt / t0 \quad (2.23c)$$

and

$$del\_exp = \exp(-dt/t0).$$

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

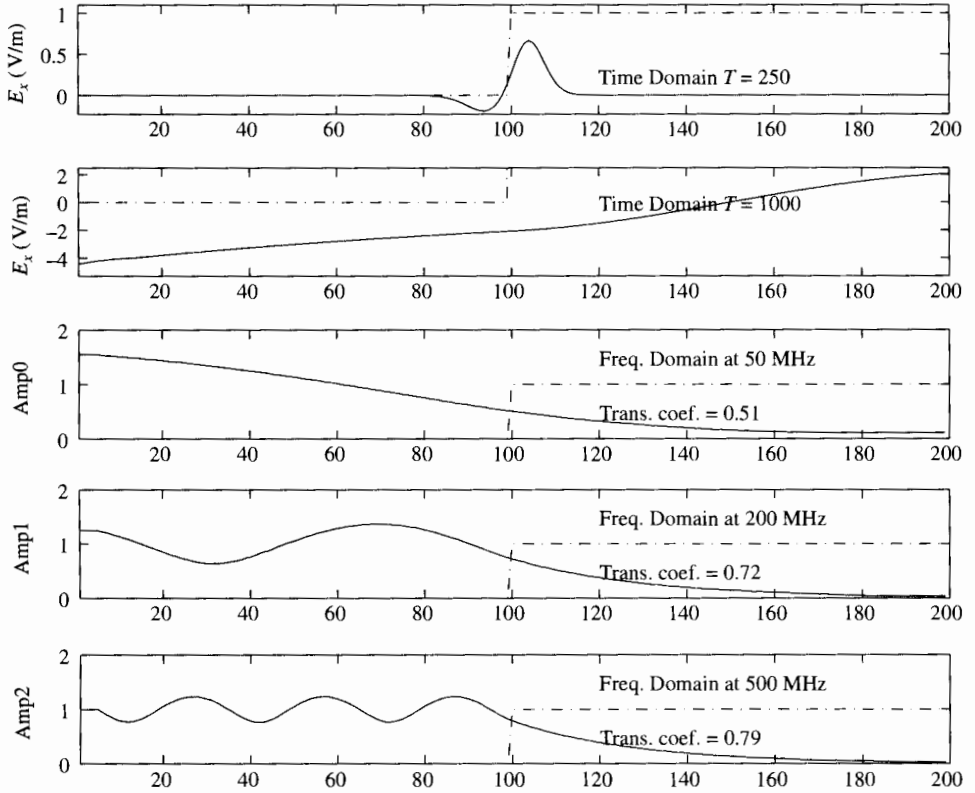
The program `fd1d_2.3.c` calculates the frequency domain amplitude and phase for three frequencies. Figure 2.3 shows a simulation of a pulse going into a frequency-dependent dielectric material with the properties

$$\epsilon_r = 2, \quad \sigma = .01, \quad \chi_1 = 2, \quad t_0 = .001 \mu \text{ sec.}$$

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

Frequency (MHz)	$\epsilon_r$	$\sigma$ (S/m)
50	6.55	.024
200	3.94	.047
500	2.46	.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 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:  $\epsilon_r = 2$ ,  $\sigma = 0.01$ ,  $\chi_1 = 2$ ,  $t_0 = .001 \mu\text{sec}$ . After 250 time steps (top), the pulse has struck the medium, and part of it has been transmitted and part reflected. After 1000 steps (second from top) the pulse has penetrated into the medium, but has spread. 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). There is another method, usually referred to as the auxiliary differential equation (ADE) method [4, 5]. Let's look at Eq. (2.17), but rewrite it in the following manner:

$$(1 + j\omega t_0) S(\omega) = \chi_1 E(\omega). \quad (2.24)$$

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, so (2.24) becomes

$$s(t) + t_0 \frac{ds(t)}{dt} = \chi_1 e(t). \quad (2.25)$$

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

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

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:

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

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

Putting the two together gives

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

In this case

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

so we have

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

This leaves only the following question: How do we know that  $\Delta t/t_0$  is small enough? Recall earlier that the cell size has to be small enough to get about ten 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've been studying the Z transform theory in the appendix, or if you are already familiar with Z transforms, you will now see the advantage of using Z transforms for the FDTD formulation of frequency-dependent media [6]. Returning to the problem of calculating  $E$  in a Debye media, we begin with our frequency domain equation

$$D(\omega) = \left( \epsilon_r + \frac{\sigma}{j\omega\epsilon_0} + \frac{\chi_1}{1 + j\omega t_0} \right) \cdot E(\omega).$$

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

$$D(z) = \epsilon_r \cdot E(z) + \frac{\sigma \cdot \Delta t/\epsilon_0}{1 - z^{-1}} \cdot E(z) + \frac{\chi_1 \cdot \Delta t/t_0}{1 + z^{-1}} \cdot E(z). \quad (2.28)$$



Note that the factor  $\Delta t$ , which is the time step, had to be added to the last two terms in going from the time domain to the Z domain. (See section A.2 in the appendix at the end of book). Similar to what we did above, we will define some auxiliary parameters:

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

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

Equation (2.28) then becomes

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

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

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

Here is the advantage of the Z transform formulation: to get to the sampled time domain, replace  $E(z)$  with  $E^n$ ,  $z^{-1} E(z)$  with  $E^{n-1}$ , and make a similar replacement with the other parameters in Eqs. (2.31), (2.29a), and (2.29b). What you get is

$$E^n = \frac{D^n - I^{n-1} - e^{-\Delta t / t_0} S^{n-1}}{\varepsilon_r + \frac{\sigma \cdot \Delta t}{\varepsilon_0} + \frac{\chi_1 \cdot \Delta t}{t_0}} \quad (2.32a)$$

$$I^n = I^{n-1} + \frac{\sigma \cdot \Delta t}{\varepsilon_0} E^n \quad (2.32b)$$

$$S^n = e^{-\Delta t / t_0} S^{n-1} + \frac{\chi_1 \cdot \Delta t}{t_0} E^n, \quad (2.32c)$$

which is *exactly* what we got in the previous section. The difference is, we didn't have to do 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 an Unmagnetized Plasma

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

$$\varepsilon^*(\omega) = 1 + \frac{\omega_p^2}{\omega(j\nu_c - \omega)}, \quad (2.33)$$

where  $\omega_p = 2\pi f_p$   
 $f_p$  is the plasma frequency  
 $\nu_c$  is the electron collision frequency.

By using partial fraction expansion, Eq. (2.33) can be written as

$$\varepsilon^*(\omega) = 1 + \frac{\omega_p^2 / \nu_c}{j\omega} - \frac{\omega_p^2 / \nu_c}{\nu_c + j\omega}. \quad (2.34)$$

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's start by taking the Z transform of (2.34) to obtain

$$\varepsilon^*(z) = \frac{1}{\Delta t} + \frac{\omega_p^2/v_c}{1-z^{-1}} - \frac{\omega_p^2/v_c}{1-e^{-v_c \cdot \Delta t} z^{-1}}. \quad (2.35)$$

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

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

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

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

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

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

$E(z)$  can be solved for by

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

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

Therefore, the FDTD simulation becomes

$$\text{ex}[k] = \text{dx}[k] - \text{sx}[k]; \quad (2.38a)$$

$$\begin{aligned} \text{sx}[k] &= (1 + \exp(-v_c \cdot \text{dt})) * \text{sxm1}[k] \\ &\quad - \exp(-v_c \cdot \text{dt}) * \text{sxm2}[k] \\ &\quad + (\text{pow}(\text{omega}, 2) * \text{dt}/v_c) * (1 - \exp(-v_c \cdot \text{dt})) * \text{ez}[k]; \end{aligned} \quad (2.38b)$$

$$\text{sxm2}[k] = \text{sxm1}[k]; \quad (2.38c)$$

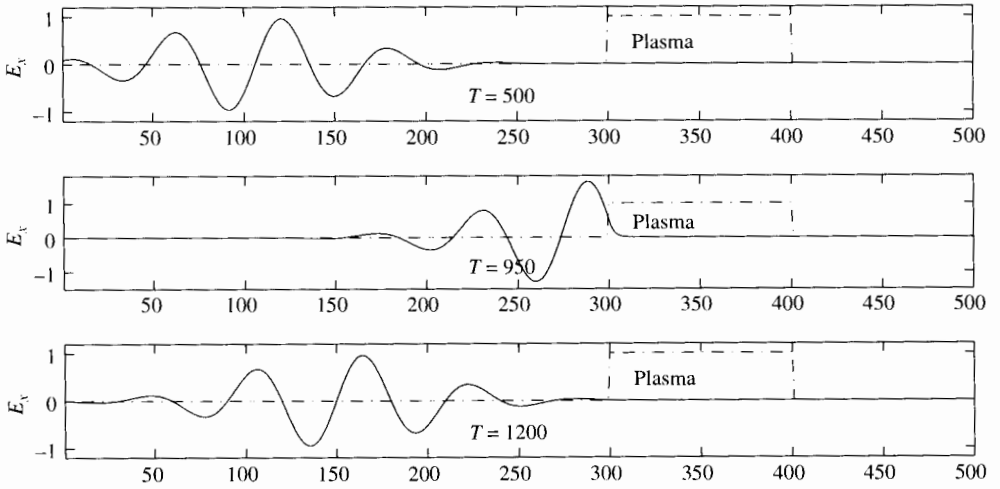
$$\text{sxm1}[k] = \text{sx}[k]; \quad (2.38d)$$

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

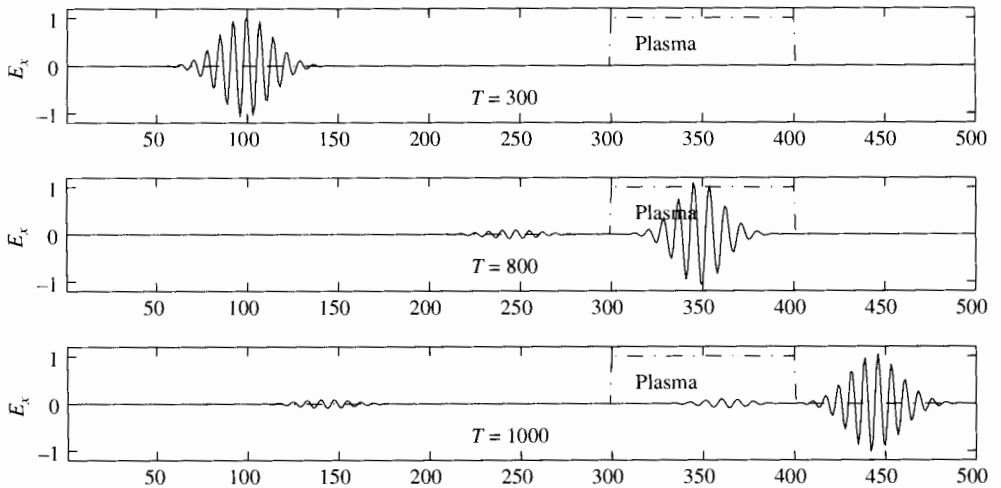
We will do a simulation of a pulse propagating in free space that comes upon a plasma. Plasma is a very interesting medium. At relatively low frequencies, it looks like a metal, and at higher frequencies, it becomes transparent. By lower or higher frequencies, I mean below or above the plasma resonance frequency  $f_p$ . This simulation will use the properties of silver:  $v_c = 57$  THz,  $f_p = 2000$  THz. 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  $\Rightarrow 10^{12}$ ). 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 ten points per wavelength, a cell size of one nanometer, i.e.,  $\Delta x = 10^{-9}$ , 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 it interacts with the plasma almost as if it were a metal barrier and 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.



**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.



**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.

### PROBLEM SET 2.4

1. Modify the program `fd1d_2.3.c` to simulate plasma and duplicate the results of Figs. 2.4 and 2.5. This is much easier than it might look. First change your cell size to 1 nanometer. Then replace the calculation of the  $E$  field for a lossy Debye medium with that of Eqs. (2.38). After doing the simulations at 500 THz 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?

### 2.5 FORMULATING A LORENTZ MEDIUM

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

$$\epsilon_r^*(\omega) = \epsilon_r + \frac{\epsilon_1}{1 + j2\delta_0 \left( \frac{\omega}{\omega_0} \right) - \left( \frac{\omega}{\omega_0} \right)^2} \tag{2.39}$$

Figure 2.6 is a graph of the dielectric constant and conductivity of a material with the following Lorentz parameters:  $\epsilon_r = 2$ ,  $\epsilon_1 = 2$ ,  $f_0 = 100$  MHz,  $\delta = .25$  ( $\omega_0 = 2\pi f_0$ ). To simulate a Lorentz medium in the FDTD formulation, we first put Eq. (2.39) into Eq. (2.3b),

$$\begin{aligned} D(\omega) &= \epsilon_r E(\omega) + \frac{\epsilon_1}{1 + j2\delta_0 \left( \frac{\omega}{\omega_0} \right) - \left( \frac{\omega}{\omega_0} \right)^2} E(\omega) \\ &= \epsilon_r E(\omega) + S(\omega), \end{aligned} \tag{2.40}$$

where we have again defined an auxiliary term,

$$S(\omega) = \frac{\omega_0^2 \epsilon_1}{\omega_0^2 + j\omega 2\delta_0 \omega_0 + (j\omega)^2} E(\omega).$$

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

$$(\omega_0^2 + j\omega 2\delta_0 \omega_0 + (j\omega)^2) S(\omega) = \omega_0^2 \epsilon_1 E(\omega).$$

Finally, we proceed to the finite difference approximations

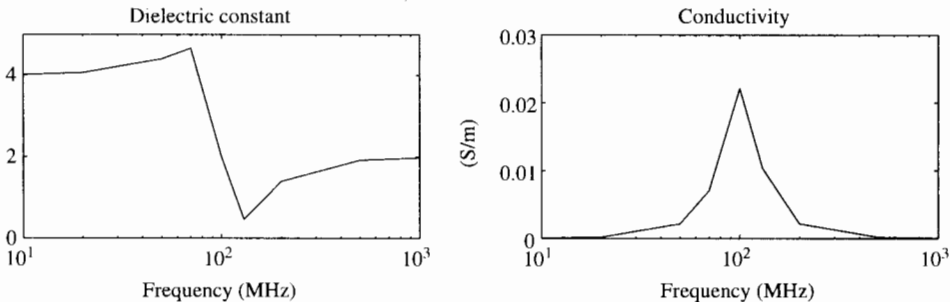
$$\omega_0^2 S^{n-1} + 2\delta_0 \omega_0 \frac{S^n - S^{n-2}}{2\Delta t} + \frac{S^n - 2S^{n-1} + S^{n-2}}{\Delta t^2} = \omega_0^2 \epsilon_1 E^{n-1}.$$

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

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

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

$$\frac{ds(t)}{dt} \cong \frac{S^n - S^{n-2}}{2\Delta t}.$$



**Figure 2.6** Relative dielectric constant and conductivity as functions of frequency for a Lorentz medium with the following properties:

$$\epsilon_r = 2, \epsilon_1 = 2, f_0 = 100 \text{ MHz}, \delta = .25.$$

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

$$\begin{aligned}
 S^n \left( \frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right) + S^{n-1} \left( \omega_0^2 - \frac{2}{\Delta t^2} \right) + S^{n-2} \left( -\frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right) &= \omega_0^2 \varepsilon_1 E^{n-1} \\
 S^n &= -\frac{\left( \omega_0^2 - \frac{2}{\Delta t^2} \right)}{\left( \frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right)} S^{n-1} - \frac{\left( -\frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right)}{\left( \frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right)} S^{n-2} + \frac{\omega_0^2 \varepsilon_1}{\left( \frac{\delta_0 \omega_0}{\Delta t} + \frac{1}{\Delta t^2} \right)} E^{n-1} \\
 S^n &= \frac{(2 - \Delta t^2 \omega_0^2)}{(1 + \Delta t \delta_0 \omega_0)} S^{n-1} - \frac{(1 - \Delta t \delta_0 \omega_0)}{(1 + \Delta t \delta_0 \omega_0)} S^{n-2} + \frac{\Delta t^2 \omega_0^2 \varepsilon_1}{(1 + \delta_0 \omega_0 \Delta t)} E^{n-1}. \tag{2.41}
 \end{aligned}$$

Now let's go back to Eq. (2.40) and take it to the sampled time domain

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

and since we are interested in solving for  $E$ , we'll write it as

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

Notice that we already have a solution for  $S^n$  in Eq. (2.41). And since we need only the previous value of  $E$ , i.e.,  $E^{n-1}$ , we are in business! We don't need to worry about expanding the auxiliary term to make sure they don't require values of  $E$  that we don't have yet.

Let's 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

$$\begin{aligned}
 \gamma &= \frac{\omega_0}{\sqrt{1 - \delta_0^2}} \\
 \alpha &= \delta_0 \omega_0 \\
 \beta &= \omega_0 \sqrt{1 - \delta_0^2}.
 \end{aligned}$$

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

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

The corresponding sampled time domain equation is

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

Well and good. But if both methods are correct, they should both yield the same answers, which means the corresponding terms in Eqs. (2.41) and (2.43) should be the same. Let's see if that's true.

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

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

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

$$\begin{aligned} e^{-\alpha \cdot \Delta t} \sin(\beta \cdot \Delta t) \cdot \gamma \cdot \Delta t &= \frac{\beta \cdot \Delta t}{1 + \alpha \cdot \Delta t} \cdot \gamma \cdot \Delta t \\ &= \frac{\left( \omega_0 \sqrt{1 - \delta_0^2} \cdot \Delta t \right)}{1 + \delta_0 \omega_0 \cdot \Delta t} \cdot \frac{\omega_0}{\sqrt{1 - \delta_0^2}} \cdot \Delta t \cong \frac{\omega_0^2 \cdot \Delta t^2}{1 + \delta_0 \omega_0 \Delta t}. \end{aligned}$$

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

$$\begin{aligned} 2e^{-\alpha \cdot \Delta t} \cos(\beta \cdot \Delta t) &\cong 2 \left( \frac{1}{1 + \alpha \cdot \Delta t} \right) \left( 1 - \frac{(\beta \cdot \Delta t)^2}{2} \right) \\ &= \frac{2 - (\beta \cdot \Delta t)^2}{1 + \alpha \cdot \Delta t} = \frac{2 - \omega_0^2 \cdot (1 - \delta_0^2) \cdot \Delta t^2}{1 + \omega_0 \delta_0 \cdot \Delta t}. \end{aligned}$$

The last step is perhaps our weakest one. It depends upon  $\delta$  being small compared to 1 so  $\delta^2$  will be negligible.

### 2.5.1 Simulation of Human Muscle Tissue

We will end 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:

$$\varepsilon_r^*(\omega) = \varepsilon_r + \frac{\sigma}{j\omega\varepsilon_0} + \varepsilon_1 \frac{\omega_0}{(\omega_0^2 + \alpha^2) + j2\alpha\omega + \omega^2}. \quad (2.44)$$

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

$$D(z) = \varepsilon_r E(z) + \frac{\sigma \cdot \Delta t / \varepsilon_0}{1 - z^{-1}} \cdot E(z) + \varepsilon_1 \frac{e^{-\alpha \cdot \Delta t} \cdot \sin(\omega_0 \cdot \Delta t) \cdot \Delta t \cdot z^{-1}}{1 - 2e^{-\alpha \cdot \Delta t} \cos(\omega_0 \cdot \Delta t) \cdot z^{-1} + e^{-2\alpha \Delta t} z^{-2}} \cdot E(z). \quad (2.45)$$

We will define two auxiliary parameters:

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

$$S(z) = \frac{\varepsilon_1 \cdot e^{-\alpha \cdot \Delta t} \cdot \sin(\omega_0 \cdot \Delta t) \cdot \Delta t \cdot E(z)}{1 - 2e^{-\alpha \cdot \Delta t} \cos(\omega_0 \cdot \Delta t) \cdot z^{-1} + e^{-2\alpha \cdot \Delta t} z^{-2}}. \quad (2.46b)$$

Now Eq. (2.45) becomes

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

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

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

$$\begin{aligned} S(z) &= 2e^{-\alpha \cdot \Delta t} \cos(\omega_0 \cdot \Delta t) \cdot z^{-1} S(z) \\ &\quad - e^{-2\alpha \cdot \Delta t} z^{-2} S(z) + \varepsilon_1 \cdot e^{-\alpha \cdot \Delta t} \cdot \sin(\omega_0 \cdot \Delta t) \cdot \Delta t \cdot E(z). \end{aligned} \quad (2.48b)$$

Remember that in calculating  $E(z)$ , we need the previous value of  $S(z)$  (as indicated by  $z^{-1} S(z)$ ), the present value of  $D(z)$ , and the present value of  $I(z)$ . The present value of  $D(z)$

is not a problem because, in the order in which the algorithm is implemented, it has already been calculated in Eq. (2.1a). However, we will need the expanded version of  $I(z)$ . Equation (2.47) becomes

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

from which  $E(z)$  can be calculated by

$$E(z) = \frac{D(z) - z^{-1} I(z) - z^{-1} S(z)}{\varepsilon_r + \sigma \cdot \Delta t / \varepsilon_0} \varepsilon_r E(z) \quad (2.49a)$$

$$I(z) = z^{-1} I(z) + \frac{\sigma \cdot \Delta t}{\varepsilon_0} \cdot E(z) \quad (2.49b)$$

$$S(z) = 2e^{-\alpha \cdot \Delta t} \cos(\omega_0 \cdot \Delta t) \cdot z^{-1} S(z) - e^{-2\alpha \cdot \Delta t} z^{-2} S(z) + \varepsilon_1 \cdot e^{-\alpha \cdot \Delta t} \cdot \sin(\omega_0 \cdot \Delta t) \cdot \Delta t \cdot E(z). \quad (2.49c)$$

Note that  $S(z)$  did not have to be expanded out because it already had a  $z^{-1}$  in the numerator.

### 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. In light of 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 1A.
3. FDTD simulation has been used extensively to model the effects of electromagnetic radiation on human tissue both for safety [8] and for therapeutic applications [9]. Human muscle tissue is highly frequency dependent. Table 2.1 shows how the dielectric constant and conductivity of muscle vary with frequency [10]. Other tissues display similar frequency dependence [11].

**TABLE 2.1** Properties of Human Muscle Tissue

Frequency (MHz)	Dielectric Constant	Conductivity (S/m)
10	160	.625
40	97	.693
100	72	.89
200	56.5	1.28
300	54	1.37
433	53	1.43
915	51	1.60

Write a program to calculate values of  $\varepsilon$ ,  $\sigma$ ,  $\chi_1$ , 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. In other words, have your program prompt you for the parameters  $\varepsilon$ ,  $\sigma$ ,  $\chi_1$ , and  $t_0$  and then calculate and display the effective dielectric constant and conductivity at each frequency of interest.) Then write a program that will do a similar calculation for the Lorentz parameters. Which one is better?

4. Using a Debye or Lorentz formulation, use the parameters you found from problem 2.5.3 to do a simulation of a pulse striking a medium of muscle tissue 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. Microwave Theory and Tech.*, vol. MTT-38, July 1990, pp. 919–927.
- [2] D. M. Sullivan, Mathematical methods for treatment planning in deep regional hyperthermia, *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-39, May 1991, pp. 864–872.
- [3] R. Luebbers, F. Hunsberger, K. Kunz, R. Standler, and M. Schneider, A frequency-dependent finite-difference time-domain formulation for dispersive materials, *IEEE Trans. Electromag. Compat.*, vol. EMC-32, Aug. 1990, pp. 222–227.
- [4] R. M. Joseph, S. C. Hagness, and A. Taflov, Direct time integration of Maxwell's equations in linear dispersive media with absorption for scattering and propagation of femtosecond electromagnetic pulses, *Optics Letters*, vol. 16, Sept. 1991, pp. 1412–1414.
- [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. Microwave Theory Tech.*, vol. 41, April 1993, pp. 658–665.
- [6] D. M. Sullivan, Frequency-dependent FDTD methods using Z transforms, *IEEE Trans. Antenna Prop.*, vol. AP-40, Oct. 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.*, vol. BME-34, Feb. 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. Microwave Theory and Tech.*, vol. MTT-38, Feb. 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. Microwave Power*, vol. 15, 1980, pp. 19–16.



```

/* FD1D_2.1.c. 1D FDTD simulation of a pulse hitting
   a dielectric medium */
/* New formulation using flux density */

# 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 = 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 Tranform 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[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] ) ;
  ix[k] = ix[k] + gb[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 < 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 %6.2f \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];
    }

/* 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 files "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);

    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, chil, 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<= 4; 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( "chil --> ");
scanf("%f", &chil);
tau = 1000.;          /* Make sure tau is > 0. */
if(chil > 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,chil);
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 + chil*dt/tau) ;
    gb[k] = sigma*dt/epsz ;
    gc[k] = chil*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 < 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 %6.2f  \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=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, " %8.5f \n", ampn[0][k]); }
fclose(fp);
    fp = fopen( "Amp1", "w");
    for ( k=1; k < KE; k++ )
        { fprintf( fp, " %8.5f \n", ampn[1][k]); }
fclose(fp);
    fp = fopen( "Amp2", "w");
    for ( k=1; k < KE; k++ )
        { fprintf( fp, " %8.5f \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 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:

$$\frac{\partial \tilde{\mathbf{D}}}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \nabla \times \mathbf{H} \quad (3.1a)$$

$$\tilde{\mathbf{D}}(\omega) = \epsilon_r^*(\omega) \cdot \tilde{\mathbf{E}}(\omega) \quad (3.1b)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \nabla \times \tilde{\mathbf{E}}. \quad (3.1c)$$

When we get to three dimensional-simulation, we will wind up dealing with six different fields:  $\tilde{\mathbf{E}}_x$ ,  $\tilde{\mathbf{E}}_y$ ,  $\tilde{\mathbf{E}}_z$ ,  $\mathbf{H}_x$ ,  $\mathbf{H}_y$ , and  $\mathbf{H}_z$ . In doing two-dimensional simulation, we choose between one of two groups of three vectors each: (1) the transverse magnetic (TM) mode, which is composed of  $\tilde{\mathbf{E}}_z$ ,  $\mathbf{H}_x$ , and  $\mathbf{H}_y$ , or (2) transverse electric (TE) mode, which is composed of  $\tilde{\mathbf{E}}_x$ ,  $\tilde{\mathbf{E}}_y$ , and  $\mathbf{H}_z$ . We will work with the TM mode. Equations (3.1) are now reduced to

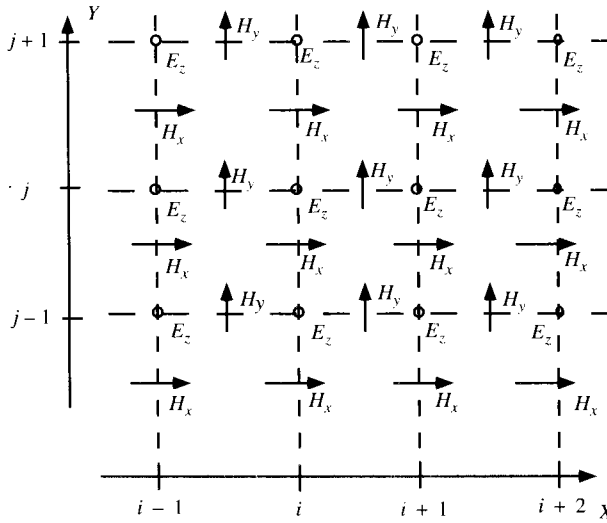
$$\frac{\partial D_z}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right) \quad (3.2a)$$

$$D_z(\omega) = \epsilon_r^*(\omega) \cdot E_z(\omega) \quad (3.2b)$$

$$\frac{\partial H_x}{\partial t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\partial E_z}{\partial y} \quad (3.2c)$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{\partial E_z}{\partial x}. \quad (3.2d)$$

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 Eqs. (3.2a), (3.2c), and



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

(3.2d) into the finite differencing scheme results in the following difference equations [1]:

$$\frac{D_z^{n+1/2}(i, j) - D_z^{n-1/2}(i, j)}{\Delta t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{H_y^n(i + 1/2, j) - H_y^n(i - 1/2, j)}{\Delta x} \right) - \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{H_x^n(i, j + 1/2) - H_x^n(i, j - 1/2)}{\Delta x} \right) \quad (3.3a)$$

$$\frac{H_x^{n+1}(i, j + 1/2) - H_x^n(i, j + 1/2)}{\Delta t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{E_z^{n+1/2}(i, j + 1) - E_z^{n+1/2}(i, j)}{\Delta x} \quad (3.3b)$$

$$\frac{H_y^{n+1}(i + 1/2, j) - H_y^n(i + 1/2, j)}{\Delta t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \frac{E_z^{n+1/2}(i + 1, j) - E_z^{n+1/2}(i, j)}{\Delta x}. \quad (3.3c)$$

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

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

we get the equations:

$$dz[i][j] = dz[i][j] + .5*(hy[i][j] - hy[i-1][j] - hx[i][j] + hx[i][j-1]); \quad (3.4a)$$

$$ez[i][j] = gaz[i][j]*(dz[i][j] - iz[i][j]); \quad (3.4b)$$

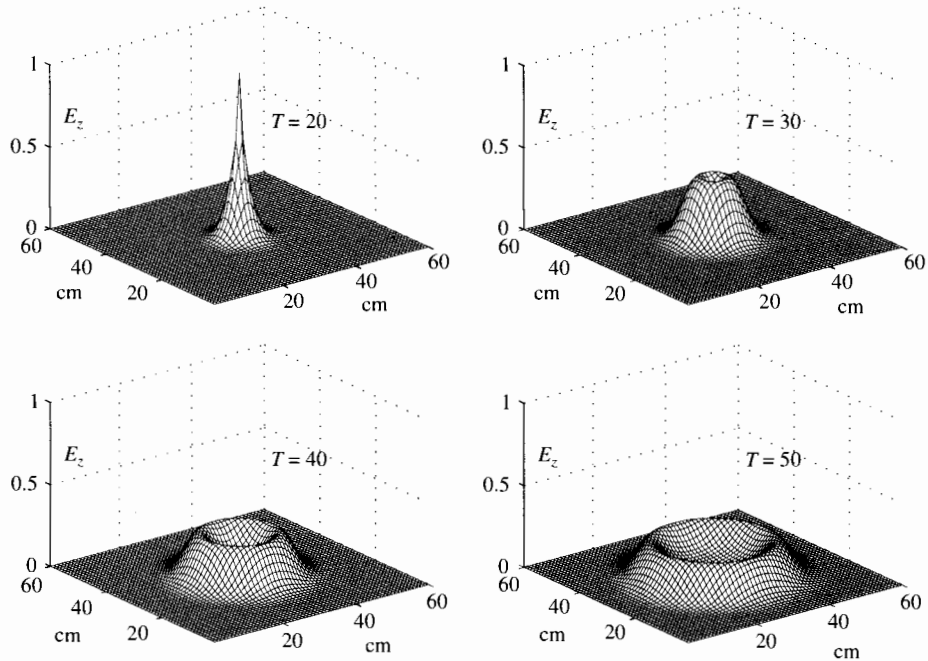
$$iz[i][j] = iz[i][j] + gbz[i][j]*ez[i][j]; \quad (3.4c)$$

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

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

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.c` implements the above equations. It has a simple Gaussian pulse source that is generated in the middle of the problem space. Figure 3.2 demonstrates a simulation for the first 50 time steps.



**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.

### 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?

## 3.2 THE PERFECTLY MATCHED LAYER (PML)

Up to now, we have only briefly mentioned the issue of absorbing boundary conditions (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,  $d_z$ ,  $e_z$ ,  $h_x$ , and  $h_y$ , as well as matrices to hold the parameters  $g_{az}$ ,  $g_{bz}$ , and  $i_z$ . 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, unpredictable reflections would be generated that would go back inward. There would be no way to determine which is the real wave and which is 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].

One of the most flexible and efficient ABCs is the perfectly matched layer (PML) developed by Berenger [4]. The basic idea is this: if a wave is propagating in medium A and it impinges upon medium B, the amount of reflection is dictated by the intrinsic impedances of the two media

$$\Gamma = \frac{\eta_A - \eta_B}{\eta_A + \eta_B}, \quad (3.5)$$

which are determined by the dielectric constants  $\varepsilon$  and permeabilities  $\mu$  of the two media

$$\eta = \sqrt{\frac{\mu}{\varepsilon}}. \quad (3.6)$$

Up to now, we have assumed that  $\mu$  was a constant, so when a propagating pulse went from  $\varepsilon = 1$  to  $\varepsilon = 4$ , 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  $\mu$  changed with  $\varepsilon$  so  $\eta$  remained a constant,  $\Gamma$  would be zero and no reflection would occur. This still doesn't 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 hits the boundary. This is accomplished by making both  $\varepsilon$  and  $\mu$  of Eq. (3.6) complex, because the imaginary part represents the part that causes decay. (See Appendix 1A at the end of Chapter 1.)

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

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

$$D_z(\omega) = \varepsilon_r^*(\omega) \cdot E_z(\omega) \quad (3.7b)$$

$$j\omega H_x = -c_0 \frac{\partial E_z}{\partial y} \quad (3.7c)$$

$$j\omega H_y = c_0 \frac{\partial E_z}{\partial x} \quad (3.7d)$$

Remember that we have eliminated  $\varepsilon$  and  $\mu$  from the spatial derivatives in Eqs. (3.7a), (3.7c), and (3.7d) for the normalized units. Instead of putting them back to implement the PML, we will add fictitious dielectric constants and permeabilities  $\varepsilon_{Fz}^*$ ,  $\mu_{Fx}^*$  and  $\mu_{Fy}^*$  [5]:

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

$$D_z(\omega) = \varepsilon_r^*(\omega) \cdot E_z(\omega) \quad (3.8b)$$

$$j\omega H_x \cdot \mu_{Fx}^*(x) \cdot \mu_{Fx}^*(y) = -c_0 \frac{\partial E_z}{\partial y} \quad (3.8c)$$

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

A few things are worth noting: first, the value  $\varepsilon_F$  is associated with the flux density  $D$ , not the electric field  $E$ ; second, we have added two values each of  $\varepsilon_F$  in Eqs. (3.8a), and  $\mu_F$  in Eq. (3.8c) and (3.8d), one for the  $x$  direction and one for the  $y$  direction; and finally, nothing was added to Eq. (3.8b). 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] shows that there are two conditions to form a PML:

1. The impedance going from the background medium to the PML must be constant,

$$\eta_0 = \eta_m = \sqrt{\frac{\mu_{Fx}^*}{\varepsilon_{Fx}^*}} = 1. \quad (3.9)$$

The impedance is 1 because of our normalized units.

2. In the direction perpendicular to the boundary (the  $x$  direction, for instance), the relative dielectric constant and relative permeability must be the inverse of those in



the other directions; i.e.,

$$\varepsilon_{Fx}^* = \frac{1}{\varepsilon_{Fy}^*} \quad (3.10a)$$

$$\mu_{Fx}^* = \frac{1}{\mu_{Fy}^*} \quad (3.10b)$$

We will assume that each of these is a complex quantity of the form

$$\varepsilon_{Fm}^* = \varepsilon_{Fm} + \frac{\sigma_{Dm}}{j\omega\varepsilon_0} \quad \text{for } m = x \text{ or } y \quad (3.11a)$$

$$\mu_{Fm}^* = \mu_{Fm} + \frac{\sigma_{Hm}}{j\omega\mu_0} \quad \text{for } m = x \text{ or } y. \quad (3.11b)$$

The following selection of parameters satisfies Eqs. (3.10a) and (3.10b) [7]:

$$\varepsilon_{Fm} = \mu_{Fm} = 1 \quad (3.12a)$$

$$\frac{\sigma_{Dm}}{\varepsilon_0} = \frac{\sigma_{Hm}}{\mu_0} = \frac{\sigma_D}{\varepsilon_0}. \quad (3.12b)$$

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

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

This fulfills the first requirement above. If  $\sigma$  increases gradually as it goes into the PML, Eqs. (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  $\varepsilon_F^*$  and  $\mu_F^*$  in Eq. (3.8)

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

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

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

and use the values of Eq. (3.12):

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

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

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

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. (Eq. (3.10a) is irrelevant for this 2D 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 Eqs. (3.13) have to be put into the FDTD formulation. First, look at the left side of Eq. (3.13.a):

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

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

$$\begin{aligned} \frac{\partial D_z}{\partial t} + \frac{\sigma_D(i)}{\varepsilon_0} D_z &\cong \frac{D_z^{n+1/2}(i, j) - D_z^{n-1/2}(i, j)}{\Delta t} + \frac{\sigma_D(i)}{\varepsilon_0} \frac{D_z^{n+1/2}(i, j) + D_z^{n-1/2}(i, j)}{2} \\ &= D_z^{n+1/2}(i, j) \frac{1}{\Delta t} \left[ 1 + \frac{\sigma_D(i) \cdot \Delta t}{2 \cdot \varepsilon_0} \right] - D_z^{n-1/2}(i, j) \frac{1}{\Delta t} \left[ 1 - \frac{\sigma_D(i) \cdot \Delta t}{2 \cdot \varepsilon_0} \right]. \end{aligned}$$

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

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

where once again we have used the fact that

$$\frac{\Delta t}{\Delta x} c_0 = \frac{\Delta x / (2 \cdot c_0)}{\Delta x} c_0 = \frac{1}{2}.$$

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

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

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

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

$$\begin{aligned} H_y^{n+1}(i + 1/2, j) &= fi3(i + 1/2) \cdot H_y^n(i + 1/2, j) \\ &+ fi2(i + 1/2) \cdot 0.5 \cdot [E_z^{n+1/2}(i + 1, j) - E_z^{n+1/2}(i, j)], \end{aligned} \quad (3.16)$$

where

$$fi2(i + 1/2) = \frac{1}{1 + \sigma_D(i + 1/2) \cdot \Delta t / (2 \cdot \varepsilon_0)} \quad (3.17a)$$

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

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 other two. Start by rewriting it as

$$j\omega H_x = -c_0 \left[ \frac{\partial E_z}{\partial y} + \frac{\sigma_D(x)}{\varepsilon_0} \frac{1}{j\omega} \frac{\partial E_z}{\partial y} \right].$$

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

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

Implementing this into an FDTD formulation gives

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

Note the extra  $\Delta t$  in front of the summation. This is part of the approximation of the time

domain integral. Finally we get

$$\begin{aligned} H_x^{n+1}(i, j + 1/2) &= H_x^n(i, j + 1/2) + \frac{c_0 \cdot \Delta t}{\Delta x} \text{curl}_e \\ &\quad + \frac{\Delta t \cdot c_0 \sigma_D(x) \cdot \Delta t}{\Delta x \epsilon_0} I_{H_x}^{n+1/2}(i, j + 1/2) \\ &= H_x^n(i, j + 1/2) + \frac{c_0 \cdot \Delta t}{\Delta x} \text{curl}_e \\ &\quad + \frac{\sigma_D(x) \cdot \Delta t}{2\epsilon_0} I_{H_x}^{n+1/2}(i, j + 1/2). \end{aligned}$$

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

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

$$I_{H_x}^{n+1/2}(i, j + 1/2) = I_{H_x}^{n-1/2}(i, j + 1/2) + \text{curl}_e \quad (3.18b)$$

$$\begin{aligned} H_x^{n+1}(i, j + 1/2) &= H_x^n(i, j + 1/2) + 0.5 \cdot \text{curl}_e \\ &\quad + fi1(i) \cdot I_{H_x}^{n+1/2}(i, j + 1/2) \end{aligned} \quad (3.18c)$$

with

$$fi1(i) = \frac{\sigma(i) \cdot \Delta t}{2\epsilon_0}. \quad (3.19)$$

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

$$xn = \frac{\sigma \cdot \Delta t}{2 \cdot \epsilon_0}$$

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

$$xn(i) = .333 * \left( \frac{i}{length\_pml} \right)^3 \quad i = 1, 2, \dots, length\_pml \quad (3.20)$$

$$fi1(i) = xn(i) \quad (3.21a)$$

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

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

Notice that the quantity in parentheses in Eq. (3.20) ranges between 0 and 1. The factor “.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— $fi2$  and  $fi3$  are different only because they are computed at the half intervals,  $i + 1/2$ . The parameters vary in the following manner:

$$fi1(i) \quad \text{from 0 to .333} \quad (3.22a)$$

$$gi2(i) \quad \text{from 1 to .75} \quad (3.22b)$$

$$gi3(i) \quad \text{from 1 to .5.} \quad (3.22c)$$

Throughout the main problem space,  $fi1$  is zero, 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. Therefore, we have to go back and add the  $y$  dependent

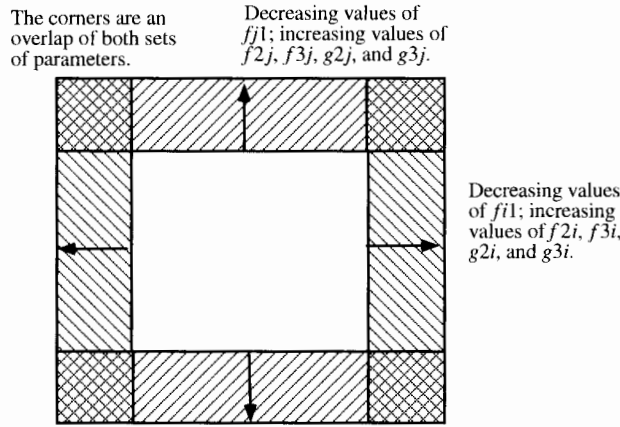


Figure 3.3 Parameters related to the perfectly matched layer (PML).

terms from Eq. (3.8) that were set aside. So instead of Eq. (3.13) we have

$$j\omega \cdot \left(1 + \frac{\sigma_D(x)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_D(y)}{j\omega\epsilon_0}\right) D_z = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right) \quad (3.23a)$$

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\epsilon_0}\right)^{-1} \left(1 + \frac{\sigma_D(y)}{j\omega\epsilon_0}\right) H_x = c_0 \cdot \left(-\frac{\partial E_z}{\partial y}\right) \quad (3.23b)$$

$$j\omega \left(1 + \frac{\sigma_D(x)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_D(y)}{j\omega\epsilon_0}\right)^{-1} H_y = c_0 \cdot \left(\frac{\partial E_z}{\partial x}\right). \quad (3.23c)$$

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

$$D_z^{n+1/2}(i, j) = gi3(i) \cdot gj3(j) \cdot D_z^{n-1/2}(i) \\ + gi2(i) \cdot gj2(j) \cdot (0.5) \cdot \begin{bmatrix} H_y^n(i+1/2, j) - H_y^n(i-1/2, j) \\ -H_x^n(i, j+1/2) + H_x^n(i, j-1/2) \end{bmatrix}.$$

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

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

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

$$H_y^{n+1}(i+1/2, j) = fi3(i+1/2) \cdot H_y^n(i+1/2, j) \\ - fi2(i+1/2) \cdot 0.5 \cdot curl\_e + fj1(j) \cdot I_{H_y}^{n+1/2}(i+1/2, j). \quad (3.24c)$$

Finally, the  $H_x$  in the  $x$  direction becomes

$$curl\_e = [E_z^{n+1/2}(i, j) - E_z^{n+1/2}(i, j+1)] \\ I_{H_x}^{n+1/2}(i, j+1/2) = I_{H_x}^{n-1/2}(i, j+1/2) + curl\_e \\ H_x^{n+1}(i, j+1/2) = fj3(j+1/2) \cdot H_x^n(i, j+1/2) \\ + fj2(j+1/2) \cdot 0.5 \cdot curl\_e \\ + fi1(i) \cdot I_{H_x}^{n+1/2}(i, j+1/2).$$

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

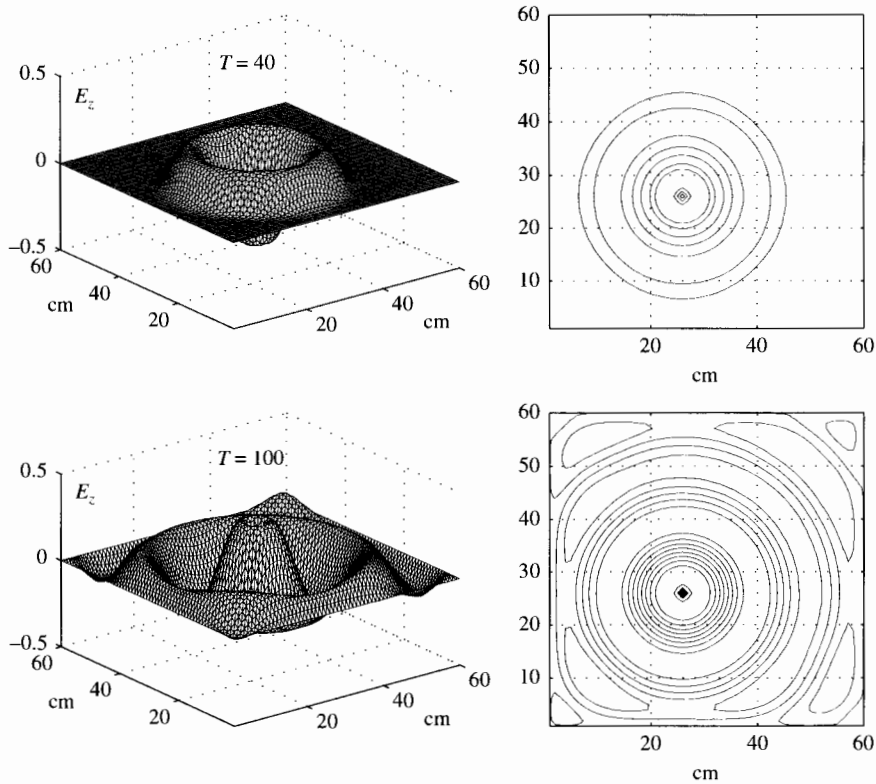
$$fi1(i) \& fj1(j) \quad \text{from 0 to .333} \quad (3.25a)$$

$$fi2(i), gi2(i), fj2(j), \& gj2(j) \quad \text{from 1 to .75} \quad (3.25b)$$

$$fi3(i), gi3(i), fj3(j), \& gj3(j) \quad \text{from 1 to .5.} \quad (3.25c)$$

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 2D parameters. Whereas memory requirements are not a main issue while we are in 2D, 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 8-point PML with the source offset five cells from center in both the  $X$  and  $Y$  directions. Note that the outgoing contours remain concentric. Only when the wave gets within eight points of the edge, which is inside the PML, does distortion start to occur.



**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 problems 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 were partially reflected.

### PROBLEM SET 3.2

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

### 3.3 TOTAL/SCATTERED FIELD FORMULATION

The simulation of plane waves is often of interest in computational electromagnetics. 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 2D FDTD program, the problem space will be divided up into two regions, the *total field* and the *scattered field* (Fig. 3.5). There are two primary reasons for doing this: (1) The propagating plane wave should not interact with the absorbing boundary conditions; (2) the load on the absorbing boundary conditions should be minimized. These boundary conditions are not perfect, i.e., 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.

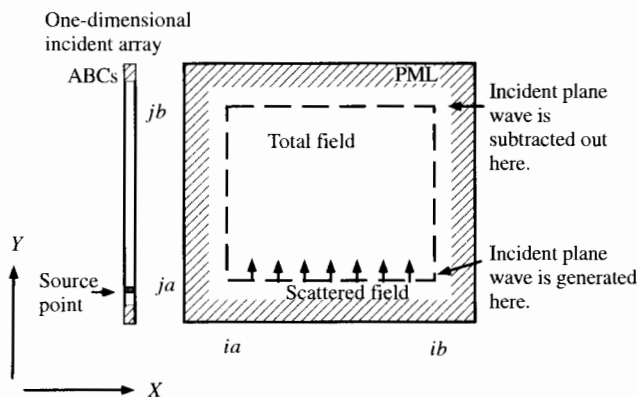


Figure 3.5 Total field/scattered field of the two-dimensional problem space.

Figure 3.5 illustrates how this is accomplished. First note that there is an auxiliary one-dimensional buffer called the *incident array*. Because this is a one-dimensional array, it is easy to generate a plane wave: a source point is chosen and the incident  $E_z$  field is just added at that point. Then a plane wave propagates away in both directions. Since 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  $j = ja$  or  $j = jb$ :

$$D_z(i, ja) = D_z(i, ja) + 0.5 \cdot H_{x\_inc}(ja - 1/2) \quad (3.26a)$$

$$D_z(i, jb) = D_z(i, jb) - 0.5 \cdot H_{x\_inc}(jb + 1/2) \quad (3.26b)$$

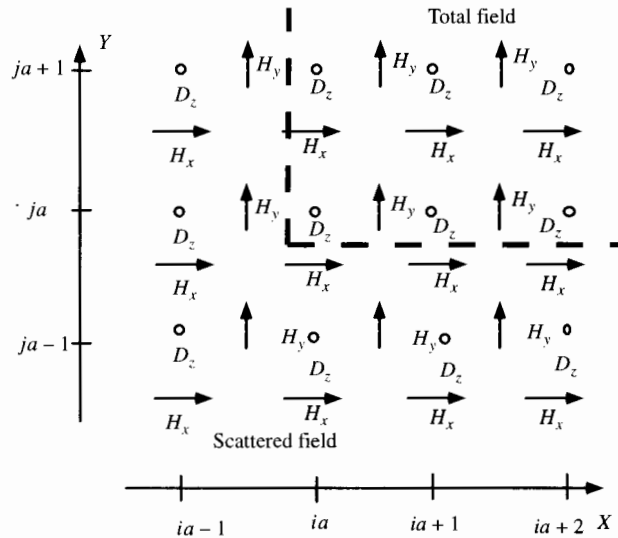


Figure 3.6 Every point is in either the total field or the scattered field.

2. The  $H_x$  field just outside  $j = ja$  and  $j = jb$ :

$$H_x(i, j_a - 1/2) = H_x(i, j_a - 1/2) + .5 \cdot E_{z\_inc}(j_a) \quad (3.27a)$$

$$H_x(i, j_b + 1/2) = H_x(i, j_b + 1/2) - .5 \cdot E_{z\_inc}(j_b) \quad (3.27b)$$

3.  $H_y$  just outside  $i = ia$  and  $i = ib$ :

$$H_y(i_a - 1/2, j) = H_x(i_a - 1/2, j) - .5 \cdot E_{z\_inc}(j) \quad (3.28a)$$

$$H_y(i_b + 1/2, j) = H_x(i_b + 1/2, j) + .5 \cdot E_{z\_inc}(j) \quad (3.28b)$$

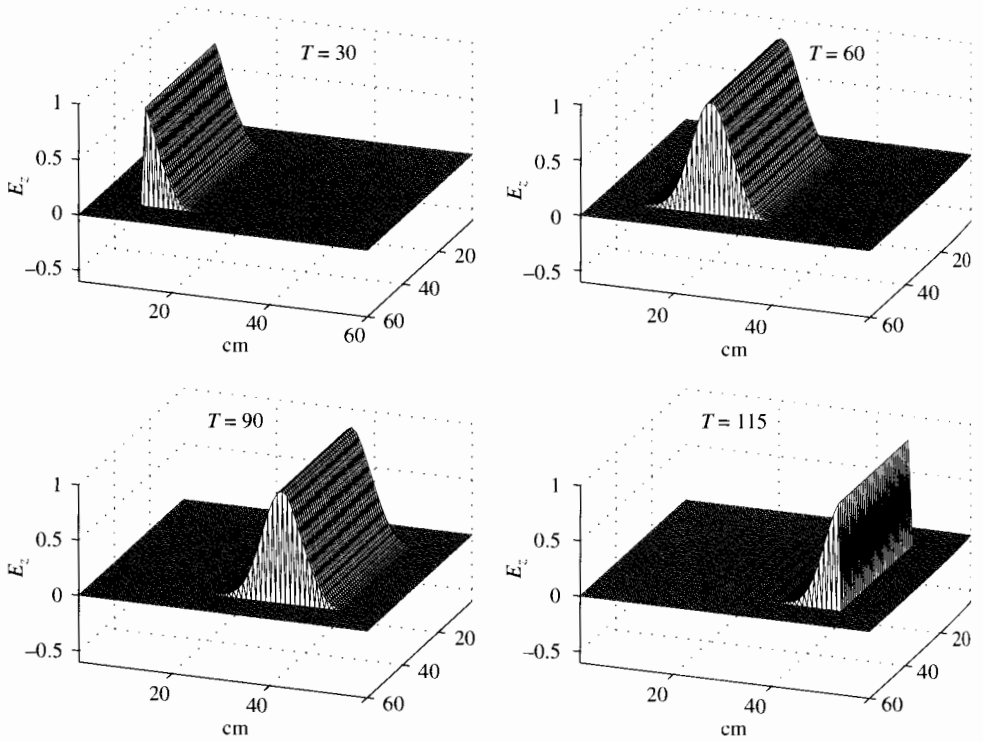
Figure 3.7 illustrates the propagation of a Gaussian pulse through the problem space. Notice how the pulse is generated at one end and completely subtracted out the other end.

### 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 have to 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:

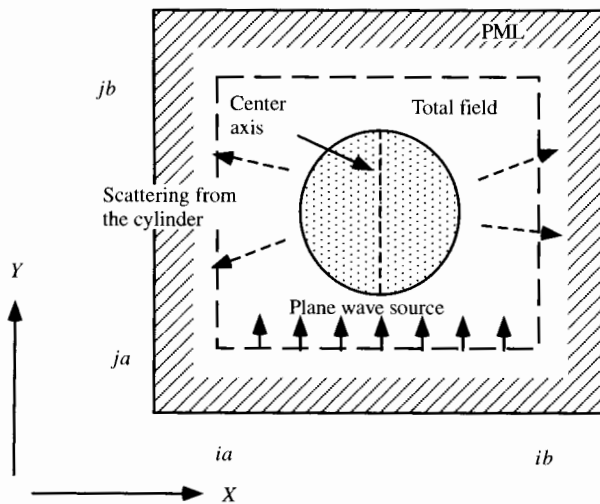
```
for ( j = ja; j < jb; j++ ) {
    for ( i=ia; i < ib; i++ ) {
        xdist = (ic-i);
        ydist = (jc-j);
        dist = sqrt(pow(xdist,2.) + pow(ydist,2.));
        if( dist <= radius) {
            ga[i][j] = 1./(epsilon + (sigma*dt/epsz));
            gb[i][j] = sigma*dt/epsz;
        } }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

Of course, this assumed that the problem space was initialized to free space. For every cell,



**Figure 3.7** Simulation of a plane wave pulse propagating in free space. The incident pulse is generated at one end and subtracted out the other end.

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.



**Figure 3.8** Diagram of the 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 weaknesses of this approach are obvious: since we determined the properties by a simple “in or out” approach, we are left with the “staircasing” at the edge of the cylinder.



Clearly 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; then determine the average dielectric properties according to how many subcells are in one medium and how many are in the other. 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; jj++ ) {
      for ( ii = -1; ii <= 1; ii++ ) {
        xdist = (ic-i) + .333*ii;
        ydist = (jc-j) + .333*jj;
        dist = sqrt(pow(xdist,2.) + pow(ydist,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]);
          } }
        } }
    ga[i][j] = 1./(eps + (sigma*dt/epsz));
    gb[i][j] = cond*dt/epsz;
  } }

```

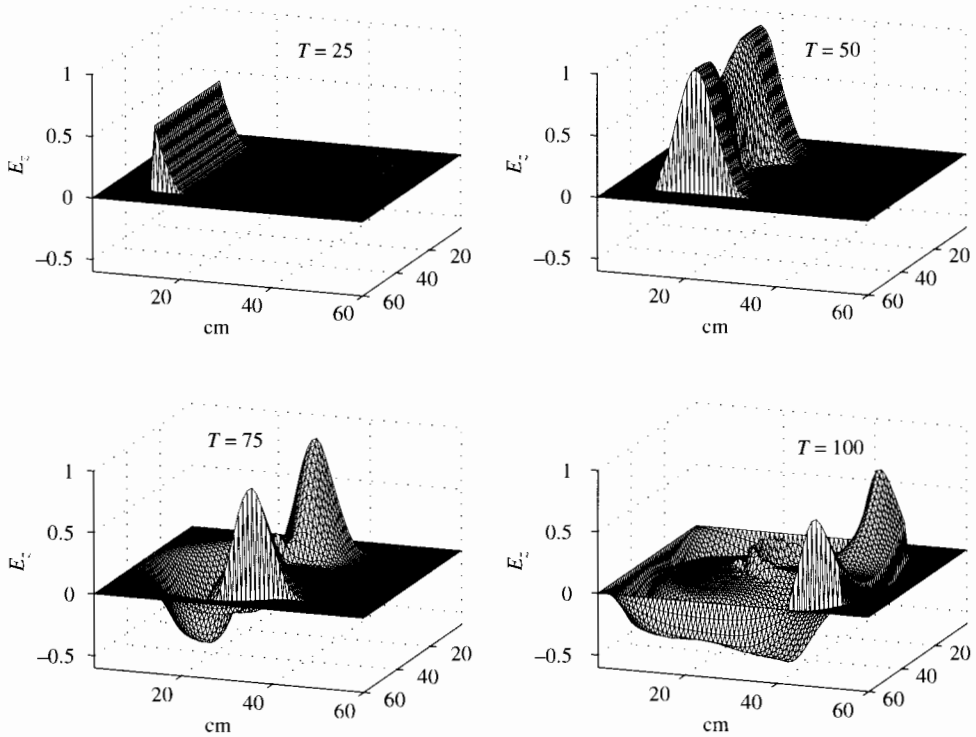
Each cell is initialized to the values  $\epsilonpsilon[0]$ ,  $\sigma[0]$ . Notice that the inner loops, which are iterated by the parameters  $ii$  and  $jj$ , each move the distance  $1/3$  of a cell length. Therefore, the cell has been divided up into nine subcells. Also, notice that as a subcell is determined to be within the radius, its contribution is added to the total epsilon ( $\epsilonpsilon[n]$ ), while subtracting out the previous epsilon ( $\epsilonpsilon[n-1]$ ). The above code is valid for an arbitrary number of layers in the cylinder, specified by `numcyl`.

The simulation of a plane wave pulse hitting a dielectric cylinder with  $\epsilon_r = 30$  and  $\sigma = .3$  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 it passes through the cylinder, and some of it goes around it. After 100 steps, the main part of the propagating pulse is being subtracted out 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 to this in section 2.4 where we were able to determine the attenuation of the energy 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 the 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

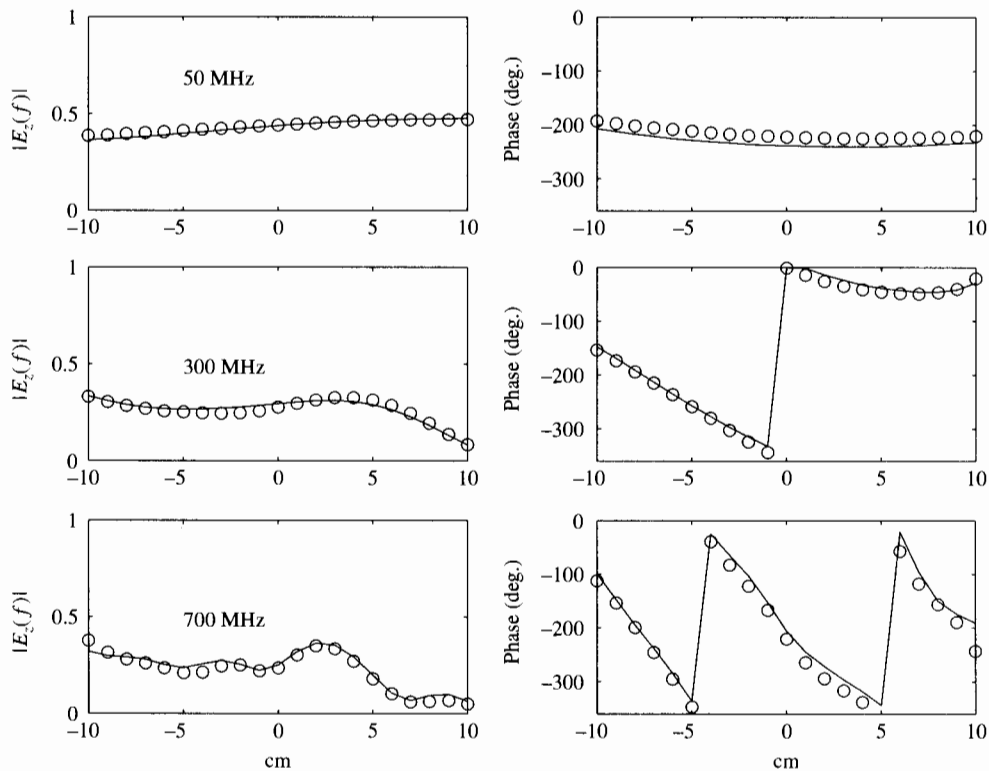


**Figure 3.9** Simulation of a plane wave pulse impinging on a dielectric cylinder. The cylinder is 20 cm in diameter and has a dielectric constant of 30 and a conductivity of 0.3 S/m.

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

### PROBLEM SET 3.3

1. The program `fd2d_3.3.c` implements the 2D 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 to 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  $j=j_a$ , propagates through the problem space, and is subtracted out at  $j=j_b$ .
3. The program `fd2d_3.4.c` differs from `fd2d_3.3.c` in that 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 the last section of this chapter.



**Figure 3.10** Comparison of the FDTD results vs. 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 dielectric constant of 30 and a conductivity of 0.3 S/m.

## REFERENCES

- [1] K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. on Antennas and Propagation*, vol. AP-17, 1996, pp. 585–589.
- [2] A. Taflov, *Computational Electrodynamics: The Finite-Difference Time-Domain*. 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 Microwave and Guided Wave Letters*, vol. 6, Feb. 1996, pp. 97–99.
- [6] Z. S. Sacks, D. M. Kingsland, R. Lee, and J. F. Lee, A perfectly matched anisotropic absorber for use as an absorbing boundary condition, *IEEE Trans. Anten. and Prop.*, vol. 43, Dec. 1995, pp. 1460–1463.
- [7] D. M. Sullivan, An unsplit step 3-D PML for use with the FDTD method, *IEEE Microwave and Guided Wave Letters*, vol. 7, July 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],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 Dz 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 Ez 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[JE];
    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/2-5;
    jc = JE/2-5;
    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 = 0.33*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 = 0.33*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 = 15.0;
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);;
dz[ic][jc] = pulse;

/* Calculate the Ez field */

```

```

for ( j=0; j < JE; j++ ) {
    for ( i=0; i < IE; i++ ) {
        ez[i][j] = ga[i][j]*dz[i][j] ;
    }
}

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

/* Set the Ez edges to 0, as part of the PML */

for ( j=0; j < JE-1; j++ ) {
    ez[0][j] = 0.0;
    ez[IE-1][j] = 0.0;
}
for ( i=0; i < IE-1; i++ ) {
    ez[i][0] = 0.0;
    ez[i][JE-1] = 0.0;
}

/* 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 */

#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_ml = 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=0; j < JE; j++ ) {
    for ( i=0; i < IE; i++ ) {
        ez[i][j] = ga[i][j]*dz[i][j] ;
    }
}

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

/* Set the Ez edges to 0, as part of the PML */

for ( j=0; j < JE-1; j++ ) {
    ez[0][j] = 0.0;
    ez[IE-1][j] = 0.0;
}
for ( i=0; i < IE-1; i++ ) {
    ez[i][0] = 0.0;
    ez[i][JE-1] = 0.0;
}

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 ---- */

```

```

/* 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 60
#define JE 60
#define NFREQS 3

main ()
{
float    real_pt[NFREQS][IE][JE], imag_pt[NFREQS][IE][JE];
float    real_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;
} } }

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

```

```

    }
        printf( " \n");
    }

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

    t0 = 25.0;
    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 ---- */

/* 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 Transform 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 = exp(-.5*(pow((t0-T)/spread,2.0) ));
ez_inc[3] = 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] + fil[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");
    { 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.));
                    printf( "%2d %9.4f \n",jc-j,amp[ic][j]);
                    fprintf( fp," %9.4f \n",amp[ic][j]);
                } } }
    close(fp);
}

}

}

```

# 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—it's just harder! It's harder because of logistical problems: you are using all vector fields and each one is in three dimensions. Nonetheless, if you pay attention and build your programs carefully, the process is straightforward.

### 4.1 FREE SPACE FORMULATION

The original FDTD paradigm was described by the *Yee cell*, (Fig. 4.1), named, of course, after Kane Yee [1]. Note that the  $E$  and  $H$  fields are assumed interleaved around a cell whose origin is at the location  $i, j, k$ . Every  $E$  field is located  $1/2$  cell width from the origin in the direction of its orientation; every  $H$  field is offset  $1/2$  cell in each direction except that of its orientation.

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

$$\frac{\partial \tilde{\mathbf{D}}}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \cdot \nabla \times \mathbf{H} \quad (4.1a)$$

$$\tilde{\mathbf{D}}(\omega) = \epsilon_r^*(\omega) \cdot \tilde{\mathbf{E}}(\omega) \quad (4.1b)$$

$$\frac{\partial \mathbf{H}}{\partial t} = -\frac{1}{\sqrt{\epsilon_0 \mu_0}} \nabla \times \tilde{\mathbf{E}}. \quad (4.1c)$$

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

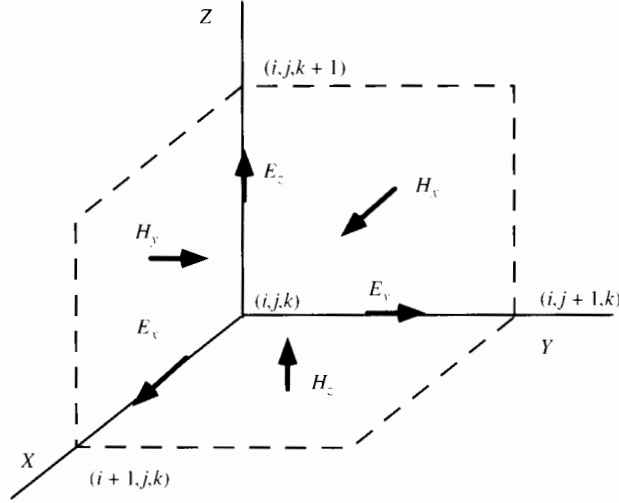


Figure 4.1 The Yee cell.

Eqs. (4.1a) and (4.1c) produce six scalar equations:

$$\frac{\partial D_x}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \right) \quad (4.2a)$$

$$\frac{\partial D_y}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \right) \quad (4.2b)$$

$$\frac{\partial D_z}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right) \quad (4.2c)$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} \right) \quad (4.2d)$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} \right) \quad (4.2e)$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\sqrt{\epsilon_0 \mu_0}} \left( \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \right). \quad (4.2f)$$

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

$$\begin{aligned} D_z^{n+1/2}(i, j, k + 1/2) &= D_z^{n-1/2}(i, j, k + 1/2) \\ &+ \frac{\Delta t}{\Delta x \cdot \sqrt{\epsilon_0 \mu_0}} (H_y^n(i + 1/2, j, k + 1/2) - H_y^n(i - 1/2, j, k + 1/2) \\ &- H_x^n(i, j + 1/2, k + 1/2) + H_x^n(i, j - 1/2, k + 1/2)) \end{aligned} \quad (4.3a)$$

$$\begin{aligned} H_z^{n+1}(i + 1/2, j + 1/2, k) &= H_z^n(i + 1/2, j + 1/2, k) \\ &- \frac{\Delta t}{\Delta x \cdot \sqrt{\epsilon_0 \mu_0}} (E_y^{n+1/2}(i + 1, j + 1/2, k) - E_y^{n+1/2}(i, j + 1/2, k) \\ &- E_x^{n+1/2}(i + 1/2, j + 1, k) + E_x^{n+1/2}(i + 1/2, j, k)). \end{aligned} \quad (4.3b)$$

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

$$\begin{aligned}
 dx[i][j][k] &= dx[i][j][k] \\
 &\quad + .5*(hz[i][j][k] - hz[i][j-1][k] \\
 &\quad \quad - hy[i][j][k] + hy[i][j][k-1]) ; \\
 dy[i][j][k] &= dy[i][j][k] \\
 &\quad + .5*(hx[i][j][k] - hx[i][j][k-1] \\
 &\quad \quad - hz[i][j][k] + hz[i-1][j][k]) ; \\
 dz[i][j][k] &= dz[i][j][k] \\
 &\quad + .5*(hy[i][j][k] - hy[i-1][j][k] \\
 &\quad \quad - hx[i][j][k] + hx[i][j-1][k]) ; \\
 \\
 hx[i][j][k] &= hx[i][j][k] \\
 &\quad + .5*(ey[i][j][k+1] - ey[i][j][k] \\
 &\quad \quad - ez[i][j+1][k] + ez[i][j][k]) ; \\
 hy[i][j][k] &= hy[i][j][k] \\
 &\quad + .5*(ez[i+1][j][k] - ez[i][j][k] \\
 &\quad \quad - ex[i][j][k+1] + ex[i][j][k]) ; \\
 hz[i][j][k] &= hz[i][j][k] \\
 &\quad + .5*(ex[i][j+1][k] - ex[i][j][k] \\
 &\quad \quad - ey[i+1][j][k] + ey[i][j][k]) ;
 \end{aligned}$$

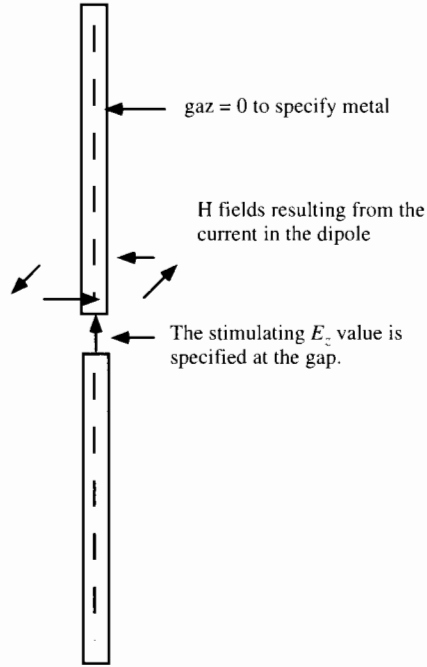
The relationship between  $E$  and  $D$ , corresponding to Eq. (4.1b), is exactly the same as the one-dimensional or two-dimensional cases, except now there will be three equations!

The program `fd3d_4.1.c` at the end of the 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. 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 way: The metal of the arms is specified by setting the  $g_{xz}$  parameters to zero in the cells corresponding to metal (See Eqs. (2.9) and (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]

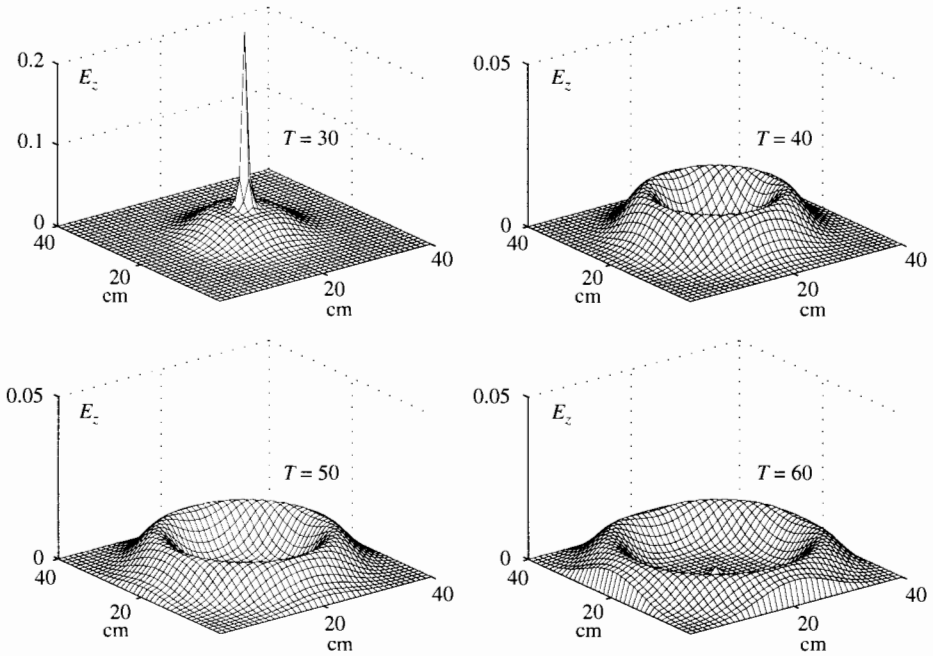
$$\oint_C \mathbf{H} \cdot d\mathbf{l} = I,$$

i.e., 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  $H$  field values around the gap, as illustrated in Fig. 4.2. I think you can see that specifying the  $E$  field in the gap is easier.

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 simulation: unless one has unusually good graphics, visualizing three dimensions can be difficult.



**Figure 4.2** A dipole antenna. The FDTD program specifies the metal arms of the dipole by setting  $g_{az} = 0$ . The source is specified by setting the  $E_z$  field to a value at the gap.



**Figure 4.3**  $E$  field radiation from a dipole antenna in a three-dimensional FDTD program.

### 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 you deal with three directions instead of two [3]. For instance, Eq. (3.23a) becomes

$$j\omega \cdot \left(1 + \frac{\sigma_x(x)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_y(y)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_z(z)}{j\omega\epsilon_0}\right)^{-1} D_z = c_0 \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right) \quad (4.4)$$

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

$$\begin{aligned} j\omega \cdot \left(1 + \frac{\sigma_x(x)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_y(y)}{j\omega\epsilon_0}\right) D_z &= c_0 \left(1 + \frac{\sigma_z(z)}{j\omega\epsilon_0}\right) \cdot \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y}\right) \\ &= c_0 \cdot \text{curl\_h} + c_0 \cdot \frac{\sigma_z(z)}{\epsilon_0} \frac{1}{j\omega} \text{curl\_h}, \end{aligned} \quad (4.5)$$

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

$$j\omega \cdot \left(1 + \frac{\sigma_x(x)}{j\omega\epsilon_0}\right) \left(1 + \frac{\sigma_y(y)}{j\omega\epsilon_0}\right) D_z = c_0 \cdot \left(\text{curl\_h} + \frac{\sigma_z(z)}{\epsilon_0} I_{D_z}\right).$$

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

$$\text{curl\_h} = \begin{bmatrix} H_y^n(i+1/2, j, k+1/2) - H_y^n(i-1/2, j, k+1/2) \\ -H_x^n(i, j+1/2, k+1/2) \\ +H_x^n(i, j-1/2, k+1/2) \end{bmatrix} \quad (4.6a)$$

$$I_{D_z}^n(i, j, k+1/2) = I_{D_z}^{n-1}(i, j, k+1/2) + \text{curl\_h} \quad (4.6b)$$

$$\begin{aligned} D_z^{n+1/2}(i, j, k+1/2) &= gi3(i) \cdot gj3(j) \cdot D_z^{n-1/2}(i, j, k+1/2) \\ &+ gi2(i) \cdot gj2(j) \cdot .5 \cdot (\text{curl\_h} + gk1(k) \cdot I_{D_z}^n(i, j, k+1/2)). \end{aligned} \quad (4.6c)$$

The one-dimensional  $g$  parameters are defined the same as Eq. (3.25).

We also mentioned in the last chapter that this is a “seamless” transition from the main problem space to the PML, but it suffers from one main drawback: the integral parameter  $I_{D_z}$  is an additional three-dimensional array that is dimensioned throughout the problem space, but used only at two edges. The three-dimensional implementation will have a total of six such arrays, which clearly is a waste of computer resources. For this reason,  $I_{D_z}$  will be broken up into two smaller three-dimensional arrays, one defined at the low values of  $k$  (`idz1`) and one defined at the high values of  $k$  (`idzh`).

```
for( i=1; i< IE, i++) {
  for( j=1; j< JE, j++) {
    for(k=0; k <ka; k++ ) {
      curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                - hx[i][j][k] + hx[i][j-1][k] );
      idz1[i][j][k] = idz1[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]*idz1[i][j][k] );
    } } }
```

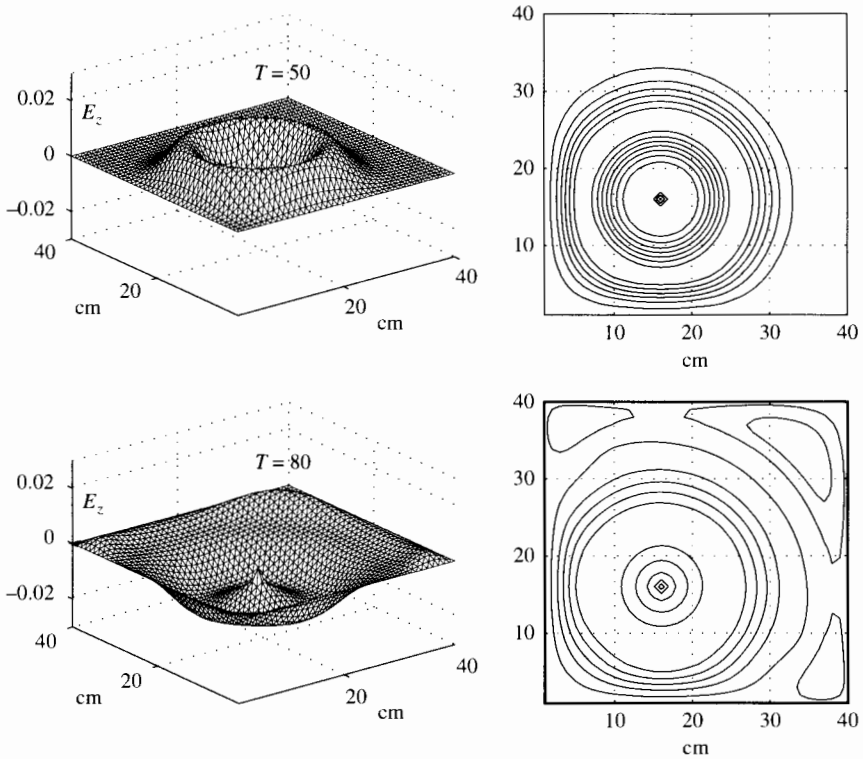
```

for( i=1; i< IE,i++) {
  for( j=1; j< JE,j++) {
    for(k=ka; k <=kb; k++ ) {
      dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
                  + gi2[i]*gj2[j].5*curl_h ;
    } } }

for( i=1; i< IE,i++) {
  for( j=1; j< JE,j++) {
    for(k=kb+1; k < KE; k++ )
      kzh = KE - k - 1;{
        curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                  - hx[i][j][k] + hx[i][j-1][k])
        idzh[i][j][kzh] = idzh[i][j][kzh]+ curl_h
        dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
                  + gi2[i]*gj2[j].5*(curl_h + gk1[k]*idzh[i][j][kzh]);
      } } }

```

Figure 4.4 shows the  $E$  field emanating from a dipole source in a program using a seven-point PML. Notice that the part of the field not in the PML radiates concentrically from the source, as it should.



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



### 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, but it also has the things described in the next section. You should be able to pick out those things having to do with the PML. Duplicate the results of Fig. 4.4.

## 4.3 TOTAL/SCATTERED FIELD FORMULATION IN THREE DIMENSIONS

Generating plane waves in three dimensions is similar to 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 out 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, Eqs. (3.26) through (3.28) are still used, but they are imposed on an entire plane instead of just 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. A similar difference exists at the  $k = kb$  boundary, where  $D_y(i, j, kb + 1)$  is just above the scattered field. Therefore, besides equations similar to Eqs. (3.26), (3.27), and (3.28) the following are necessary:

$$D_y(i, j + 1/2, ka) = D_y(i, j + 1/2, ka) - .5 \cdot H_{x\_inc}(j) \quad (4.7a)$$

$$D_y(i, j + 1/2, kb + 1) = D_y(i, j + 1/2, kb + 1) + .5 \cdot H_{x\_inc}(j). \quad (4.7b)$$

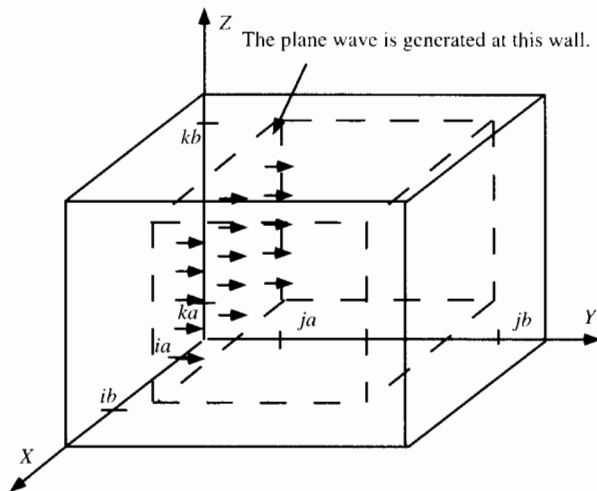
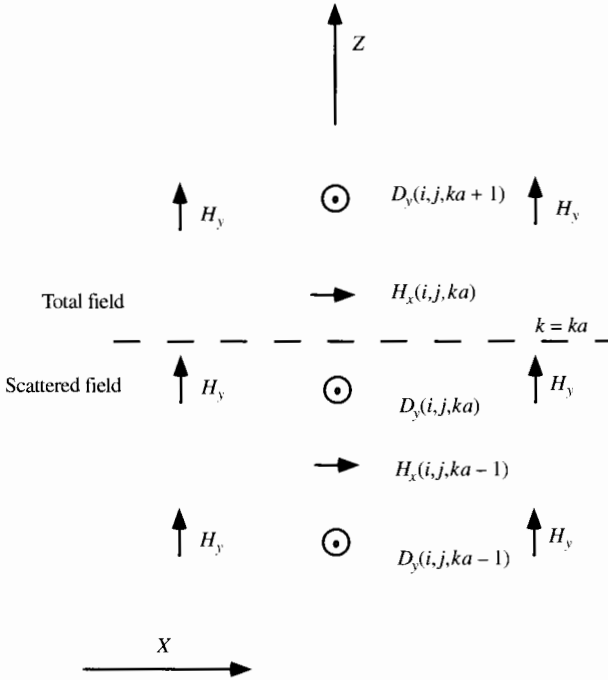


Figure 4.5 Total/scattered field in 3D.

### 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 interacts with them. In two dimensions, we chose a cylinder because we had an analytic solution with which



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

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 through an expansion of the modified Bessel functions [4].

Specifying a sphere in three dimensions is very 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:

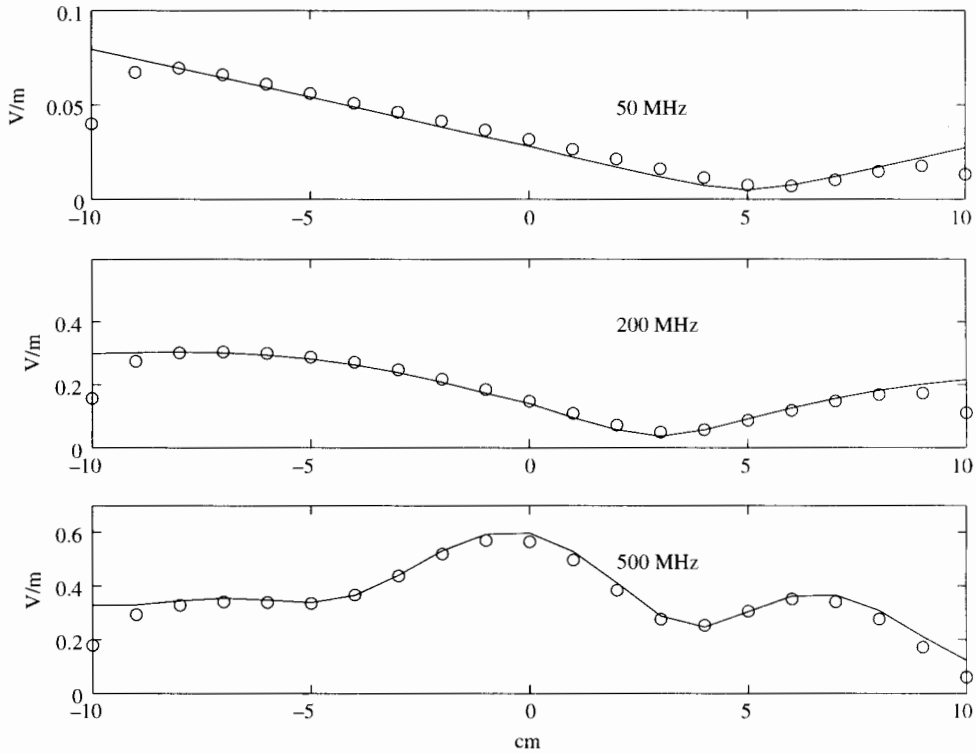
```

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(xdist,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, e.g., three loops instead of two, there is another that should be pointed out; the parameter  $zdist$  calculates the distance to the point as if it were 1/2 a cell further in the  $z$  direction. That's because it is! Look at the diagram of the Yee cell in Fig. 4.1. Each  $E$  field is assumed offset from  $i, j, k$  by a half a cell in its own direction.

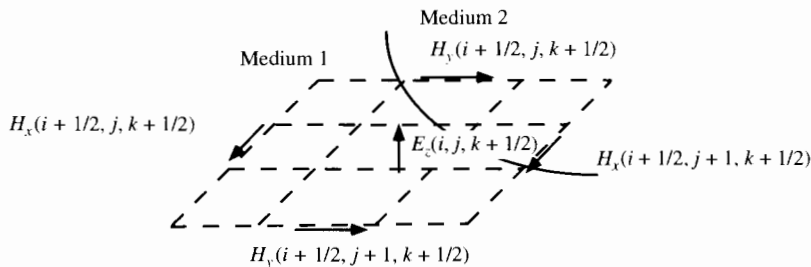
In Fig. 4.7, we show a comparison between FDTD results and Bessel function expansion results for a plane wave incident on a dielectric sphere. The sphere had a diameter of 20 cm, a dielectric constant of 30, and a conductivity of 0.3. The FDTD program is  $40 \times 40 \times 40$ , and



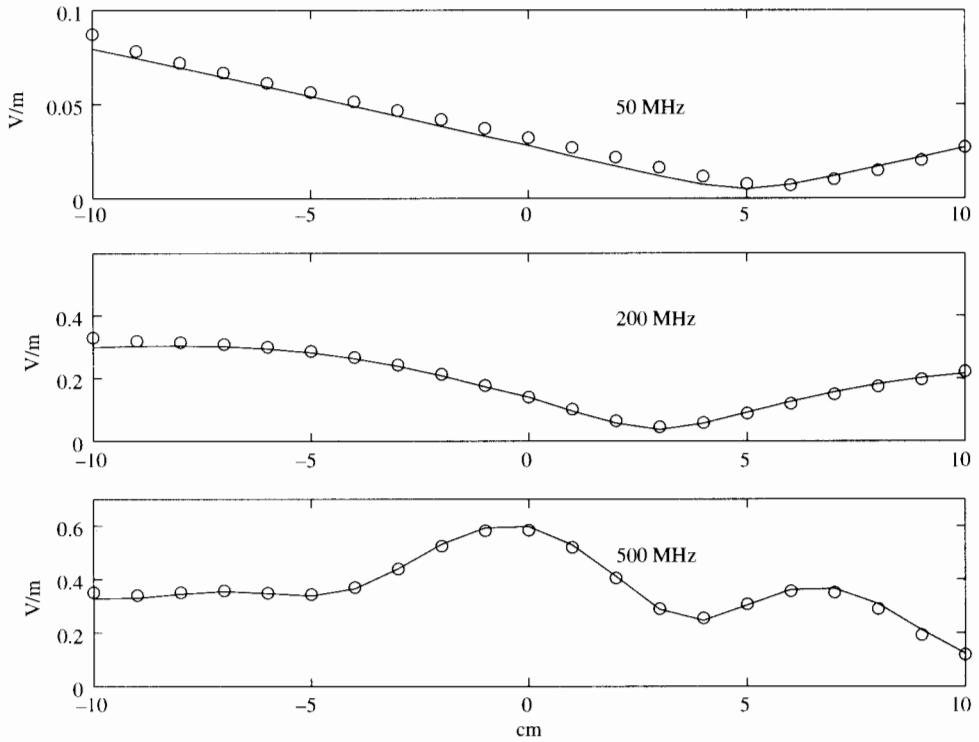
**Figure 4.7** Comparison of FDTD calculation (circles) with Bessel function expansion calculations (lines) along the main axis of a dielectric sphere, 20 cm in diameter, with  $\epsilon_r = 30$  and  $\sigma = 0.3$ . The program for the FDTD calculation used the simple “in or out” strategy to determine the parameters.

uses a seven-point PML. Apparently, the “in and out” method of determining the parameters like  $g_{az}$  and  $g_{bz}$  is not as forgiving as it was in two dimensions!

This method of determining parameters in three dimensions will give the same “staircasing” that we faced in two dimensions. We can do an averaging over subcells to improve efficiency. We might jump to the conclusion that because this meant averaging subcells in a plane in two dimensions, we will 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. Since this calculation is confined to a plane, we may think of the calculation



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



**Figure 4.9** Comparison of FDTD calculation (circles) with Bessel function expansion calculations (lines) along the main axis of a dielectric sphere, 20 cm in diameter, with  $\epsilon_r = 30$  and  $\sigma = 0.3$ . The program used for the FDTD calculation averaged over nine subcells within each cell to determine the parameters.

of  $E_z$  as being a two-dimensional problem. Therefore, in determining the parameters  $g_{az}$  and  $g_{bz}$ , we will go by how much of the 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 ( j = ja; j < jb; j++ ) {
  for ( i=ia; i < ib; i++ ) {
    eps = epsilon(0);
    cond = sigma(0);
    for ( jj = -1; jj <= 1; jj++ ) {
      for ( ii = -1; ii <= 1; ii++ ) {
        xdists = (ic-i) + .333*ii;
        ydist = (jc-j) + .333*jj;
        zdist = (jc-j-.5);
        dist =
          sqrt(pow(xdist,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] = 1./(eps + (sigma*dt/epsz));
    gbz[i][j] = cond*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 3D 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.” 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.

### REFERENCES

- [1] K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. on Antennas and Propagation*, vol. AP-17, 1996, pp. 585–589.
- [2] D. K. Cheng, *Field and Wave Electromagnetics*, Menlo Park, CA: Addison-Wesley, 1992.
- [3] D. M. Sullivan, A simplified PML for use with the FDTD method, *IEEE Microwave and Guided Wave Letters*, vol. 6, Feb. 1996, pp. 97–99.
- [4] 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 40
#define JE 40
#define KE 40

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();

    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=1; k < 30; k++) (
        gaz[ic][jc][k] = 0.;
    )
    gaz[ic][jc][kc] = 0.;

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 Dx field */
for ( k=1; k < KE; k++ ) {
    for ( j=1; j < JE; j++ ) {
        for ( i=1; i < IE; i++ ) {
            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 ( k=1; k < KE; k++ ) {
    for ( j=1; j < JE; j++ ) {
        for ( i=1; i < IE; i++ ) {
            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 ( k=1; k < KE; k++ ) {
    for ( j=1; j < JE; j++ ) {
        for ( i=1; i < IE; i++ ) {
            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] ) ;
        } } }

```

```

/* Source */

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

/* Calculate the E from D field */
for ( k=1; k < KE-1; k++ ) {
    for ( j=1; j < JE-1; j++ ) {
        for ( i=1; i < IE-1; i++ ) {
            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];
        } } }

/* Calculate the Hx field */
for ( k=1; k < KE-1; k++ ) {
    for ( j=1; j < JE-1; j++ ) {
        for ( i=1; i < IE; i++ ) {
            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 ( k=1; k < KE-1; k++ ) {
    for ( j=1; j < JE; j++ ) {
        for ( i=1; i < IE-1; i++ ) {
            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 ( k=1; k < KE; k++ ) {
    for ( j=1; j < JE-1; j++ ) {
        for ( i=1; i < IE-1; i++ ) {
            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 ---- */

printf( "Ez \n");
for ( k=1; k < KE; k++ ) {
    printf( "%2d ",k);
    for ( i=1; i < IE-1; i++ ) {
        printf( "%6.3f",ez[i][jc][k]);
    }
}

```



```
        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,"%7.4f ",ez[i][j][kc]);
        }
        fprintf( fp," \n");
    }
    fclose(fp);
}
}
```

```

/* Fd3d_4.2.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;

    float idxl[ia][JE][KE],idxh[ia][JE][KE];
    float ihxl[ia][JE][KE],ihxh[ia][JE][KE];
    float idyl[ie][ja][KE],idyh[ie][ja][KE];
    float ihyl[ie][ja][KE],ihyh[ie][ja][KE];
    float idzl[ie][je][ka],idzh[ie][je][ka];
    float ihzl[ie][je][ka],ihzh[ie][je][ka];
    int ixh, jyh, kzh;

    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; n < 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]);
}

for ( i=0; i < ia; i++ ) {
    for ( j=0; j < JE; j++ ) {
        for ( k=0; k < KE; k++ ) {
            idxl[i][j][k] = 0.0;
            idxh[i][j][k] = 0.0;
            ihxl[i][j][k] = 0.0;
            ihxh[i][j][k] = 0.0;
        } } }

for ( i=0; i < IE; i++ ) {
    for ( j=0; j < ja; j++ ) {
        for ( k=0; k < KE; k++ ) {
            idyl[i][j][k] = 0.0;
            idyh[i][j][k] = 0.0;
            ihyl[i][j][k] = 0.0;
            ihyh[i][j][k] = 0.0;
        } } }

for ( i=0; i < IE; i++ ) {
    for ( j=0; j < JE; j++ ) {
        for ( k=0; k < ka; k++ ) {
            idzl[i][j][k] = 0.0;
            idzh[i][j][k] = 0.0;
            ihzl[i][j][k] = 0.0;
            ihzh[i][j][k] = 0.0;
        } } }

/* Boundary Conditions */

for ( i=0; i < IE; i++ ) {
    gil[i] = 0.;
    fil[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 --> ");
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);
  fil[i] = xn;
  fil[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.);
  gil[i] = xn;
  gil[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( "f \n");
for ( i=0; i < IE; i++ ) {
  printf( "%2d %6.4f %6.4f %6.4f %6.4f\n",
          i,fil[i],gi2[i],gi3[i]);
  printf( "      %6.4f %6.4f %6.4f %6.4f\n",
          gil[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.+xxn);
  gj2[JE-j-1] = 1./(1.+xxn);
  gj3[j] = (1.-xxn)/(1.+xxn);
  gj3[JE-j-1] = (1.-xxn)/(1.+xxn);
  xxn = (npml-j-.5)/npml;
  xn = .33*pow(xxn,3.);
  gj1[j] = xn;
  gj1[JE-j-2] = xn;
  fj2[j] = 1./(1.+xxn);
  fj2[JE-j-2] = 1./(1.+xxn);
  fj3[j] = (1.-xxn)/(1.+xxn);
  fj3[JE-j-2] = (1.-xxn)/(1.+xxn);
}

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.+xxn);
  gk2[KE-k-1] = 1./(1.+xxn);
  gk3[k] = (1.-xxn)/(1.+xxn);
  gk3[KE-k-1] = (1.-xxn)/(1.+xxn);
  xxn = (npml-k-.5)/npml;
  xn = .33*pow(xxn,3.);
  gk1[k] = xn;
  gk1[KE-k-2] = xn;
  fk2[k] = 1./(1.+xxn);
  fk2[KE-k-2] = 1./(1.+xxn);
  fk3[k] = (1.-xxn)/(1.+xxn);
  fk3[KE-k-2] = (1.-xxn)/(1.+xxn);
}

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;
        }
    }
}

printf( " Gax \n");
for ( j=ja; j <= jb; j++ ) {
    printf( "%3d", j);
}

```

```

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

/*      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];
    xdists = (ic-i);
    ydist = (jc-j-.5);
    zdists = (kc-k);
    dist = sqrt(pow(xdists,2.) + pow(ydist,2.) + pow(zdists,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];
    xdists = (ic-i);
    ydist = (jc-j);
    zdists = (kc-k-.5);
    dist = sqrt(pow(xdists,2.) + pow(ydist,2.) + pow(zdists,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 Transform 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 < ia; 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] ) ;
      idxl[i][j][k] = idxl[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]*idxl[i][j][k]);
    } } }

for ( i=ia; i <= ib; 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] ) ;
      dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
        + gj2[j]*gk2[j]*.5*curl_h ;
    } } }

for ( i=ib+1; i < IE; i++ ) {
  ixh = i - ib - 1;
  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] ) ;
      idxh[ixh][j][k] = idxh[ixh][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]*idxh[ixh][j][k]);
    } } }

/* Calculate the Dy field */

for ( i=1; i < IE; i++ ) {
  for ( j=1; j < ja; 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] ) ;
      idyl[i][j][k] = idyl[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]*idyl[i][j][k]);
    } } }

for ( i=1; i < IE; i++ ) {
  for ( j=ja; j <= jb; 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] ) ;
        dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
                    + gi2[i]*gk2[k]*.5* curl_h ;
    } } }

    for ( i=1; i < IE; i++ ) {
        for ( j=jb+1; j < JE; j++ ) {
            jyh = j - jb - 1;
            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] ) ;
                idyh[i][jyh][k] = idyh[i][jyh][k] + curl_h;
                dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
                            + gi2[i]*gk2[k]*.5*( curl_h + gj1[j]*idyh[i][jyh][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 < ka; k++ ) {
                curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                          - hx[i][j][k] + hx[i][j-1][k] ) ;
                idz1[i][j][k] = idz1[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]*idz1[i][j][k] );
            } } }

    for ( i=1; i < IE; i++ ) {
        for ( j=1; j < JE; j++ ) {
            for ( k=ka; k <= kb; k++ ) {
                curl_h = ( hy[i][j][k] - hy[i-1][j][k]
                          - hx[i][j][k] + hx[i][j-1][k] ) ;
                dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
                            + gi2[i]*gj2[j]*.5* curl_h ;
            } } }

    for ( i=1; i < IE; i++ ) {
        for ( j=1; j < JE; j++ ) {
            for ( k=kb+1; k < KE; k++ ) {

```

```

    kzh = k - kb - 1;
    curl_h = ( hy[i][j][k] - hy[i-1][j][k]
              - hx[i][j][k] + hx[i][j-1][k] ) ;
    idzh[i][j][kzh] = idzh[i][j][kzh] + curl_h;
    dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
    + gi2[i]*gj2[j]*.5*( curl_h + gk1[k]*idzh[i][j][kzh] );
} } }

/* 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 */

/*
pulse = sin(2*pi*400*1e6*dt*T);
for ( k=kc-6; k <= kc+6; k++ ) {
    dz[ic][jc][k] = 0.;
}
pulse = exp(-.5*(pow((t0-T)/spread,2.0) ));
dz[ic][jc][kc] = pulse;

printf("%4.0f %6.2f \n ",T,pulse); */

/* 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 < ia; 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] ) ;
            ihxl[i][j][k] = ihxl[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]*ihxl[i][j][k] );
        } } }

    for ( i=ia; i <= ib; 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] ) ;
                hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
                    + fj2[j]*fk2[k]*.5*curl_e ;
            } } }

    for ( i=ib+1; i < IE; i++ ) {
        ixh = i - ib-1;
        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] ) ;
                ihxh[ixh][j][k] = ihxh[ixh][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]*ihxh[ixh][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 < ja; 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] ) ;
    ihyl[i][j][k] = ihyl[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]*ihyl[i][j][k] ) ;
} } }

for ( i=0; i < IE-1; i++ ) {
  for ( j=ja; j <= jb; 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] ) ;
      hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
        + fi2[i]*fk3[k]*.5*curl_e ;
    } } }

for ( i=0; i < IE-1; i++ ) {
  for ( j=jb+1; j < JE; j++ ) {
    jyh = j - jb-1;
    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] ) ;
      ihyh[i][jyh][k] = ihyh[i][jyh][k] + curl_e ;
      hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
        + fi2[i]*fk3[k]*.5*( curl_e + fj1[j]*ihyh[i][jyh][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 < ka; 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] ) ;
    } } }

for ( i=0; i < IE-1; i++ ) {
  for ( j=0; j < JE-1; j++ ) {
    for ( k=ka; k <= kb; k++ ) {

```

```

        curl_e = ( ex[i][j+1][k] - ex[i][j][k]
                  - ey[i+1][j][k] + ey[i][j][k] );
        hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
                    + fi2[i]*fj2[j]*.5*curl_e ;
    } } }

    for ( i=0; i < IE-1; i++ ) {
        for ( j=0; j < JE-1; j++ ) {
            for ( k=kb+1; k < KE; k++ ) {
                kzh = k - kb - 1;
                curl_e = ( ex[i][j+1][k] - ex[i][j][k]
                          - ey[i+1][j][k] + ey[i][j][k] );
                ihzh[i][j][kzh] = ihzh[i][j][kzh] + curl_e;
                hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
                            + fi2[i]*fj2[j]*.5*( curl_e + fk1[k]*ihzh[i][j][kzh] );
            } } }
    }

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

printf( "JC Plane \n" );

printf( "Ez \n" );
for ( k=0; k < KE; k++ ) {
    printf( "%2d ",k);
    for ( i=0; i < IE; i++ ) {
        printf( "%6.3f",ez[i][jc][k]);
    }
    printf( " \n" );
}

printf( "KC Plane \n" );

printf( "Ez \n" );
for ( j=0; j < JE; j++ ) {
    printf( "%2d ",j);
    for ( i=0; i < IE; i++ ) {
        printf( "%6.3f",ez[i][j][kc]);
    }
    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,"%9.6f ",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++ ) {
            fprintf( fp,"%7.4f ",ez[i][j][k]);
        }
        fprintf( fp," \n");
    }
    fclose(fp);

    printf("T = %4.0f \n",T);

/* 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");
        {   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

## Two Applications Using FDTD

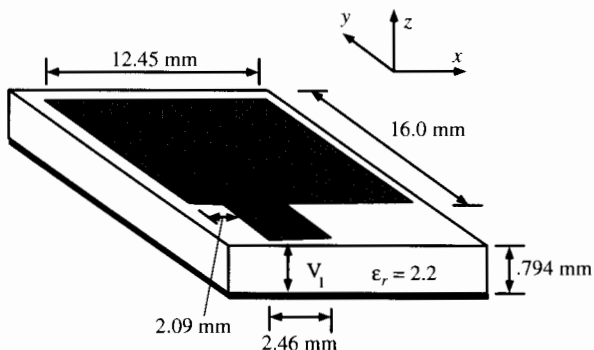
So far, the examples have been rather simple exercises to illustrate the use of FDTD—for instance, the radiation from a dipole antenna or a plane wave illuminating a dielectric sphere. In this chapter, we will look at two very specific applications: the characterization of a stripline antenna and the calculation of the far field of an aperture antenna using a time domain transformation.

### 5.1 SIMULATION OF A MICROSTRIP ANTENNA

In this section we will use an FDTD program to characterize a microstrip antenna [1–4]. In microwave engineering, devices are often characterized by the scattering parameters, usually given by  $S_{i,j}$ , where  $i$  is the input port and  $j$  is the output port [1]. Specifically, the parameter  $S_{11}$  determines the frequency domain at port one when port one is used for both the input and output. This characterization of a microstrip antenna illustrates the versatility of the FDTD method. This example differs from earlier examples in the following ways: (1) there are two different background media; (2) we will use cells which are of different sizes in the three directions; (3) we have to model metallic surfaces that are not simple wires, and (4) we are not interested so much in the resulting field distribution as we are in the input parameters of the antenna itself.

#### 5.1.1 Description of the Problem

Figure 5.1 illustrates the antenna we are going to simulate. Remember, we are more interested in the internal properties of the antenna rather than in the resulting radiating pattern, so it will not be necessary to model a large part of the surrounding area. However, since  $S_{11}$  depends on the geometry of the antenna, it is crucial that we model the metallic patch to its dimensions as closely as possible. Looking at the dimensions in Fig. 5.1, we might be inclined to say we need cells of about .05 mm to accurately get these dimensions. Then we could model the  $X$  direction as 12.45 mm, the  $Y$  direction as 16 mm, and the thickness of the substrate as .05 or .1 mm. This has two drawbacks: the accuracy of the thickness of the substrate would be



**Figure 5.1** Line-fed rectangular microstrip antenna.  $V_1$  represents the input port, the point into which the input voltage is fed and also at which the output voltage is measured.

unacceptable, and the dimensions of the rectangular patch in FDTD cells would be 249 by 320! Even though the  $Z$  dimension would be small, we would still have a huge three-dimensional problem.

This is one of those times when it is better to use cells with different sizes in the different directions. Following the example of Sheen, et al. [1], we choose  $\Delta x = 0.389$  mm,  $\Delta y = 0.4$  mm, and  $\Delta z = .265$  mm. Now we have a rectangular patch that is  $32 \Delta x$  by  $40 \Delta y$ ; the substrate is  $3 \Delta z$  thick. In choosing the time step, we take the smallest dimension  $\Delta z$  to get

$$\Delta t = \frac{\Delta z}{2 \cdot c_0} = .441 \text{ picoseconds.}$$

Now the question is: How do we change the code? We could go back to Eqs. (4.3a) and (4.3b) and instead of getting a simple 0.5 factor in front of the spatial derivatives, we would get something different for each direction. However, the following will prove simpler. Just use  $\Delta z = .265$  mm. as the basic cell size for the FDTD algorithm, and then modify any spatial derivative in the  $X$  or  $Y$  direction with the following factors:

$$ra\_x = \frac{\Delta z}{\Delta x} = \frac{.265}{.389} = 0.6812 \tag{5.1a}$$

$$ra\_y = \frac{\Delta z}{\Delta x} = \frac{.265}{.400} = 0.6625. \tag{5.1b}$$

(See the program `fd3d_patch.c` at the end of the chapter.)

### 5.1.2 Modeling the Materials

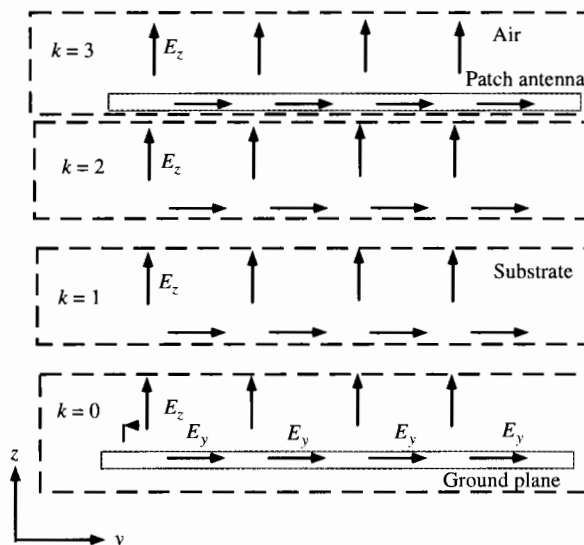
We will assume that we are only dealing with three materials: free space, the dielectric material of the substrate, and metal. We will assume that the substrate has a relative dielectric constant of 2.2 and no appreciable loss term. Therefore, the relationship between the flux density and electric field need only be the simple one

$$ex[i][j][k] = gax[i][j][k] * dx[i][j][k];$$

and we set

$$gax[i][j][k] = 1./2.2$$

at those points corresponding to the substrate. Now what about the metal? Remember we said that metal could be specified by ensuring that the  $E$  fields within those points corresponding to metal remain as zero? That is what we'll do. In fact, we can do that simply by setting  $gax$ ,  $gay$ , or  $gaz$  equal to zero at those points. In choosing which values to set as metal, we must be cognizant of the relative positions of the fields in the Yee cell (Fig. 4.1). Suppose we are viewing the FDTD lattice under the antenna, looking in the  $X$  direction. This is illustrated in



**Figure 5.2** Positions of the  $E$  fields relative to the materials being modeled. Notice that the  $E_x$  and  $E_y$  fields at  $k = 0$  are metal because they are in the ground plane, but  $E_z$  at  $k = 0$  is modeled as part of the substrate. Similarly,  $E_x$  and  $E_y$  at  $k = 3$  are metal because they are part of the antenna, but  $E_z$  at  $k = 3$  is in air.

Fig. 5.2. From Fig. 5.1, we see that the entire configuration lays on a ground plane, so we will specify all  $E_x$  and  $E_y$  values for  $k = 0$  to remain at zero. (Only the  $E_y$  values are shown in Fig. 5.2.) However, since the  $E_z$  value at  $k = 0$  lies  $1/2$  cell above the  $k = 0$  plane, it will be in the substrate. Similarly, in specifying the metal of the antenna,  $E_x$  and  $E_y$  are set to zero for the  $k = 3$  level, but  $E_z$  at  $k = 0$  is in free space.

### 5.1.3 Source

So far, we have used either a point source or a plane wave as input to our simulations. Now we will be doing something in between. The input to the antenna is that small strip leading to the larger rectangle. The source will be initialized by a uniform  $E$  field between the antenna surface and the ground plane at the point  $V_1$  in Fig. 5.1.

We can't just enter a hard source, because the main information is the reflection coming back the opposite way. A hard source would look like a metal barrier. Actually, one of the best ways is just to use the plane wave source that we've been using, but limit the active region to that little bit under the input strip. So we eliminate everything having to do with the generation of the plane wave, except at  $j = ja$ . There we have to modify the input:

$$\begin{aligned} \text{ex}[i][ja][k] &= \text{ez}[i][ja][k] \\ &\quad + .5 * \text{shape}[i][k] * \text{hx\_inc}[ja] . \\ \text{hx}[i][ja-1][k] &= \text{hx}[i][ja-1][k] \\ &\quad + .5 * \text{shape}[i][k] * \text{ez\_inc}[ja] \end{aligned}$$

The function  $\text{shape}[i][k]$  has the value of one for that portion directly under the stripline and zero everywhere else.

### 5.1.4 Boundary Conditions

We have been using the PML for our absorbing boundary condition, but that was always for homogenous media. The question is, what modifications are necessary when half the medium is the dielectric substrate, and half is free space. The answer is: none! If you didn't love the PML before, you should now. Most other boundary conditions require substantial modifications when there is a transition from one medium to another. The PML does not.

There is only one thing we have to do differently. As Fig. 5.1 shows, the whole structure that we are modeling rests on a conducting medium. We will also use this as the  $k = 0$  border of our problem space. Therefore, no PML is needed on this face.

There is one other change having to do with the one-dimensional incident simulation. Because the pulse is generated in the substrate medium, the incident simulation should also be in the substrate medium, which is accomplished with the following minor change:

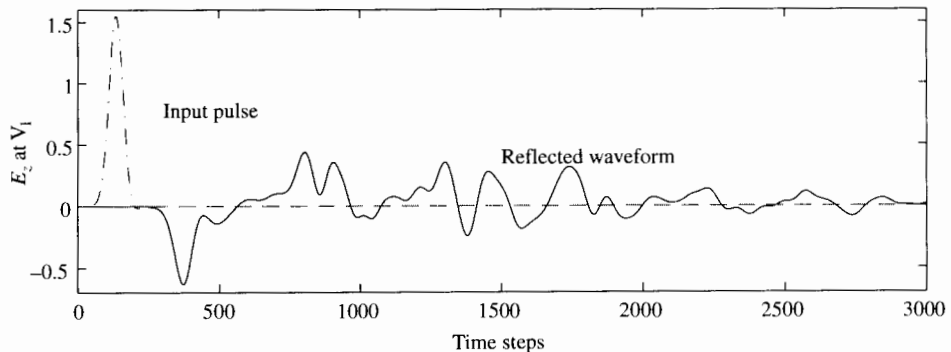
$$\begin{aligned} \text{ez\_inc}[j] &= \text{gj3}[j] * \text{ez\_inc}[j] \\ &+ \text{gj2}[j] * (.5/\text{eps\_sub}) * (\text{hx\_inc}[j-1] - \text{hx\_inc}[j]) \\ \text{hx\_inc}[j] &= \text{fj3}[j] * \text{hx\_inc}[j] \\ &+ \text{fj2}[j] * .5 * (\text{ez\_inc}[j] - \text{ez\_inc}[j+1]) \end{aligned}$$

Notice that in calculating  $\text{ez\_inc}$ , it was necessary to include the term  $\text{eps\_sub}$ . Furthermore, we could no longer use the simple free space boundary condition for the incident buffer. Instead we have implemented a PML using the same PML parameters as the main problem space.

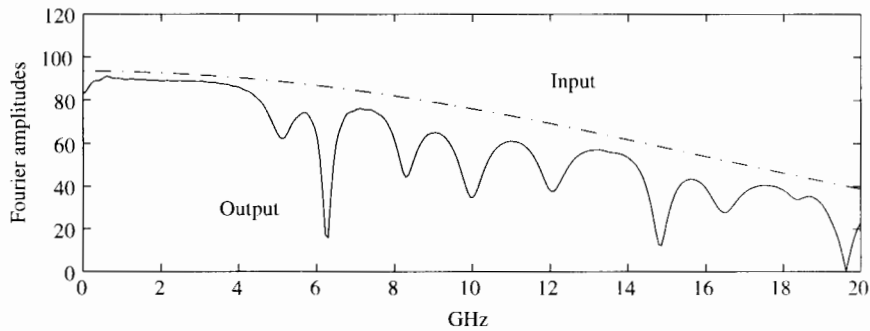
### 5.1.5 Calculating the $S_{11}$

The output of this new program is different from previous programs in the following ways: (1) the information of interest is at a single point ( $V_1$  in Fig. 5.1) as opposed to calculating the resulting fields throughout an entire area; (2) the frequency response is calculated over an entire range of frequencies, as opposed to getting the response at a few discrete frequencies; and (3) we will store the time domain data during the simulation, and then calculate the frequency response after the simulation is over. This third difference is possible because we need only the time domain data at one point  $V_1$ . As we mentioned earlier, it is usually the voltage that is of interest in calculating  $S_{11}$ . Therefore, to calculate  $V_1$ , we would need  $E_z$  at the three points in the  $k$  direction between the ground plate and the lead of the conductor. In practice, any one of the  $E_z$  values is adequate because magnitude divides out.

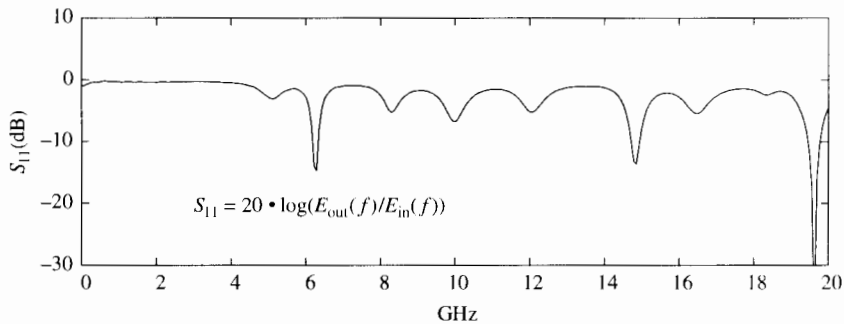
Figure 5.3 shows the time domain data from a simulation. The first 350 points are taken as the input. This is the dashed line in Fig. 5.3. The rest of the data is the reflection coming out of the conductor, which appears as the solid line. We take this as the output. Figure 5.4 shows the absolute values of the Fourier transforms of the input and output. To get the transfer function of the reflected waveform, we divide the output by the input. In actuality, engineers



**Figure 5.3** Time domain  $E_z$  field at the input point  $V_p$ . The dashed line is the input pulse and the solid line is the reflected waveform.



**Figure 5.4** Amplitudes of the Fourier transforms of the input pulse (dashed line) and the reflected waveform (solid line).



**Figure 5.5** The input impedance ( $S_{11}$  parameter) in decibels.

usually prefer this to appear in the form of decibels, which is computed by

$$S_{11}(f)_{dB} = 20 \cdot \log_{10} \frac{E_{out}(f)}{E_{in}(f)}. \quad (5.2)$$

This is plotted in Fig. 5.5, which compares favorably with the experimental results that Sheen et al. [1] used to verify their simulation results.

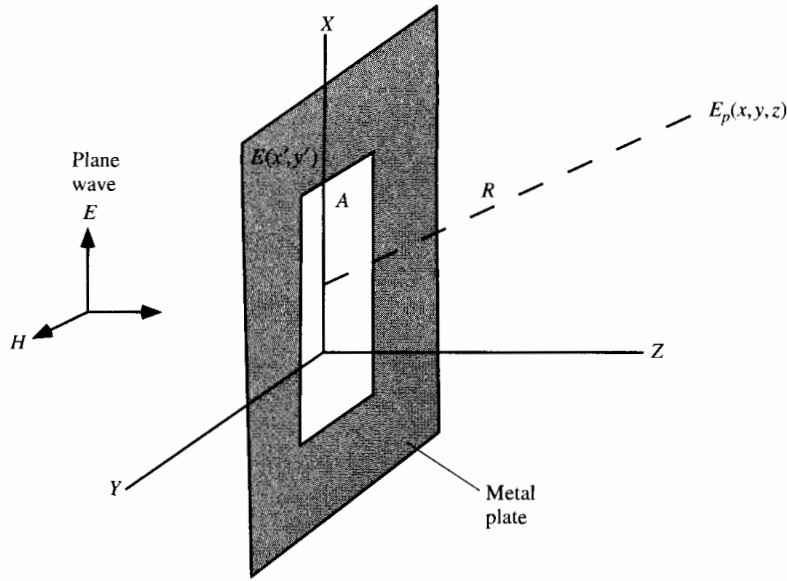
The simulation of circuits using FDTD has become an active research field [2–4].

### PROBLEM SET 5.1

1. The program `fd3d_patch.c` at the end of this section was used to do the simulations. Get this program running and verify the results. This program is provided in its entirety. However, don't type it in from scratch. Take the program `fd3d_4.3.c` and make the appropriate changes the way it is described in the text.

## 5.2 CALCULATION OF THE FAR FIELD OF AN APERTURE ANTENNA

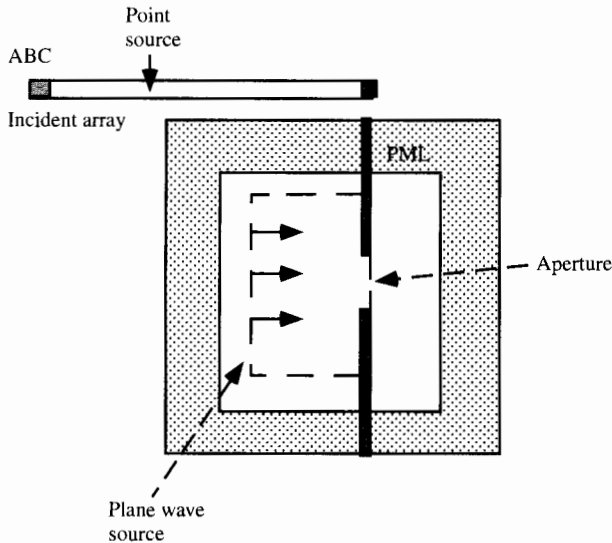
In this example, we will look at the calculation of the far field pattern from a slot antenna like the one shown in Fig. 5.6. There is a source to the left of the aperture, which we can assume is a plane wave. We are interested in the radiation pattern that comes out from the slot. The calculation of the impinging wave on the slot is not a problem. We know how to simulate a plane wave, and we know how to simulate metal by specifying some cells as having very high conductivities, or in some other way ensuring that the  $E$  fields at that point are zero. However,



**Figure 5.6** The electric field in the aperture  $E(x', y')$  can be used to calculate the far field  $E_p(x, y, z)$ . The aperture is an opening in a metal plate that lies in the  $xy$  plane.

we also want to know the radiation pattern many wavelengths away from the slot. This would require modeling a prohibitively large problem space for simulation with FDTD. In cases like this, one usually uses the equivalence principle [5] to calculate the far fields after FDTD has calculated the near fields [6–10]. The necessary computation can often be considerable.

In this section we present a method that calculates the far field at a few specific points as the FDTD simulation is running. We can do this because we know that the source of the far field is confined to a limited number of points in the slot. We will also use the equivalence principle, which says we can treat the slot of the antenna as a group of radiating currents, similar to an antenna. Therefore, we will only have to model the area immediately in front of and in back of the slot. This is illustrated in Fig. 5.7. The first thing we have to do is develop the formulation for the calculation of the external fields based on the field within the slot.



**Figure 5.7** Diagram of the FDTD program used to calculate the data needed to determine the external points.

This section illustrates several important characteristics of time domain simulation. Perhaps the most important is the following: while computer simulation makes possible the calculation of many problems which can't be handled analytically, it does not replace electromagnetic theory. Specifically, we will use the well-known Green's function approach to calculate the far field. This section will further demonstrate how a complicated frequency domain expression can be converted into the sampled time domain for implementation into FDTD.

### 5.2.1 Formulating the Transformation from the Aperture

We begin with the vector potential [11, 12]

$$\mathbf{A}(\mathbf{r}) = \frac{1}{4\pi} \iint_S \mathbf{J}_s(\mathbf{r}') \frac{e^{-jkR}}{R} dS', \quad (5.3)$$

where  $\mathbf{R} = |\bar{\mathbf{r}} - \bar{\mathbf{r}}'|$ ,

$\mathbf{J}_s$  = the current source,

$\frac{e^{-jkR}}{R}$  = the Green's function,

and the magnitude of  $R$  is

$$R = [(x - x')^2 + (y - y')^2 + z^2]^{1/2}.$$

The primed parameters indicate the source while the unprimed parameters are the scattered field (See Fig. 5.6). The  $H$  field can be obtained by

$$\mathbf{H} = \nabla_r \times \mathbf{A}. \quad (5.4)$$

Putting these together gives

$$\begin{aligned} \mathbf{H} &= \frac{1}{4\pi} \nabla_r \times \int_S \mathbf{J}_s(\mathbf{r}') \frac{e^{-jkR}}{R} dS' \\ &= \frac{1}{4\pi} \int_S \nabla_r \times \left[ \mathbf{J}_s(\mathbf{r}') \frac{e^{-jkR}}{R} \right] dS'. \end{aligned} \quad (5.5)$$

The integrand can be simplified by the following:

$$\begin{aligned} \nabla_r \times \left[ \mathbf{J}(\mathbf{r}') \frac{e^{-jkR}}{R} \right] &= \frac{e^{-jkR}}{R} \nabla_r \times \mathbf{J}(\mathbf{r}') - \mathbf{J}(\mathbf{r}') \times \nabla_r \frac{e^{-jkR}}{R} \\ &= -\mathbf{J}(\mathbf{r}') \times \nabla_r \frac{e^{-jkR}}{R}. \end{aligned}$$

The  $\nabla_r \times \mathbf{J}(\mathbf{r}')$  term can be dropped because  $\mathbf{J}(\mathbf{r}')$  is only a function of  $\mathbf{r}'$ . Also,

$$\nabla_{r'} \frac{e^{-jkR}}{R} = -\nabla_r \frac{e^{-jkR}}{R},$$

and

$$\nabla' \frac{e^{-jkR}}{R} = \frac{e^{-jkR}}{R} \left( jk + \frac{1}{R} \right) \hat{\mathbf{r}},$$

where

$$\hat{\mathbf{r}} = r_x \hat{\mathbf{x}} + r_y \hat{\mathbf{y}} + r_z \hat{\mathbf{z}} = \frac{\mathbf{R}}{R}. \quad (5.6)$$

We now have

$$\mathbf{H} = \frac{1}{4\pi} \iint_S \left( \mathbf{J}(\mathbf{r}') \times \hat{\mathbf{R}} \right) \left( jk + \frac{1}{R} \right) \frac{e^{jkR}}{R} dS'.$$

Using the principle of duality [5] we can deduce the following equation:

$$\mathbf{E} = \frac{-1}{4\pi} \iint_S (\mathbf{M}(r') \times \hat{r}) \left( jk + \frac{1}{R} \right) \frac{e^{-jkR}}{R} dS'. \quad (5.7)$$

Now we have an expression for the external  $\mathbf{E}$  field in terms of the magnetic current  $\mathbf{M}$  in the aperture. The magnetic current is a fictitious current that is calculated from the  $\mathbf{E}$  fields in the aperture [5]. We know that the fields in the aperture result from an  $x$ -polarized plane wave, and the direction normal to the aperture is the  $z$  direction, so we can write

$$\begin{aligned} \mathbf{M} &= 2\mathbf{E}_a \times \hat{n} \\ &= 2E_a \hat{x} \times \hat{z} = -2E_a \hat{y}. \end{aligned} \quad (5.8)$$

Hence,

$$\mathbf{E} = \frac{1}{4\pi} \iint_{\text{aper.}} (2E_a(r') \hat{y} \times \hat{r}) \left( jk + \frac{1}{R} \right) \frac{e^{-jkR}}{R} dS'.$$

From Eq. (5.6)

$$\hat{y} \times \hat{r} = (r_x \hat{z} - r_z \hat{x}),$$

so we can write the  $\mathbf{E}$  field in terms of the two polarizations:

$$E_x = \frac{1}{2\pi} \iint_{\text{aper.}} r_z \cdot E_a \left( jk + \frac{1}{R} \right) \frac{e^{-jkR}}{R} dS' \quad (5.9a)$$

$$E_z = -\frac{1}{2\pi} \iint_{\text{aper.}} r_x \cdot E_a \left( jk + \frac{1}{R} \right) \frac{e^{-jkR}}{R} dS'. \quad (5.9b)$$

$r_x$  and  $r_z$  represent the portions of the unit vector  $\hat{r}$  in the  $x$  and  $z$  directions, respectively.

From here on, we will restrict our development to the  $E_x$  term, which is the dominant term as long as the incoming plane wave in Fig. 5.6 is  $x$  polarized. Just be aware that we could always come back and duplicate the following results for  $E_z$ . We begin by separating  $E_x$  into two terms

$$E_x = \frac{1}{2\pi} \iint_{\text{aper.}} r_z \cdot E_a \cdot jk \frac{e^{-jkR}}{R} dS' + \frac{1}{2\pi} \iint_{\text{aper.}} r_z \cdot E_a \frac{e^{-jkR}}{R^2} dS'. \quad (5.10)$$

Starting with the first term, we will rewrite it as an explicit frequency domain function of  $x$ ,  $y$ , and  $z$ :

$$\hat{E}_{x1}(x, y, z, \omega) = \frac{1}{2\pi} \iint_{\text{aper.}} r_z \cdot \frac{\hat{E}_a(x', y', \omega)}{R} jk e^{-jkR} dx' dy'.$$

The following changes will be helpful:

$$jk = \frac{j\omega}{c_0} \quad e^{-jkR} = e^{-j\omega R/c_0}.$$

So now we have

$$\hat{E}_{x1}(x, y, z, \omega) = \frac{1}{2\pi c_0} \iint_{\text{aper.}} r_z \cdot j\omega \frac{\hat{E}_a(x', y', \omega) e^{-j\omega R/c_0}}{R} dx' dy'. \quad (5.11)$$

The first step to implementing an equation like (5.11) into the FDTD formulation is to take it to the time domain:

$$E_{x1}(x, y, z, t) = \frac{1}{2\pi c_0} \iint_{\text{aper.}} r_z \cdot \frac{d}{dt} \frac{E_z(x', y', t - R/c_0)}{R} dx' dy'. \quad (5.12)$$



The exponential term in the frequency domain resulted in a retardation in the time domain; the  $j\omega$  term became a time domain differentiation. In FDTD, the time is quantized as

$$t \cong n \cdot \Delta t$$

and the distance  $R$  is quantized as

$$R \cong \Delta x \cdot R_{\Delta x}$$

where  $\Delta x$  is the cell size used in the FDTD simulation. (We use cubic cells, i.e.,  $\Delta x = \Delta y = \Delta z$ .) We will use the usual relationship between the time step and the cell size:

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

Therefore, the retardation in  $E_a$  in Eq. (5.12) is approximated by

$$\frac{R}{c_0} = \frac{R_{\Delta x} \cdot \Delta x}{c_0} = \frac{R_{\Delta x} \cdot (\Delta t \cdot 2c_0)}{c_0} = 2 \cdot R_{\Delta x} \cdot \Delta t. \quad (5.13)$$

We will further use the spatial quantization

$$x \cong i \cdot \Delta x, y \cong j \cdot \Delta x, \text{ and } z \cong k \cdot \Delta x$$

to give

$$E_a(x', y', t - R/c_0) \cong E_a(i', j', n - 2 \cdot R_{\Delta x}). \quad (5.14)$$

The first-order approximation for the time derivative in Eq. (5.12) is

$$\frac{d}{dt} E_a(x', y', t - R/c_0) \cong \frac{E_a(i', j', n - 2 \cdot R_{\Delta x}) - E_a(i', j', n - 1 - 2 \cdot R_{\Delta x})}{\Delta t}.$$

For convenience, we will define the parameter

$$\text{del}_-E_a^n(i, j, n - 2 \cdot R_{\Delta x}) \cong E_a(i', j', n - 2 \cdot R_{\Delta x}) - E_a(i', j', n - 1 - 2 \cdot R_{\Delta x}). \quad (5.15)$$

The finite difference approximation of Eq. (5.12) is

$$\begin{aligned} E_{x1}^n(i, j, k) &\cong \frac{1}{2\pi c_0} \sum_{i'} \sum_{j'} r_z \frac{\text{del}_-E_a^n(i', j', n - 2 \cdot R_{\Delta x})}{R_{\Delta x} \cdot \Delta x \cdot \Delta t} \Delta x^2 \\ &= \frac{1}{2\pi c_0} \sum_{i'} \sum_{j'} r_z \frac{\text{del}_-E_a^n(i', j', n - 2 \cdot R_{\Delta x})}{R_{\Delta x} \cdot \Delta x \cdot (\Delta x/2 \cdot c_0)} \Delta x^2. \end{aligned} \quad (5.16)$$

By dividing out extra terms, we obtain the expression

$$E_{x1}^n(i, j, k) \cong \frac{1}{\pi} \sum_{i'} \sum_{j'} r_z \frac{\text{del}_-E_a^n(i', j', n - 2 \cdot R_{\Delta x})}{R_{\Delta x}}. \quad (5.17)$$

We will now go back and transform the second term of Eq. (5.10) into the time domain

$$E_{x2}(x, y, z, t) = \frac{1}{2\pi} \iint_{\text{aper.}} r_z \frac{E_a(x, y, t - R/c_0)}{R^2} dx' dy'.$$

Following the same quantization procedures, we obtain the following expression:

$$E_{x2}^n(i, j, k) \cong \frac{1}{2\pi} \sum_{i'} \sum_{j'} r_z \frac{E_a^n(i', j', n - 2 \cdot R_{\Delta x})}{R_{\Delta x}^2}. \quad (5.18)$$

Finally, we write the two components of  $E_x$  together:

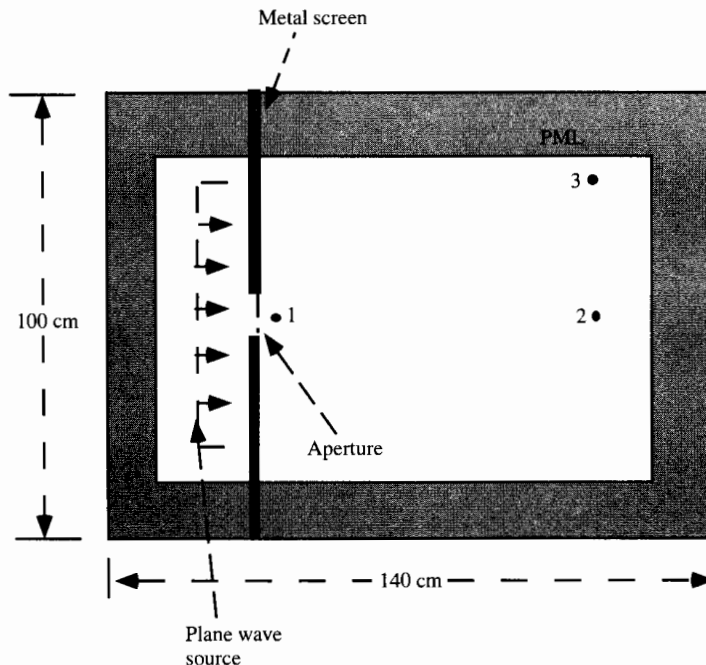
$$E_x^n(i, j, k) \cong \frac{1}{\pi} \sum_{i'} \sum_{j'} \frac{r_z}{R_{\Delta x}} \left[ \text{del}_-E_a^n(i', j', n - 2 \cdot R_{\Delta x}) + \frac{E_a^n(i', j', n - 2 \cdot R_{\Delta x})}{2 \cdot R_{\Delta x}} \right]. \quad (5.19)$$

Obviously, we would obtain identical expressions for the calculation of  $E_z$  in Eq. (5.9b), except with an  $r_x$  term instead of an  $r_z$  term.

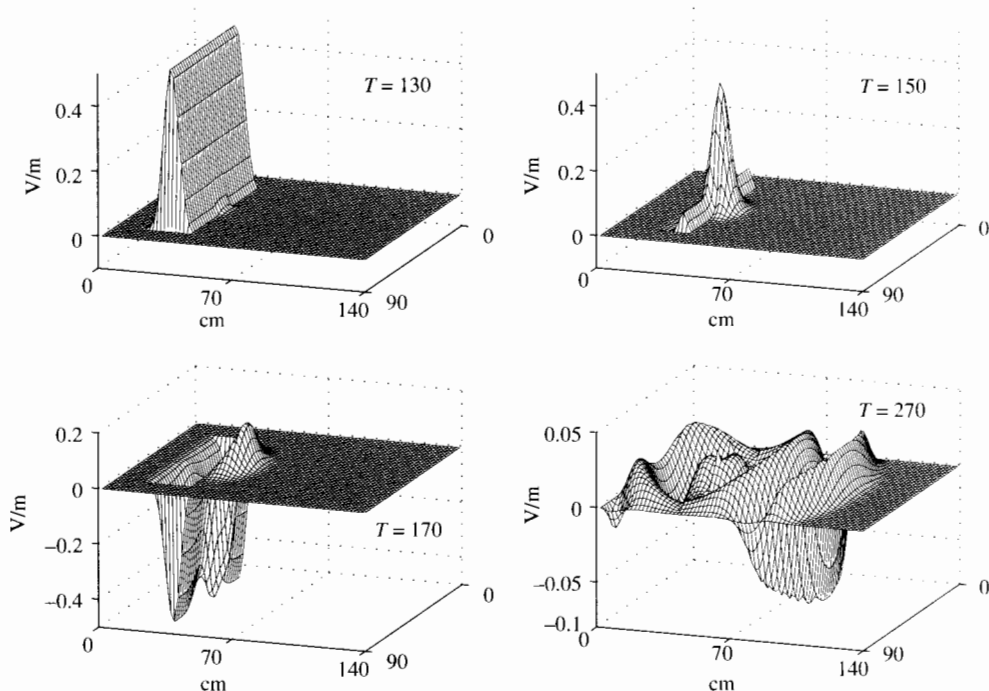
There is a point worth emphasizing: Eq. (5.19) was developed using no assumptions other than Eq. (5.8), which assumes that the magnetic current is exclusively  $y$  polarized because the incoming plane wave was  $z$  polarized. These equations are not far field approximations, nor are they restricted to the area perpendicular to the aperture.

### 5.2.2 Verification of the Accuracy of the Transformation

The accuracy of this method is verified by using an FDTD program that calculates the  $E$  field at three external points via Eq. (5.19) and compares these to the FDTD computed values. This is illustrated in Fig. 5.8. The FDTD program is  $140 \times 100 \times 100$  cells and occupies twelve megawords of core memory on a Cray T90 supercomputer. It is bordered by a 15 cell PML. Each cell is a 1 cm cube. The slot antenna is a rectangular aperture that is 10 cm in the  $X$  direction and 5 cm in the  $Y$  direction in Fig. 5.6. The plane wave pulse is a Gaussian pulse 0.25 nanoseconds wide. A simulation is illustrated in Fig. 5.9 showing the incoming plane wave interacting with the aperture. The comparison results are shown in Fig. 5.10. The first plot, labeled *Source*, is the Gaussian pulse of the incoming plane wave. The second plot, *Egrid*, is the  $E$  field in the center of the aperture. The third plot, labeled *E-deriv*, is the derivative function of Eq. (5.15). The other plots correspond to  $E$  field data at the points in Fig. 5.8, with the solid lines representing the FDTD simulation and the dashed line the computation of Eq. (5.19). Point 1 is 2 cm directly perpendicular from the center of the aperture; point 2 is 80 cm out. Point 3 is also 80 cm from the aperture, but 30 cm to the side. Clearly, the agreement is quite good. These points were chosen as representative. We could have done the calculation



**Figure 5.8** Diagram of the problem space for the three-dimensional FDTD program. By using the time domain values in the aperture, the time domain fields at points 1, 2, and 3 are calculated via Eq. (5.5). The accuracy is verified by comparison with the time domain data calculated directly by the FDTD program. Point 1 is 2 cm in front of the aperture, point 2 is 80 cm in front of the aperture, and point 3 is 80 cm from the aperture and 30 cm to the side.



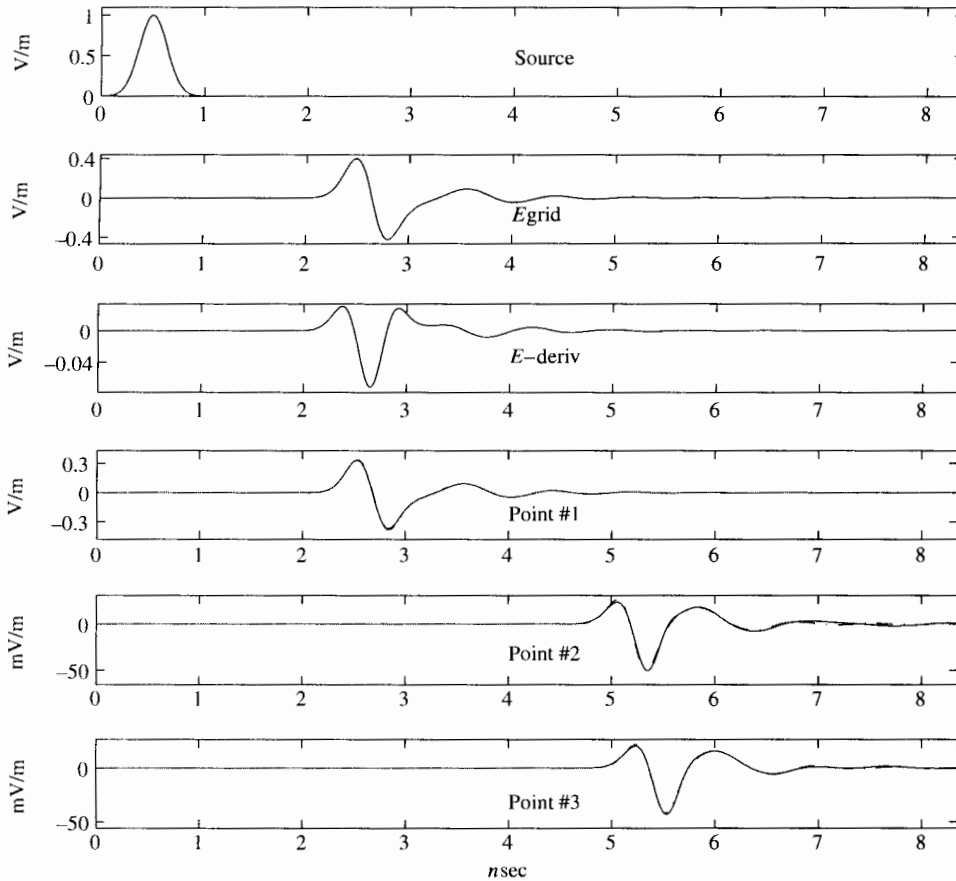
**Figure 5.9** Simulation of a plane wave hitting a slot in a metal screen, as illustrated in Fig. 5.8. After 100 time steps the plane wave is propagating toward the screen. After 150 steps, it has hit the screen, and part of it has started to propagate through the aperture. By 170 steps, part of the pulse has made it through the aperture, while the majority has been reflected. After 270 steps, the pulse resulting from the interaction with the aperture is propagating away from the slot.

for any number of points at any position in the three-dimensional space, including distances far out of reach of a reasonable FDTD computation.

It is interesting to compare the shapes of point 1 with those of points 2 and 3. Look at the Egrid plot in Fig. 5.10. Its amplitude is an order of magnitude greater than E-deriv. However, in the generating function of Eq. 5.19, the function  $E_a$  is divided by an additional term of  $R_{\Delta x}$  compared with the derivative function  $del\_E_a$ . It is not surprising that the waveform at point 1 resembles Egrid more closely than E-deriv, because it is in the near field and the additional  $1/R_{\Delta x}$  effect has not attenuated it. However, when we get to point 2, which is 80 cm away, the Egrid term has been substantially attenuated compared to E-deriv, and points 2 and 3 both resemble E-deriv more closely.

This simulation was done in the large FDTD space to verify the method. Now that we have confidence in the accuracy of the transformation, we can go back and use the much smaller problem space illustrated in Fig. 5.7 to calculate any points in the far field.

You may have already noticed one glaring disadvantage to this method: the far field is only being calculated at a small number of points. Of course, we could increase this number very substantially, but it still would not give us the time domain wave patterns that we are able to see when using FDTD. We could simply store all the time domain data that appears in the aperture. Then we could go back and generate the far field at any given time. However, the storage of all that time domain data could be a problem, particularly if the aperture is fairly large and the time domain excitation is fairly long.



**Figure 5.10** Comparison of the transformation values and the direct FDTD as illustrated in Fig. 5.8. The top graph labeled “Source” is the Gaussian plane wave pulse that impinges on the metal screen. “ $E$  deriv” is the derivative of the  $E$  field in the center of the aperture, as calculated by Eq. (5.15). Points 1, 2, and 3 show the comparison of the FDTD generated time domain data (solid line) and the time domain data generated via Eq. (5.19). The aperture was 10 cm ( $x$  direction) by 5 cm ( $y$  direction). The  $E$  field was  $x$  polarized.

One solution is to use wavelets to compress the time domain data down to a manageable set of parameters [13]. Then the parameters can be uncompressed and the time domain used to generate the far fields [14]. The use of wavelets is beyond the scope of this book. However, it is mentioned to illustrate how important the use of signal processing techniques can be in doing FDTD simulation.

### 5.2.3 FDTD Implementation of the Far Field Calculations

The file `fdd_far_field.c` printout at the end of this chapter shows some of the critical parts of the program used to calculate the far field points via the transformation of Eq. (5.19). (The coordinates used in Fig. 5.6 are the traditional coordinates used in electromagnetic theory; specifically the  $Z$  direction is the direction of propagation. Of course, we are used to using the  $Y$  direction as the direction of propagation, so the coordinate system used in `fdd_far_field.c` is essentially rotated ninety degrees compared to Fig. 5.6.) First of all, note that it would be very wasteful to stop and calculate the values for the parameters  $R_{\Delta x}$  and  $r_z$  every time, because

they consist of values that are used at every iteration. Therefore, they are calculated prior to the main FDTD loop and stored in the arrays  $R\_dist$  and  $r\_z$ , respectively. Also note that the value  $n - 2 \cdot R_{\Delta x}$  is an integer, so  $2 \cdot R_{\Delta x}$  will have to be an integer. These values are stored in the array  $N\_shift$ . The values of  $E_x(i, j, k)$  in Eq. (5.19) are calculated and stored in the array  $E_{ana}$ .

## PROBLEM SET 5.2

1. Write a program to simulate a plane wave impinging on an aperture in a metal screen. By now you should have enough experience to alter one of your previous programs. Begin with the simulation of a plane wave. Add the metal screen in the main problem space, and in the incident buffer, as shown in Fig. 5.7. Your program should have the ability to generate a plane wave and then subtract the reflection from the screen.
2. When you have the program of problem 5.2.1 running, then add the ability to put a rectangular aperture in the screen. You should be able to generate waveforms like the ones shown in Fig. 5.9.
3. Add the transformation of Eq. (5.19) to your aperture simulation. Duplicate the results of Fig. 5.10.

## REFERENCES

- [1] D. M. Sheen, S. M. Ali, M. D. Abouzahra, and J. A. Kong, Application of the three-dimensional finite-difference time-domain method to the analysis of planar microstrip circuits, *IEEE Trans. Microwave Theory Tech.*, vol. MTT-38, July 1990, pp. 446–453.
- [2] P. C. Cherry and M. F. Iskander, FDTD analysis of high frequency electronic interconnection effects, *IEEE Trans. Microwave Theory Tech.*, vol. MTT-43, Oct. 1995, pp. 2445–2451.
- [3] S. G. Garcia, T. M. Hung-Boa, R. G. Martin, and B. G. Olmedo, On the application of finite methods in time domain to anisotropic dielectric waveguides, *IEEE Trans. Microwave Theory Tech.*, vol. MTT-44, Dec. 1996, pp. 2195–2206.
- [4] A. P. Zhao, J. Juntunen, and A. V. Rasisanen, An efficient FDTD algorithm for the analysis of microstrip patch antennas printed on a general anisotropic dielectric substrate, *IEEE Trans. Microwave Theory Tech.*, vol. MTT-47, July 1999, pp. 1142–1146.
- [5] R. Harrington, *Time-Harmonic Electromagnetic Fields*, New York: McGraw-Hill, 1961.
- [6] A. Taflov, *Computational Electrodynamics: The Finite-Difference Time-Domain*. Boston: Artech House, 1995.
- [7] K. S. Yee, D. Ingham, and D. Shlager, Time-domain extrapolation to the far field based on FDTD calculations, *IEEE Trans. on Antennas and Propagation*, vol. AP-39, March 1991, pp. 410–413.
- [8] R. J. Luebbers, K. S. Kunz, M. Schneider, and F. Hunsberger, A finite-difference time-domain near zone to far zone transformation, *IEEE Trans. on Antennas and Propagation*, vol. AP-39, March, 1991, pp. 29–433.
- [9] M. Kragalott, M. S. Kluskens, and W. P. Pala, Time-domain fields exterior to a two-dimensional FDTD space, *IEEE Trans. on Antennas and Propagation*, vol. AP-45, Nov. 1997, pp. 1655–1663.
- [10] J. De Moerloose, and D. De Zutter, Surface integral representation radiation boundary condition for the FDTD method, *IEEE Trans. on Antennas and Propagation*, vol. AP-41, Nov. 1993, pp. 890–895.
- [11] D. K. Cheng, *Field and Wave Electromagnetics*, Menlo Park, CA: Addison-Wesley, 1992.
- [12] S. Silver, *Microwave Antenna Theory and Design*. London: Peter Pergrinus, Ltd., 1984.
- [13] G. Strang, and T. Nguyen, *Wavelets and Filter Banks*, Wellesley, MA: Wellesley-Cambridge Press, 1996.
- [14] D. M. Sullivan and J. Young, Far field time domain calculation from aperture radiators using the FDTD method, *IEEE Trans. on Antennas and Propagation*, submitted for review.

```

/* Fd3d_patch.c. 3D simulation of a patch antenna. */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE 62
#define JE 120
#define KE 14
#define ia 14
#define ja 15
#define ka 7
#define NFREQS 3
#define ktop 2

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 gax[IE][JE][KE],gay[IE][JE][KE],gaz[IE][JE][KE];
    int l,m,n,i,j,k,ic,jc,kc,nsteps,n_pml;
    float ddx,ra_x,ra_y,dt,T,epsz,muz,pi,eaf,npml;
    int ib,jb,kb;
    float xn,xxn,xnum,xd;
    float t0,spread,pulse;
    FILE *fp, *fopen(),*fpt;
    float ez_inc[JE],hx_inc[JE];
    float ez_low_m1,ez_low_m2,ez_high_m1,ez_high_m2;

    float idxl[ia][JE][KE],idxh[ia][JE][KE];
    float ihxl[ia][JE][KE],ihxh[ia][JE][KE];
    float idyl[IE][ja][KE],idyh[IE][ja][KE];
    float ihyl[IE][ja][KE],ihyh[IE][ja][KE];
    float idzl[IE][JE][ka],idzh[IE][JE][ka];
    float ihzl[IE][JE][ka],ihzh[IE][JE][ka];
    int ixh, jyh, kzh;

    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_e;

    int ii,jj,kk,numsph;
    float dist,xdist,ydist,zdist;

    int istart,iend,k_ref,jend,i_ref;

```

```

int j_patch_st,j_patch_end,j_ref;
int i_patch_st,i_patch_end;
float eps_sub;
float shape[IE][KE],half_wv;

ic = IE/2 ;
jc = JE/2 ;
kc = KE/2 ;
ib = IE - ia - 1;
jb = JE - ja - 1;
kb = KE - ka - 1;

i_patch_st = ia+5;
i_patch_end = i_patch_st + 31;
j_patch_end = jb - 5;
j_patch_st = j_patch_end - 39;
j_ref = j_patch_st - 30;
k_ref = ktop - 1;

pi = 3.14159;
epsz = 8.8e-12;
muz = 4*pi*1.e-7;
ddz = .265e-3; /* Cell size */
ra_y = .6625;
ra_x = .6812;
dt = ddz/6e8; /* Time steps */
printf("%12.5e %12.5e \n", ddz,dt);

INITIALIZE PARAMETERS, CALCULATE PML PARAMETERS

/* Dielectric constant of the substrate material */
eps_sub = 2.2;
printf("%f \n", eps_sub);

for ( j=0; j < JE; j++ ) {
    for ( i=0; i < IE; i++ ) {
        for ( k=0; k <= ktop; k++ ) {
            gax[i][j][k] = 1./eps_sub;
            gay[i][j][k] = 1./eps_sub;
            gaz[i][j][k] = 1./eps_sub;
        } } }

/* Add a metal plate at k =0 */

for ( i=1; i < IE-1; i++ ) {
    for ( j=1; j < JE-1; j++ ) {
        k=0 ;
        gax[i][j][k] = 0. ;
        gay[i][j][k] = 0.;
    } }

```

```

    irstart = ia + 6;
    iend   = irstart + 6;
    i_ref  = irstart + (iend - irstart )/2;
    printf("%d %d %d\n", irstart,iend,i_ref);

    half_wv = (iend - irstart)/2.;
    printf("half_wv = %5.2f\n", half_wv);

        for ( i=irstart; i <= iend; i++ ) {
            for ( k=0; k <= ktop; k++ ) {
                shape[i][k] =1.; } }

/* Add the conductor lead at k = ktop+1 */

for ( j=1; j <= j_patch_st; j++ ) {
    for ( i=irstart; i <= iend-1; i++ ) {
        gax[i][j][ktop+1] = 0.;
    }
    for ( i=irstart; i <= iend-1; i++ ) {
        gay[i][j][ktop+1] = 0.;
    } }

/* Add the rectangular patch at k = ktop */

for ( j=j_patch_st; j <= j_patch_end; j++ ) {
    for ( i=ia+1; i <= ib-1; i++ ) {
        gax[i][j][ktop+1] = 0.;
    }
    for ( i=ia+1; i <= ib-1; i++ ) {
        gay[i][j][ktop+1] = 0.;
    } }

t0 = 150.0;
spread = 25.0;
    printf( "Pulse width is %12.5e \n",spread*dt);
T = 0;
nsteps = 1;

    fpt = fopen( "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 incident buffer */

for ( j=1; j < JE; j++ ) {
    ez_inc[j] = gj3[j]*ez_inc[j]
                + gj2[j]*(.5*ra_y/eps_sub)*( hx_inc[j-1] - hx_inc[j] );
}

/* Source */

pulse = exp(-.5*(pow((t0-T)/spread,2.0) ));
ez_inc[ja-2] = pulse;
printf("%4.0f  %6.2f \n ",T,pulse);

/* Calculate the Dx field */

for ( i=1; i < ia; i++ ) {
    for ( j=1; j < JE; j++ ) {
        for ( k=1; k < KE; k++ ) {
            curl_h = ( ra_y*(hz[i][j][k] - hz[i][j-1][k])
                      - hy[i][j][k] + hy[i][j][k-1]) ;
            idxl[i][j][k] = idxl[i][j][k] + curl_h;
            dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
                + gj2[j]*gk2[k]*.5*(curl_h + gil[i]*idxl[i][j][k]);
        } } }

    for ( i=ia; i <= ib; i++ ) {
        for ( j=1; j < JE; j++ ) {
            for ( k=1; k < KE; k++ ) {
                curl_h = ( ra_y*(hz[i][j][k] - hz[i][j-1][k])
                          - hy[i][j][k] + hy[i][j][k-1]) ;
                dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
                    + gj2[j]*gk2[k]*.5*curl_h ;
            } } }

    for ( i=ib+1; i < IE; i++ ) {
        ixh = i - ib - 1;
        for ( j=1; j < JE; j++ ) {
            for ( k=1; k < KE; k++ ) {
                curl_h = ( ra_y*(hz[i][j][k] - hz[i][j-1][k])
                          - hy[i][j][k] + hy[i][j][k-1]) ;
                idxh[ixh][j][k] = idxh[ixh][j][k] + curl_h;
                dx[i][j][k] = gj3[j]*gk3[k]*dx[i][j][k]
                    + gj2[j]*gk2[k]*.5*(curl_h + gil[i]*idxh[ixh][j][k]);
            } } }

/* Calculate the Dy field */

for ( i=1; i < IE; i++ ) {
    for ( j=1; j < ja; j++ ) {
        for ( k=1; k < KE; k++ ) {

```

```

    curl_h = ( hx[i][j][k] - hx[i][j][k-1]
              - ra_x*(hz[i][j][k] - hz[i-1][j][k]) );
    idyl[i][j][k] = idyl[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]*idyl[i][j][k]);
} } }

for ( i=1; i < IE; i++ ) {
  for ( j=ja; j <= jb; j++ ) {
    for ( k=1; k < KE; k++ ) {
      curl_h = ( hx[i][j][k] - hx[i][j][k-1]
                - ra_x*(hz[i][j][k] - hz[i-1][j][k]) );
      dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
                    + gi2[i]*gk2[k]*.5* curl_h ;
    } } }

for ( i=1; i < IE; i++ ) {
  for ( j=jb+1; j < JE; j++ ) {
    jyh = j - jb - 1;
    for ( k=1; k < KE; k++ ) {
      curl_h = ( hx[i][j][k] - hx[i][j][k-1]
                - ra_x*(hz[i][j][k] - hz[i-1][j][k]) );
      idyh[i][jyh][k] = idyh[i][jyh][k] + curl_h;
      dy[i][j][k] = gi3[i]*gk3[k]*dy[i][j][k]
                    + gi2[i]*gk2[k]*.5*( curl_h + gj1[j]*idyh[i][jyh][k]);
    } } }

/* Calculate the Dz field */

for ( i=1; i < IE; i++ ) {
  for ( j=1; j < JE; j++ ) {
    for ( k=0; k < ka; k++ ) {
      curl_h = ( ra_x*(hy[i][j][k] - hy[i-1][j][k])
                - ra_y*(hx[i][j][k] - hx[i][j-1][k]) );
      idzl[i][j][k] = idzl[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]*idzl[i][j][k] );
    } } }

for ( i=1; i < IE; i++ ) {
  for ( j=1; j < JE; j++ ) {
    for ( k=ka; k <= kb; k++ ) {
      curl_h = ( ra_x*(hy[i][j][k] - hy[i-1][j][k])
                - ra_y*(hx[i][j][k] - hx[i][j-1][k]) );
      dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
                    + gi2[i]*gj2[j]*.5* curl_h ;
    } } }

for ( i=1; i < IE; i++ ) {
  for ( j=1; j < JE; j++ ) {

```

```

    for ( k=kb+1; k < KE; k++ ) {
    kzh = k - kb - 1;
        curl_h = ( ra_x*(hy[i][j][k] - hy[i-1][j][k])
                  - ra_y*(hx[i][j][k] - hx[i][j-1][k]) );
        idzh[i][j][kzh] = idzh[i][j][kzh] + curl_h;
        dz[i][j][k] = gi3[i]*gj3[j]*dz[i][j][k]
        + gi2[i]*gj2[j]*.5*( curl_h + gk1[k]*idzh[i][j][kzh] );
    } } }

/* Incident Dz */
for ( i=istart; i <= iend; i++ ) {
    for ( k=0; k <= ktop; k++ ) {
dz[i][ja][k] = dz[i][ja][k]
                + (.5/eps_sub)*shape[i][k]*hx_inc[ja-1];
    } }

/* Calculate the E from D field */

/* Ex and Ey are zero at k = 0. */
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];
        } } }

    /* Write out the time domain data at the input port */

    fprintf( fpt,"%8.4f \n",ez[i_ref][j_ref][k_ref]);

/* Calculate the incident field */

    for ( j=0; j < JE-1; j++ ) {
        hx_inc[j] = fj3[j]*hx_inc[j]
                + .5*fj2[j]*( ez_inc[j] - ez_inc[j+1]
                );

/* Calculate the Hx field */

for ( i=0; i < ia; 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]
                      - ra_y*(ez[i][j+1][k] - ez[i][j][k]) );
            ihx1[i][j][k] = ihx1[i][j][k] + curl_e;
            hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
            + fj2[j]*fk2[k]*.5*( curl_e + fil[i]*ihx1[i][j][k] );
        } } }

```

```

for ( i=ia; i <= ib; 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]
        - ra_y*(ez[i][j+1][k] - ez[i][j][k]) );
      hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
        + fj2[j]*fk2[k]*.5*curl_e ;
    } } }

for ( i=ib+1; i < IE; i++ ) {
  ixh = i - ib-1;
  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]
        - ra_y*(ez[i][j+1][k] - ez[i][j][k]) );
      ihxh[ixh][j][k] = ihxh[ixh][j][k] + curl_e;
      hx[i][j][k] = fj3[j]*fk3[k]*hx[i][j][k]
        + fj2[j]*fk2[k]*.5*( curl_e + fil[i]*ihxh[ixh][j][k] );
    } } }

/* Incident Hx */
for ( i=istart; i <= iend; i++ ) {
  for ( k=0; k <= ktop; k++ ) {
    hx[i][ja-1][k] = hx[i][ja-1][k]
      + (.5/eps_sub)*shape[i][k]*ez_inc[ja];
  } }

/* Calculate the Hy field */

for ( i=0; i < IE-1; i++ ) {
  for ( j=0; j < ja; j++ ) {
    for ( k=0; k < KE-1; k++ ) {
      curl_e = ( ra_x*(ez[i+1][j][k] - ez[i][j][k])
        - ex[i][j][k+1] + ex[i][j][k] ) ;
      ihyl[i][j][k] = ihyl[i][j][k] + curl_e ;
      hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
        + fi2[i]*fk2[k]*.5*( curl_e + fj1[j]*ihyl[i][j][k] );
    } } }

for ( i=0; i < IE-1; i++ ) {
  for ( j=ja; j <= jb; j++ ) {
    for ( k=0; k < KE-1; k++ ) {
      curl_e = ( ra_x*(ez[i+1][j][k] - ez[i][j][k])
        - ex[i][j][k+1] + ex[i][j][k] ) ;
      hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
        + fi2[i]*fk3[k]*.5*curl_e ;
    } } }

for ( i=0; i < IE-1; i++ ) {
  for ( j=jb+1; j < JE; j++ ) {

```

```

    jyh = j - jb-1;
    for ( k=0; k < KE-1; k++ ) {
        curl_e = ( ra_x*(ez[i+1][j][k] - ez[i][j][k])
                  - ex[i][j][k+1] + ex[i][j][k] );
        ihyh[i][jyh][k] = ihyh[i][jyh][k] + curl_e ;
        hy[i][j][k] = fi3[i]*fk3[k]*hy[i][j][k]
          + fi2[i]*fk2[k]*.5*( curl_e + fj1[j]*ihyh[i][jyh][k] );
    } } }

/* Calculate the Hz field */

for ( i=0; i < IE-1; i++ ) {
    for ( j=0; j < JE-1; j++ ) {
        for ( k=0; k < ka; k++ ) {
            curl_e = ( ra_y*(ex[i][j+1][k] - ex[i][j][k])
                      - ra_x*(ey[i+1][j][k] - ey[i][j][k]) );
            ihz1[i][j][k] = ihz1[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]*ihz1[i][j][k] );
        } } }

for ( i=0; i < IE-1; i++ ) {
    for ( j=0; j < JE-1; j++ ) {
        for ( k=ka; k <= kb; k++ ) {
            curl_e = ( ra_y*(ex[i][j+1][k] - ex[i][j][k])
                      - ra_x*(ey[i+1][j][k] - ey[i][j][k]) );
            hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
              + fi2[i]*fj2[j]*.5*curl_e ;
        } } }

for ( i=0; i < IE-1; i++ ) {
    for ( j=0; j < JE-1; j++ ) {
        for ( k=kb+1; k < KE; k++ ) {
            kzh = k - kb - 1;
            curl_e = ( ra_y*(ex[i][j+1][k] - ex[i][j][k])
                      - ra_x*(ey[i+1][j][k] - ey[i][j][k]) );
            ihzh[i][j][kzh] = ihzh[i][j][kzh] + curl_e;
            hz[i][j][k] = fi3[i]*fj3[j]*hz[i][j][k]
              + fi2[i]*fj2[j]*.5*( curl_e + fk1[k]*ihzh[i][j][kzh] );
        } } }

}

/* ---- 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][ktop] );
    }
    fprintf( fp, " \n" );
}
fclose( fp );

printf("T = %4.0f \n", T);
}

fclose( fpt );
}
```

```

/* Fd3d_far_field.c */

/* These arrays are calculated before the main loop */
/* ipos, jpos, and kpos are arrays which specify the location
   at which the far field values will be calculated.
   iwid and kwid specify the size of the slot in the I and k
   directions, respectively.*/

for ( l=1; l <= 3;l++ ) {
  for ( i = ic-iwid; i <= ic+iwid; i++ ) {
    for ( k = kc-kwid; k <= kc+kwid; k++ ) {
      R_dist[i][k][l] = sqrt( pow( (ic - ipos(l)-i),2.)
                             + pow( (kc - kpos(l)-k),2.) ));
      r_z[i][k][l] = jpos/R_dist[i][k][l];
      N_shift[i][k][l] = 2*R_dist[i][k][l];
    } }

/* Just before the new value of Ez is calculated from the Flux
   density, the old value is stored in ezml */
/* jgrid_end is the location of the slot antenna in the
   j direction */

for ( i = ic-iwid; i <= ic+iwid; i++ ) {
  for ( k = kc-kwid; k <= kc+kwid; k++ ) {
    ezml[i][k] = ez[i][jgrid_end][k];
  } }

/* Here is the calculation of the of the far field values.
   This is done right after the calculation of the Ez field. */

for ( l=1; l <= 3;l++ ) {
  for ( i = ic-iwid; i <= ic+iwid; i++ ) {
    for ( k = kc-kwid; k <= kc+kwid; k++ ) {
      Ena[l][iT + N_shift[i][k][l]] = Eana[l][iT + N_shift[i][k][l] ]
      + ( r_z[i][k][l]/pi)*
      ( ez[i][jgrid_end][k] - ezml[i][k] )/R_dist[i][k][l]
      + ez[i][jgrid_end][k]/(2*pow(R_dist[i][k][l]),2.)
    } } }

```





# 6

## Using FDTD for Other Types of Simulation

In this chapter, we will discuss the use of the FDTD method for applications other than electromagnetics. The first application is an acoustic simulation problem. The mathematics of acoustics is similar to electromagnetics [1], so it is natural that the developments of FDTD are used in acoustic simulation [2, 3]. The second example is the simulation of the Schroedinger equation, which is at the heart of quantum mechanics. Quantum mechanics is radically different from both acoustics and electromagnetics. However, the Schroedinger equation is still a wave equation and it lends itself to simulation with FDTD. In both cases, only simple examples of one-dimensional simulations are presented.

### 6.1 THE ACOUSTIC FDTD FORMULATION

In the following development, we will deal only with pressure waves and ignore elastic waves. We start with the first-order acoustic equations,

$$\kappa \frac{\partial}{\partial t} p(\mathbf{x}, t) = \nabla \cdot \mathbf{u} \quad (6.1a)$$

$$\rho_0 \rho_r \frac{d}{dt} \mathbf{u}(\mathbf{x}, t) = \nabla p(\mathbf{x}, t), \quad (6.1b)$$

where  $p(\mathbf{x}, t)$  is the pressure field  $[F/m^2] = [kg/(m - sec^2)]$ ,  
 $\mathbf{u}(\mathbf{x}, t)$  is the vector velocity field  $[m/sec]$ ,  
 $\rho_0$  is the mass density of water  $[kg/m^3]$ ,  
 $\rho_r$  is the relative (to water) mass density of the medium,  
 $\kappa$  is the compressibility of the medium  $[(m - sec^2)/kg]$ . (Table 6.1)

Notice that water has been chosen as the background medium instead of air. The compressibility is

$$\kappa = \frac{1}{\rho \cdot c^2} = \frac{1}{\rho_0 \cdot \rho_r \cdot c^2}, \quad (6.2)$$

where  $c$  is the velocity of sound.

TABLE 6.1 Acoustic Properties of a Few Materials

Material	Speed of Sound (m/sec)	Density (kg/m <sup>3</sup> )	Compressibility (m · s <sup>2</sup> /kg)
1. Water	1500	1000	4.4 × 10 <sup>-8</sup>
2. Air	343	1.21	7.02 × 10 <sup>-6</sup>
3. Metal	5900	7800	

Eq. (6.1a) can be written

$$\frac{dp(x, y, z, t)}{dt} = \frac{1}{\kappa(x, y, z)} \left[ \frac{du_x(x, y, z, t)}{dx} + \frac{du_y(x, y, z, t)}{dy} + \frac{du_z(x, y, z, t)}{dz} \right].$$

We will use a differencing scheme similar to Yee's FDTD for electromagnetics [4]; however, we will assume that the pressure locations are at nodes in a three-dimensional lattice, and that the velocities are located between the nodes, as shown in Fig. 6.1.

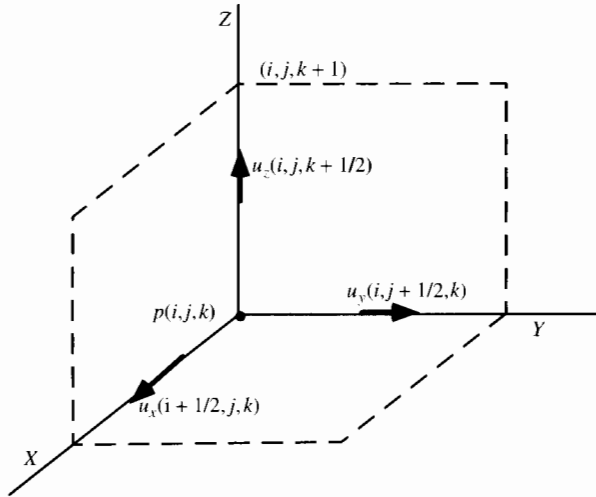


Figure 6.1 The acoustic Yee cell.

Using first-order differencing in time and space, Eq. (6.1.a) can be written

$$\begin{aligned} \frac{p^{n+1/2}(i, j, k) - p^{n-1/2}(i, j, k)}{\Delta t} &= \frac{1}{\kappa(i, j, k)} \left[ \frac{u_x^n(i + 1/2, j, k) - u_x^n(i - 1/2, j, k)}{\Delta x} \right] \\ &+ \frac{1}{\kappa(i, j, k)} \left[ \frac{u_y^n(i, j, k + 1/2) - u_y^n(i, j, k - 1/2)}{\Delta y} \right] \\ &+ \frac{1}{\kappa(i, j, k)} \left[ \frac{u_z^n(i, j, k + 1/2) - u_z^n(i, j, k - 1/2)}{\Delta z} \right]. \end{aligned}$$

First, the compressibility will be replaced by Eq. (6.2). Then  $\Delta t$  will be moved to the right side, giving a difference equation suitable to the FDTD formulation:

$$\begin{aligned} p^{n+1/2}(i, j, k) &= p^{n-1/2}(i, j, k) + \frac{\Delta t \cdot \rho_0 \rho_r c^2}{\Delta x} \cdot [u_x^n(i + 1/2, j, k) - u_x^n(i - 1/2, j, k)] \\ &+ \frac{\Delta t \cdot \rho_0 \rho_r c^2}{\Delta y} \cdot [u_y^n(i, j + 1/2, k) - u_y^n(i, j - 1/2, k)] \\ &+ \frac{\Delta t \cdot \rho_0 \rho_r c^2}{\Delta z} \cdot [u_z^n(i, j, k + 1/2) - u_z^n(i, j, k - 1/2)]. \end{aligned} \quad (6.3)$$

A similar procedure in Eq. (6.1b) yields three equations of the type

$$u_z^{n+1/2}(i, j, k + 1/2) = u_z^{n-1/2}(i, j, k + 1/2) + \frac{\Delta t}{\rho_r(i, j, k + 1/2) \cdot \rho_0 \cdot \Delta z} \cdot [p^n(i, j, k + 1) - p^n(i, j, k)]. \quad (6.4)$$

Obviously, equations in the  $X$  and  $Y$  direction would be similar. We will limit ourselves to a simple one-dimensional problem in the  $Z$  direction and rewrite Eqs. (6.3) and (6.4) as follows:

$$p^{n+1/2}(k) = p^{n-1/2}(k) + ga(k) \cdot [u_z^n(k + 1/2) - u_z^n(k - 1/2)] \quad (6.5a)$$

$$u_z^{n+1}(k + 1/2) = u_z^n(k + 1/2) + gb(k + 1/2) \cdot [p^{n+1/2}(k + 1) - p^{n+1/2}(k)], \quad (6.5b)$$

where we have the parameters

$$ga(k) = \frac{\Delta t \cdot \rho_0 \rho_r \cdot c^2}{\Delta z} \quad (6.6a)$$

$$gb(k + 1/2) = \frac{\Delta t}{\rho(k + 1/2) \cdot \rho_0 \cdot \Delta z}. \quad (6.6b)$$

Notice that we have chosen to write  $ga$  in terms of the speed of sound and the pressure rather than the compressibility since these are the parameters that are most widely used.

As before,  $\Delta t$  is chosen after  $\Delta z$  is chosen according to

$$\Delta t \leq \frac{\Delta z}{c_{\max}},$$

where  $c_{\max}$  is the fastest velocity of sound that we encounter. We will suppose that this will be metal, in which the velocity of sound is 5900 meters per second. Just to give ourselves a margin, we will take

$$\Delta t = \frac{\Delta z}{10^4}.$$

Since  $\rho_0 = 1000$  kg/m, Eqs. (6.6) become

$$ga(k) = 10^{-1} \cdot \rho_r(k) c^2(k) \quad (6.7a)$$

$$gb(k + 1/2) = \frac{10^{-7}}{\rho_r(k + 1/2)}. \quad (6.7b)$$

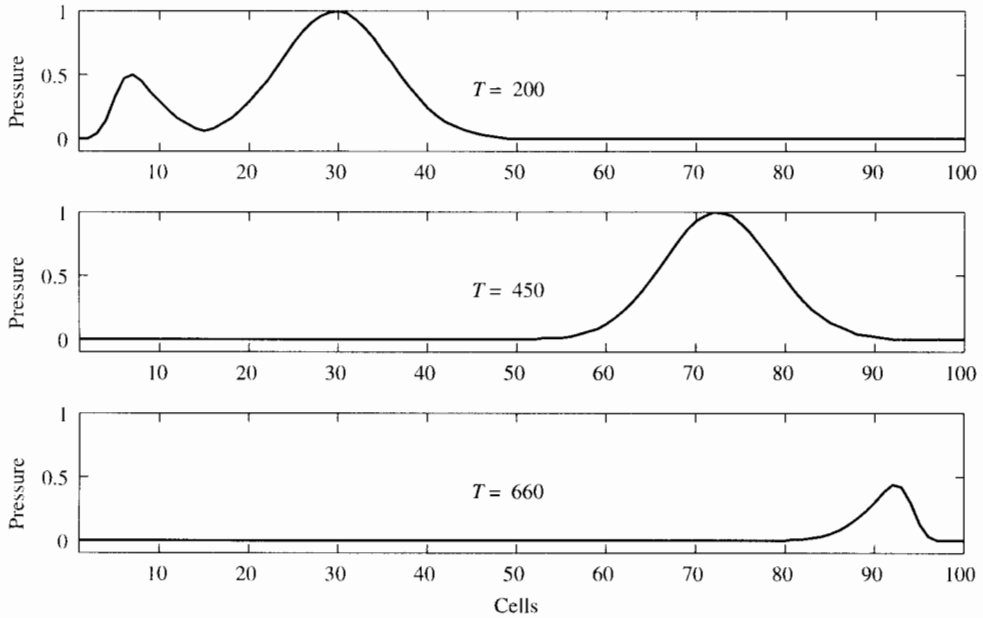
For water, which we will use as our background medium, Eqs. (6.7a) and (6.7b) turn out to be

$$ga(k) = 10^{-1} \cdot \rho_r(k) c_r^2(k) = 10^{-1} \cdot 1 \cdot (1,500)^2 = 2.25 \times 10^5 \quad (6.8a)$$

$$gb(k + 1/2) = 10^{-7}. \quad (6.8b)$$

The program `us1.c` at the end of the chapter is a one-dimensional acoustic simulation program. In its present form, it is only simulating water. Figure 6.2 shows the result. A pulse is generated at one end and subtracted out at the other. So where did the PML come from?

Here is how this program was developed: I went back and made a copy of the program `fd1d_1.1.c`. Using the text editor, I replaced `ex` with `p`, and `hy` with `u`. Then I went to the program we developed in Chapter 5 to simulate microstrip antennas, and I took the PML from the 1D incident buffer and copied it over. I added the parameters `ga` and `gb`, changed some constants, and so forth, and there it was! The entire process took about 30 minutes. The point is, an acoustic simulation program, complete with PML, was written by using the ideas we had already developed for EM simulation.



**Figure 6.2** Simulation of an acoustic pulse. The pulse is generated at cell 15 and propagates outward in both directions. After 200 time steps, the part that propagated to the left is being absorbed by the PML. Similarly, after 660 time steps, the part that propagated to the right is being absorbed.

### PROBLEM SET 6.1

1. Develop your own program `us1.c` using the procedure described and get it running.
2. Put a metal sheet of about 10 cells in the center of the problem space and see what happens. Then put in an air pocket of 10 cells and see what happens.
3. Repeat the above problem, but for a sinusoid waveform of 2.5 MHz in a Gaussian envelope. In determining the cell size remember: 10 points per wavelength of the shortest wavelength in the problem. Since metal has the highest speed, it will have the shortest wavelength.

## 6.2 SIMULATION OF THE SCHROEDINGER EQUATION

In this section a simple formulation of the Schroedinger equation in FDTD is described. The Schroedinger equation is the basis of quantum mechanics. If you do not know any quantum mechanics, this section may prove to be a little puzzling. (An excellent text for beginning quantum mechanics is the one by Griffiths [5].) Problems involving quantum mechanics, unlike the previous section where it wasn't difficult to pick up the basics of acoustics, are quite different. For instance, the parameter used in our wave equations is  $\psi(r, t)$ , which, like the electric field or the pressure, is a function of space and time. But unlike the electric field or pressure, it is a complex number, even though it is a time domain parameter. It is also difficult to attribute any distinct physical meaning to  $\psi$ . It is the product of  $\psi$  with its complex conjugate,  $\psi(r, t)^* \psi(r, t)$ , that indicates the probability of a particle being at the point  $r$  at time  $t$ .

### 6.2.1 Formulating the Schroedinger Equation into FDTD

The time-dependent Schroedinger equation is given by [5–8]

$$i\hbar \frac{\partial \psi(r, t)}{\partial t} = -\frac{\hbar^2}{2m} \nabla^2 \psi(r, t) + V(r, t) \cdot \psi(r, t) \quad (6.9)$$

or

$$\frac{\partial \psi(r, t)}{\partial t} = i \frac{\hbar}{2m} \nabla^2 \psi(r, t) - \frac{i}{\hbar} V(r, t) \cdot \psi(r, t), \quad (6.10)$$

where  $m$  is the mass of the particle [kg],

$\hbar = 1.054 \times 10^{-34}$  [J – sec] is Plank’s constant,

$V$  is the potential [J]. (It will often be more convenient to express the potential in electron volts (eV), where  $1 \text{ eV} = 1.602 \times 10^{-19} \text{ J}$ .)

To avoid using complex numbers, we will split the variable  $\psi(r, t)$  into its real and imaginary components

$$\psi(r, t) = \psi_{\text{real}}(r, t) + i\psi_{\text{imag}}(r, t). \quad (6.11)$$

Inserting Eq. (6.11) into Eq. (6.10) and separating real and imaginary parts results in the following coupled set of equations:

$$\frac{\partial \psi_{\text{real}}(r, t)}{\partial t} = -\frac{\hbar}{2m} \cdot \nabla^2 \psi_{\text{imag}}(r, t) + \frac{1}{\hbar} V \psi_{\text{imag}}(r, t) \quad (6.12a)$$

$$\frac{\partial \psi_{\text{imag}}(r, t)}{\partial t} = \frac{\hbar}{2m} \cdot \nabla^2 \psi_{\text{real}}(r, t) - \frac{1}{\hbar} V \psi_{\text{real}}(r, t). \quad (6.12b)$$

We begin by assuming a one-dimensional space, arbitrarily taking the  $z$  direction. Starting with Eq. (6.12a), the finite difference approximations in space and time result in

$$\begin{aligned} & \frac{\psi_{\text{real}}^n(k) - \psi_{\text{real}}^{n-1}(k)}{\Delta t} \\ &= -\frac{\hbar}{2m} \frac{\psi_{\text{imag}}^{n-1/2}(k+1) - 2\psi_{\text{imag}}^{n-1/2}(k) + \psi_{\text{imag}}^{n-1/2}(k-1)}{(\Delta z)^2} + \frac{1}{\hbar} V(k) \cdot \psi_{\text{imag}}^{n-1/2}(k), \end{aligned}$$

from which we get

$$\begin{aligned} \psi_{\text{real}}^n(k) &= \psi_{\text{real}}^{n-1}(k) \\ &\quad - \frac{\Delta t}{\Delta z^2} \frac{\hbar}{2m} \left[ \psi_{\text{imag}}^{n-1/2}(k+1) - 2\psi_{\text{imag}}^{n-1/2}(k) + \psi_{\text{imag}}^{n-1/2}(k-1) \right] \\ &\quad + \frac{\Delta t}{\hbar} V(k) \cdot \psi_{\text{imag}}^{n-1/2}(k). \end{aligned} \quad (6.13)$$

We assume we are simulating an electron and therefore we will take  $m$  to be the mass of an electron. We will take  $\Delta z$  to be one-tenth of an Angstrom, so

$$m = 9.1 \times 10^{-31} \text{ [kg]}$$

$$\Delta z = 1 \times 10^{-11} \text{ [m]}.$$

At this point, we don’t know what the time step,  $\Delta t$ , should be. Unfortunately, there is no specific Courant condition to guide us, as was the case in EM simulation. We will start by assigning what seems to be a “reasonable” number, i.e., something that will likely keep

Eq. (6.13) from blowing up. Let's just take 1/8 for now:

$$\frac{\Delta t}{\Delta z^2} \frac{\hbar}{2m} = \frac{1}{8}, \quad (6.14)$$

which means

$$\Delta t = \frac{1}{8} \frac{2m}{\hbar} \Delta z^2 = \frac{1}{4} \cdot \frac{9.1 \times 10^{-31} \text{ kg}}{1.055 \times 10^{-34} \text{ J} \cdot \text{sec}} \cdot (10^{-11} \text{ m})^2 = 2.165 \times 10^{-19} \text{ sec.}$$

Now the constant in front of the potential term can be calculated:

$$\frac{\Delta t}{\hbar} = \frac{2.165 \times 10^{-19} \text{ sec}}{1.055 \times 10^{-34} \text{ J} \cdot \text{sec}} = 2.053 \times 10^{15} \text{ J}^{-1} \cdot \left( \frac{1.602 \times 10^{-19} \text{ J}}{1 \text{ eV}} \right) = 3.285 \times 10^{-4} \text{ eV}^{-1}.$$

The two coupled equations can then be written

$$\begin{aligned} \psi_{\text{real}}^n(k) &= \psi_{\text{real}}^{n-1}(k) \\ &\quad - \frac{1}{8} \left[ \psi_{\text{imag}}^{n-1/2}(k+1) - 2\psi_{\text{imag}}^{n-1/2}(k) + \psi_{\text{imag}}^{n-1/2}(k-1) \right] \\ &\quad + \frac{\Delta t}{\hbar} V(k) \cdot \psi_{\text{imag}}^{n-1/2}(k) \end{aligned} \quad (6.15a)$$

$$\begin{aligned} \psi_{\text{imag}}^{n+1/2}(k) &= \psi_{\text{imag}}^{n-1/2}(k) \\ &\quad + \frac{1}{8} \left[ \psi_{\text{real}}^n(k+1) - 2\psi_{\text{real}}^n(k) + \psi_{\text{real}}^n(k-1) \right] \\ &\quad - \frac{\Delta t}{\hbar} V(k) \cdot \psi_{\text{real}}^n(k). \end{aligned} \quad (6.15b)$$

To simulate a particle moving in free space, it is necessary to specify both the real and imaginary parts in the spatial domain. For example, we will initiate a particle at a wavelength of  $\lambda$  in a Gaussian envelope of width  $\sigma$  with the following two equations:

$$\psi_{\text{real}}(k) = e^{-.5 \cdot \left( \frac{k-k_0}{\sigma} \right)^2} \cos \left( \frac{2\pi(k-k_0)}{\lambda} \right) \quad (6.16a)$$

$$\psi_{\text{real}}(k) = e^{-.5 \cdot \left( \frac{k-k_0}{\sigma} \right)^2} \sin \left( \frac{2\pi(k-k_0)}{\lambda} \right), \quad (6.16b)$$

where  $k_0$  is the center of the pulse. Once these equations are calculated, the amplitudes must be normalized [5], to satisfy

$$\int_{-\infty}^{\infty} \psi^*(x) \cdot \psi(x) dx = 1. \quad (6.17)$$

## 6.2.2 Calculating the Expectation Values of the Observables

Two quantities of importance in quantum mechanics are the expected values of the kinetic energy and the potential energy. They are calculated from  $\psi(x, t)$  as follows:

**Kinetic energy.** The expected value of the kinetic energy is given by

$$\langle T \rangle = \left\langle \psi \left| -\frac{\hbar^2}{2m} \frac{\partial^2}{\partial z^2} \right| \psi \right\rangle = -\frac{\hbar^2}{2m} \int_{-\infty}^{\infty} \left( \psi^* \frac{\partial^2 \psi}{\partial z^2} \right) dz.$$

The Laplacian operator  $\frac{\partial^2}{\partial z^2}$  is approximated by

$$\frac{\partial^2 \psi_{\text{real}}(k)}{\partial z^2} \cong [\psi_{\text{real}}(k-1) - 2\psi_{\text{real}}(k) + \psi_{\text{real}}(k+1)] / \Delta z^2$$

$$\frac{\partial^2 \psi_{\text{imag}}(k)}{\partial z^2} \cong [\psi_{\text{imag}}(k-1) - 2\psi_{\text{imag}}(k) + \psi_{\text{imag}}(k+1)] / \Delta z^2.$$

The kinetic energy in the simulation is calculated by

$$\langle T \rangle = -\frac{\hbar}{2 \cdot m_e} \sum_{i=1}^N \left\{ [\psi_{\text{real}}(k) - i \psi_{\text{imag}}(k)] \cdot \left[ \frac{\partial^2 \psi_{\text{real}}(k)}{\partial z^2} + i \frac{\partial^2 \psi_{\text{imag}}(k)}{\partial z^2} \right] \right\}. \quad (6.18)$$

**Potential energy.** The expected value of the potential energy is

$$\langle V \rangle = \langle \psi | V | \psi \rangle = \int_{-\infty}^{\infty} V(z) |\psi(z, t)|^2 dz,$$

which is calculated by

$$\langle V \rangle = \sum_{i=1}^N V(k) \cdot [\psi_{\text{real}}^2(k) + \psi_{\text{imag}}^2(k)], \quad (6.19)$$

where  $V(k)$  is the potential at that cell.

### 6.2.3 Simulation of an Electron Striking a Potential Barrier

Figure 6.3 shows a simulation of an electron moving in free space next to a region with a potential of 100 eV. It is initiated at 146 eV, which is pure kinetic energy because there is no potential. At 350 time steps, it has propagated to the right. The waveform has begun to spread, but the total kinetic energy remains the same. After 1300 time steps, it has struck the potential. Part of the waveform has penetrated the potential barrier, and part has been reflected. Notice that the total energy has remained the same, but now part of it is in the form of potential energy, because part of the waveform is at a potential of 100 eV. Does this mean the electron has split into two parts? No. It means there is a certain probability that it has penetrated the barrier and a certain probability that it has been reflected. These probabilities are calculated by the following:

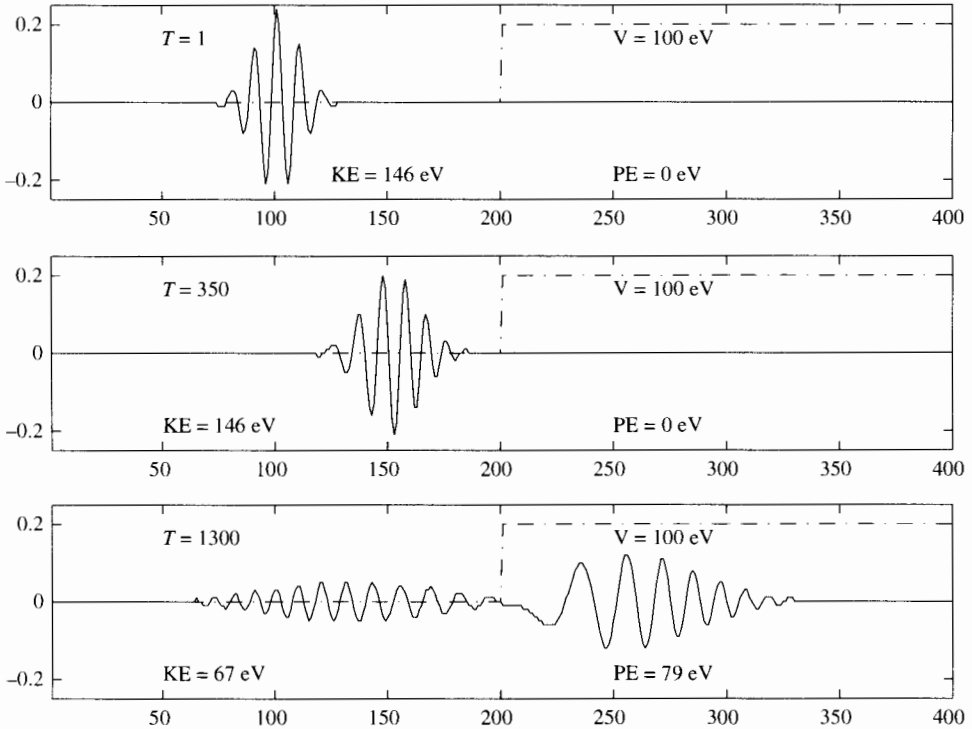
$$\text{Probability of reflection} = \int_{-\infty}^{x_{\text{Pot.}}} \psi^*(x) \cdot \psi(x) dx \quad (6.20a)$$

$$\text{Probability of transmission} = \int_{x_{\text{Pot.}}}^{\infty} \psi^*(x) \cdot \psi(x) dx, \quad (6.20b)$$

where  $x_{\text{Pot.}}$  is the location where the potential begins. For this particular problem, Eq. (6.20a) calculated 0.206 and Eq. (6.19b) calculated 0.794. Of course, the two must sum to one.

### PROBLEM SET 6.2

1. The program `se.c` at the end of this section was used to do the simulation in Fig. 6.3. Get this program running and duplicate the results.
2. Repeat the simulation of problem 6.2.1, but use a potential of  $-100$  eV.
3. Repeat the simulation of problem 6.2.1 using a potential of 1000 eV. What happens? Does this make sense?



**Figure 6.3** Simulation of an electron moving in free space and then hitting a potential. While the electron is in free space, all the energy is kinetic. After it strikes the potential, part of the energy has been converted to potential energy. The waveform at  $T = 1300$  indicates that there is some probability that the electron was reflected and some probability that it penetrated the potential barrier.

## REFERENCES

- [1] L. Beranek, *Acoustics*. New York: McGraw-Hill, 1954.
- [2] J. J. Bowman, T. B. A. Senior, and P. L. E. Uslenghi, *Electromagnetic and Acoustical Scattering by Simple Shapes*. New York: Hemisphere, 1987.
- [3] J. G. Maloney and K. E. Cummings, Adaptation of FDTD techniques to acoustic modeling, *11th Annual Review of Progress in Applied Computational Electromagnetics*, Monterey, CA, vol. 2, March 1995, pp. 724–731.
- [4] K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antennas and Propagat.*, vol. 17, 1966, pp. 585–589.
- [5] D. J. Griffiths, *Introduction to Quantum Mechanics*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [6] C. Cohen-Tannoudji, B. Diu, and F. Laloe, *Quantum Mechanics*. New York: Wiley, 1977.
- [7] S. Brandt and H. D. Dahmen, *The Picture Book of Quantum Mechanics*. Berlin: Springer Verlag, 1995.
- [8] S. Borowitz, *Fundamentals of Quantum Mechanics*. New York: W. A. Benjamin, 1967.



```

/* us1.c. 1D FDTD Acoustic simulation */

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

#define KE 100

main ()
{
    float p[KE],u[KE];
    int n,k,kc,ke,NSTEPS,n_pml;
    float ddx,dt,T;
    float t0,spread,pulse,xn,xxn,npml;
    FILE *fp, *fopen();
    float g2[KE],g3[KE],f2[KE],f3[KE];

    /* Initialize */
    for ( k=1; k < KE; k++ )
    { p[k] = 0.;
      u[k] = 0.;
      g2[k] = 1.;
      g3[k] = 1.;
      f2[k] = 1.;
      f3[k] = 1.;
    }

    /* PML */
    npml = 10;
    n_pml = npml;
    for( k=0; k < n_pml; k++ ) {
        xxn = (npml-k)/npml;
        xn = .33*pow(xxn,3.);
        g2[k] = 1./(1.+xn);
        g2[KE-k-1] = 1./(1.+xn);
        g3[k] = (1.-xn)/(1.+xn);
        g3[KE-k-1] = (1.-xn)/(1.+xn);
        xxn = (npml-k-.5)/npml;
        xn = .33*pow(xxn,3.);
        f2[k] = 1./(1.+xn);
        f2[KE-k-2] = 1./(1.+xn);
        f3[k] = (1.-xn)/(1.+xn);
        f3[KE-k-2] = (1.-xn)/(1.+xn);
    }

    printf( "f & g \n");
    for ( k=0; k < KE; k++ ) {
        printf( "%2d %6.4f %6.4f \n",k,g2[k],g3[k]);
        printf( "    %6.4f %6.4f \n",f2[k],f3[k]);}
}

```

```

kc = KE/2;
t0 = 150.0;
spread = 50;
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 Pressure */
        for ( k=1; k < KE; k++ )
        { p[k] = g3[k]*p[k] + g2[k]*(2.25e5)*( u[k-1] - u[k] ) ; }

        /* Put a Gaussian pulse in the middle */

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

        /* Calculate the velocity */
        for ( k=0; k < KE-1; k++ )
        { u[k] = f3[k]*u[k] + f2[k]*(6.67e-8)*( p[k] - p[k+1] ) ; }

    }

    /* End of the Main FDTD Loop */

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

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

    /* Write the U field out to a file "Vel" */
    fp = fopen( "Vel","w");
    for ( k=1; k <= KE; k++ )

```

```
    { fprintf( fp, " %6.2f \n",u[k]); }  
    fclose(fp);  
  
    printf( "T = %5.0f\n",T);  
}  
  
}
```

```

/* se.c. 1D FDTD Schroedinger simulation */

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

#define KE 400

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;
    float lambda,sigma,ptot;
    float ddx,dt,ra,T;
    float pulse,xn,xxn,npml;
    FILE *fp, *fopen();
    float lap_rl,lap_im,ke_rl,ke_im,kine,PE;

    /* Initialize */
    for ( k=0; k < KE; k++ )
        { psi_rl[k] = 0.;
          psi_im[k] = 0.;
          vp[k] = 0.;
        }

    pi = 3.14159;
    melec = 9.2e-31; /* Mass of an electron */
    hbar = 1.055e-34; /* Plank's constant */

    ddx = .1e-10; /* Cell size */
    ra = 1./8.;
    printf( "ra = %e \n",ra);
    dt = .25*(melec/hbar)*pow(ddx,2.);
    printf( " dt = %e \n",dt);

    printf( "kstart,Vpot (in eV) --> ");
    scanf("%d",&kstart);
    scanf("%f",&Vpot);
    for ( k=kstart; k < KE; k++ )
        { vp[k] = Vpot*1.602e-19; } /* Convert eV to Joules */

    /* Write the Potential to a file "Vpot" */
    fp = fopen( "Vpot","w");
    for ( k=0; k < KE; k++ )
        { fprintf( fp," %6.2f \n",vp[k]*6.2415e18); /* Write in eV*/
          printf(" %6.2f \n",(dt/hbar)*vp[k]); }
    fclose(fp);
}

```

```

    /* Initialize the pulse */
    printf( "lambda,sigma --> ");
    scanf("%f ", &lambda );
    scanf("%f", &sigma);
    kcenter = kstart/2;
    printf( "kcenter = %3d \n",kcenter);
    ptot = 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.) );
      ptot = ptot + pow(psi_rl[k],2.) + pow(psi_im[k],2.); }

    printf( " ptot= %f \n",ptot);

    /* Normalize the waveform */
    for ( k=1; k < kstart; k++ )
    { psi_rl[k] = psi_rl[k]/sqrt(ptot);
      psi_im[k] = psi_im[k]/sqrt(ptot); }

    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++ )
    { 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] ; }

    /* Calculate the imaginary part */

    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 */

/* Print out the real and imaginary parts */
for ( k=0; k < KE; k++ )
{ printf( "%3d   %6.2f   %6.2f\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,"   %6.2f \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.;

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(ddd,2.))
    *sqrt(pow(ke_rl,2.) + pow(ke_im,2.));
printf( " ke = %6.3f eV \n",6.23e18*kine); /* convert J to eV */
printf( " pe = %6.3f eV \n",6.23e18*PE);

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

```

# A

## The Z Transform

One of the most useful techniques in engineering analysis is transforming a problem from the time domain to the frequency domain. Using a Fourier transform, differential equations are changed to algebraic equations, often substantially simplifying the analysis. When dealing with linear systems, the relationship between the input and output is a convolution integral. However, this reduces to simple multiplication in the frequency domain. When dealing with transient signals, it is often more convenient to use the Laplace transform, but the principle is the same.

The Laplace and Fourier transforms are appropriate for analog signals. When dealing with digital signals, the Z transform is used. The reasons for using it are analogous: (1) complicated difference equations in the time domain become algebraic equations in the Z domain, and (2) the relationship between the input and output of a linear system is a multiplication in the Z domain instead of a convolution in the sampled time domain.

### A.1 DEFINITION OF THE Z TRANSFORM

The Z transform is extremely useful when dealing with functions in the sampled time domain; i.e., instead of the function  $x(t)$ , we have a function of the type

$$x(n) = \sum_{n=-\infty}^{\infty} x(t)\delta(t - n \cdot \Delta t),$$

where  $\Delta t$  is a uniform time interval, and  $\delta$  is the Dirac delta function:

$$\begin{aligned}\delta(t) &= 1 \quad \text{if } t = 0 \\ &= 0 \quad \text{elsewhere.}\end{aligned}$$

The Z transform is defined by

$$Z[x(n)] = X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}. \quad (\text{A.1})$$

A power of  $z^{-1}$  is associated with each delay interval  $\Delta t$ . Suppose we have a function  $x(n)$ ,

$$x(n) = \delta(t) + .5 \cdot \delta(t - \Delta t) + .25 \cdot \delta(t - 2 \cdot \Delta t). \quad (\text{A.2})$$

In the Z domain, this is written

$$X(z) = 1 + .5z^{-1} + .25z^{-2}. \quad (\text{A.3})$$

Note that the first term is simply 1, because  $z^{-0} = 1$ .

In its simplest form, the  $z^{-1}$  operator associated with the Z transform can be thought of as a delay operator. So if

$$y(n) = x(n - 1),$$

then the Z transform of this equation would be written

$$Y(z) = z^{-1}X(z).$$

And if

$$w(n) = y(n - 1),$$

then the Z transform of  $w(n)$  is

$$W(z) = z^{-1}Y(z) = z^{-1}(z^{-1}X(z)) = z^{-2}X(z). \quad (\text{A.4})$$

If  $W(z)$  is defined as above, and  $X(z)$  as defined in Eq. (A.3), then

$$W(z) = z^{-2}1 + .5z^{-3} + .25z^{-4},$$

which, in turn, going back to the sampled time domain, means that

$$w(n) = \delta(t - 2 \cdot \Delta t) + .5 \cdot \delta(t - 3 \cdot \Delta t) + .25 \cdot \delta(t - 4 \cdot \Delta t).$$

We can add two Z transforms by making sure like powers of  $z^{-1}$  are lumped together. For instance, adding  $X(z)$  of Eq. (A.3) and  $W(z)$  of Eq. (A.4) gives

$$\begin{aligned} X(z) + W(z) &= [1 + .5z^{-1} + .25z^{-2}] + [z^{-2} + .5z^{-3} + .25z^{-4}] \\ &= 1 + .5z^{-1} + 1.25z^{-2} + .5z^{-3} + .25z^{-4}. \end{aligned}$$

Going back to the sampled time domain,

$$x(n) + w(n) = \delta(t) + .5 \cdot \delta(t - \Delta t) + 1.25 \cdot \delta(t - 2 \cdot \Delta t) + .5 \cdot \delta(t - 3 \cdot \Delta t) + .25 \cdot \delta(t - 4 \cdot \Delta t),$$

which could have been obtained by adding  $x(n)$  and  $w(n)$ , being sure to keep like delta ( $\delta$ ) terms together.

## A.2 CONVOLUTION USING THE Z TRANSFORM

Figure A.1 illustrates a simple linear system. In this example, suppose  $h(n)$  is a system that adds the present value of  $x(n)$  to its previous value  $x(n - 1)$  and outputs it as the new value of  $y(n)$ :

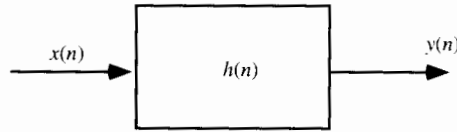
$$y(n) = x(n) + x(n - 1). \quad (\text{A.5})$$

This function  $h(n)$  is referred to as the “impulse response” for the following reason: if the input  $x(n)$  is an impulse  $\delta(n)$  then the output is  $h(n)$ . (See Ziemer, et al. [1], or Phillips and Parr [2].) Therefore,

$$h(n) = \delta(n) + \delta(n - \Delta t). \quad (\text{A.6})$$



**Figure A.1** A simple system diagram. The output  $y(n)$  and the input  $x(n)$  are related by the impulse response  $h(n)$ .



A more mathematically concise expression of Eq. (A.5) is

$$y(n) = \sum_{i=0}^1 x(n-i) \cdot h(i). \quad (\text{A.7})$$

Equation A.7 is a discrete convolution. Notice that the index  $i$  ranges only between 0 and 1 because  $h(n)$  in Eq. (A.6) has only these two terms. This could be generalized for any upper limit to infinity.

If, instead of an impulse, we use the values of  $x(n)$  from Eq. (A.2),  $y(n)$  is calculated from Eq. (A.7):

$$y(n) = \delta(t) + 1.5 \cdot \delta(t - \Delta t) + .75 \cdot \delta(t - 2 \cdot \Delta t) + .25 \cdot \delta(t - 3 \cdot \Delta t). \quad (\text{A.8})$$

This process was made tractable only by the small number of terms used in this example.

As an alternative approach, take the Z transforms of  $x(n)$  and  $h(n)$ :

$$X(z) = 1 + .5 \cdot z^{-1} + .25 \cdot z^{-2}$$

$$H(z) = 1 + z^{-1}.$$

Multiplying the two together gives

$$H(z) \cdot X(z) = (1 + .5 \cdot z^{-1} + .25 \cdot z^{-2}) \cdot (1 + z^{-1}) = 1 + 1.5 \cdot z^{-1} + .75 \cdot z^{-2} + .25 \cdot z^{-2}.$$

Going back to the time domain gives the  $y(n)$  of Eq. (A.8). This illustrates the powerful *convolution theorem*: convolution in the discrete time domain becomes multiplication in the Z domain. The  $H(z)$ , which is the Z transform of the impulse response, is referred to as the “transfer function.”

### A.2.1 Proof of the Convolution Theorem

Starting with the definition of convolution in the discrete time domain,

$$y(n) = \sum_{i=0}^{\infty} h(n-i) \cdot x(i), \quad (\text{A.9})$$

we take the Z transform of both sides

$$\sum_{n=0}^{\infty} y(n)z^{-n} = \sum_{n=0}^{\infty} \sum_{i=0}^{\infty} h(n-i) \cdot x(i)z^{-n}$$

and then interchange the summation signs

$$Y(z) = \sum_{i=0}^{\infty} x(i) \sum_{n=0}^{\infty} h(n-i) \cdot z^{-n}.$$

Multiplying by  $z^{-i} \cdot z^i$  gives

$$Y(z) = \sum_{i=0}^{\infty} x(i)z^{-i} \sum_{n=0}^{\infty} h(n-i) \cdot z^{-n+i}$$

and using the parameter  $m = n - i$  results in

$$Y(z) = \sum_{i=0}^{\infty} x(i) \cdot z^{-i} \sum_{m=-i}^{\infty} h(m) \cdot z^{-m}.$$

However, as long as  $h(n)$  is a causal function, it is zero for values of  $m$  less than zero, so we can write

$$Y(z) = \sum_{i=0}^{\infty} x(i) \cdot z^{-i} \sum_{m=0}^{\infty} h(m) \cdot z^{-m},$$

which gives

$$Y(z) = H(z) \cdot X(z). \quad (\text{A.10})$$

Note that Eq. (A.1) is usually referred to as the *bilateral Z Transform* because it is defined for both positive and negative  $n$ . However, we will almost always use causal functions, so the summation will be over the positive  $n$ 's.

### A.2.2 Example: A Low-Pass Filter

Suppose the impulse response in Fig. A.1 is an exponentially decaying function

$$h(n) = Ae^{-n \cdot \Delta t / t_0} \quad n = 0, 1, 2, 3, \dots \quad (\text{A.11})$$

and the input is the discretized unit step function

$$u(n) = 1. \quad n = 0, 1, 2, 3, \dots \quad (\text{A.12})$$

Since

$$\sum_{n=0}^{\infty} a^{-n} = \frac{1}{1 - a^{-1}} \quad \text{when } a \leq 1,$$

$H(z)$  can be calculated

$$Z[h(n)] = H(z) = A \sum_{n=0}^{\infty} [e^{\Delta t / t_0} z]^{-n} = \frac{A}{1 - e^{-\Delta t / t_0} z^{-1}} \quad (\text{A.13})$$

and similarly

$$Z[u(n)] = U(z) = \frac{1}{1 - z^{-1}}.$$

Or we could simply refer to a table of Z transforms, such as Table A.1. The desired convolution is

$$Y(z) = H(z) \cdot U(z) = \frac{A}{1 - e^{-\Delta t / t_0} z^{-1}} \cdot \frac{1}{1 - z^{-1}} = \frac{A}{1 - (1 + e^{-\Delta t / t_0})z^{-1} + e^{-\Delta t / t_0} z^{-2}}. \quad (\text{A.14})$$

To get a solution in the time domain, we take the partial fraction expansion of  $Y(z)$  [3]

$$Y(z) = \frac{A}{1 - e^{-\Delta t / t_0} z^{-1}} \cdot \frac{1}{1 - z^{-1}} = A \cdot \left[ \frac{B}{1 - e^{-\Delta t / t_0} z^{-1}} + \frac{C}{1 - z^{-1}} \right], \quad (\text{A.15})$$

where

$$B = -\frac{e^{-\Delta t / t_0}}{1 - e^{-\Delta t / t_0}} \quad \text{and} \quad C = \frac{1}{1 - e^{-\Delta t / t_0}}.$$

The two terms in Eq. (A.15) can be taken back into the sampled time domain by finding the

TABLE A.1 Transforms among the Time, Frequency, Sampled-Time, and Z Domains

Time Domain	Frequency Domain	Sampled-Time Domain	Z Domain
$\delta(t)$	1	$\delta(n)$	1
$u(t)$	$\frac{1}{j\omega}$	$u(n)$	$\frac{1}{1-z^{-1}}$
$tu(t)$	$\frac{1}{(j\omega)^2}$	$n \cdot u(n)$	$\frac{z^{-1}}{(1-z^{-1})^2}$
$e^{-\alpha t} \cdot u(t)$	$\frac{1}{\alpha + j\omega}$	$e^{-\alpha n \cdot \Delta t} \cdot u(n)$	$\frac{1}{1-z^{-1}e^{-\alpha \Delta t}}$
$e^{-\alpha t} \sin(\beta t) \cdot u(t)$	$\frac{\beta}{(\alpha^2 + \beta^2) + j2\alpha\omega - \omega^2}$	$e^{-\alpha n \cdot \Delta t} \sin(\beta n \cdot \Delta t) \cdot u(n)$	$\frac{e^{-\alpha \Delta t} \cdot \sin(\beta \cdot \Delta t) \cdot z^{-1}}{1 - 2e^{-\alpha \Delta t} \cdot \cos(\beta \cdot \Delta t) \cdot z^{-1} + e^{-2\alpha \Delta t} \cdot z^{-2}}$
$e^{-\alpha t} \cos(\beta t) \cdot u(t)$	$\frac{\alpha + j\omega}{(\alpha^2 + \beta^2) + j2\alpha\omega - \omega^2}$	$e^{-\alpha n \cdot \Delta t} \cos(\beta n \cdot \Delta t) \cdot u(n)$	$\frac{1 - e^{-\alpha \Delta t} \cdot \cos(\beta \cdot \Delta t) \cdot z^{-1}}{1 - 2e^{-\alpha \Delta t} \cdot \cos(\beta \cdot \Delta t) \cdot z^{-1} + e^{-2\alpha \Delta t} \cdot z^{-2}}$

time domain terms corresponding to the two Z domain terms in Table A.1, giving

$$y(n) = \frac{A}{1 - e^{-t/t_0}} [1 - e^{-(n+1) \cdot \Delta t/t_0}] \quad n = 0, 1, 2, 3, \dots \quad (\text{A.16})$$

Equation (A.16) is an analytic solution. An alternative approach exists. Consider Eq. (A.14) as a purely algebraic problem where we are solving for  $Y(z)$ :

$$[1 - (1 + e^{-\Delta t/t_0})z^{-1} - e^{-\Delta t/t_0}z^{-2}] \cdot Y(z) = A$$

$$Y(z) = (1 + e^{-\Delta t/t_0})z^{-1}Y(z) - e^{-\Delta t/t_0}z^{-2}Y(z) + A. \quad (\text{A.17})$$

Remember that the  $z^{-1}$  is an operator that just means a delay of one. Therefore, we can take (A.17) directly back to the sampled time domain

$$y(n) = (1 + e^{-\Delta t/t_0})y(n-1) - e^{-\Delta t/t_0}y(n-2) + A \cdot \delta(n). \quad (\text{A.18})$$

Note the following: the A term of Eq. (A.17) became  $A \cdot \delta(n)$  because a constant in the Z domain is a delta function in the time domain (Table A.1). To convince yourself this is true, substitute  $\delta(n)$  for  $x(n)$  in Eq. (A.1), the definition of the Z transform. Only the  $n = 0$  term survives, i.e., a constant. It is not obvious, but Eq. (A.18) is equivalent to Eq. (A.16). Equation (A.16) is an analytic solution, while Eq. (A.18) is more appropriate when calculating the solution iteratively. Here is a computer code to calculate Eq. (A.17):

```

delta = 1;
for ( n=0; n < nmax; n++) {
    y(n) = (1. + exp(-dt/t0))*y[n-1] - exp(-dt/t0)*y[n-2] +
    A*delta;
    delta = 0.;
}

```

There is yet another approach. In the example above, we specified the input  $x(n)$  as  $u(n)$ . This is the step function, sometimes referred to as the Heavyside function. Suppose  $x(n)$  is left as an unspecified function. Then

$$Y(z) = H(z)X(z) = \frac{A}{1 - e^{-\Delta t/t_0}z^{-1}} \cdot X(z),$$

and following the same process

$$\begin{aligned} Y(z) (1 - e^{-\Delta t/t_0} z^{-1}) &= A \cdot X(z) \\ Y(z) &= e^{-\Delta t/t_0} z^{-1} Y(z) + A \cdot X(z) \\ y(n) &= e^{-t/t_0} y(n-1) + A \cdot x(n). \end{aligned} \quad (\text{A.19})$$

Now the appropriate computer code is given by:

```
x = 1;
for ( n=0; n < nmax; n++) {
    y(n) = exp(-dt/t0)*y[n-1] + A*x;
}
```

The result is identical to that generated in the first code; however, this is the more general form. The function  $x$  is just specified as 1, and assuming that  $n = 0$  corresponds to  $t = 0$ ,  $x$  is the step function. However, we could replace  $x$  with any function and it would be convolved with the exponential. In fact, this is a simple one-pole digital filter.

### A.3 CONVOLUTION OF SAMPLED SIGNALS

In dealing with discrete functions, there are actually two types of problems: (1) the discrete functions are sequences of numbers, or (2) the discrete functions are sampled versions of continuous functions. (Note: Electromagnetic simulation definitely uses the second kind!) The key point separating the two is whether interval between samples is an issue. In the previous section, we treated the first problem; now we will look at the second.

Suppose we had started with continuous functions  $x(t)$ ,  $h(t)$ , and  $y(t)$  instead of  $x(n)$ ,  $h(n)$ , and  $y(n)$ , respectively. The convolution in the time domain is

$$y(t) = \int_0^{\infty} h(\tau) \cdot x(t - \tau) d\tau, \quad (\text{A.20})$$

where it has once again been assumed that the system response  $h(t)$  is causal. Suppose that in order to simulate it on a computer, this problem had to be implemented in the discrete domain. The finite difference approximation of the integral in Eq. (A.20) is

$$\begin{aligned} y(n) &\cong \sum_{i=0}^n h(n-i) \cdot x(i) \cdot \Delta t \\ &= \Delta t \sum_{i=-\infty}^n h(n-i) \cdot x(i), \end{aligned} \quad (\text{A.21})$$

where  $\Delta t$  is the time interval between samples. Taking the Z transform of both sides

$$\sum_{n=0}^{\infty} y(n) z^{-n} = \Delta t \sum_{n=0}^{\infty} \sum_{i=0}^n h(n-i) \cdot x(i) z^{-n},$$

the development becomes identical to the previous section, except we obtain the extra  $\Delta t$ :

$$Y(z) = \Delta t \cdot X(z) \cdot H(z). \quad (\text{A.22})$$

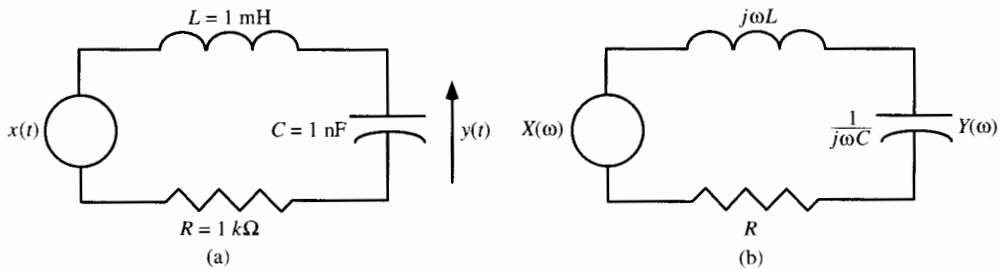
Some other important properties of the Z transform are given in Table A.2.

#### A.3.1 Simulation of a Two-Pole Digital Filter

In this section, we will design a digital filter equivalent to the RLC circuit in Fig. A.2a. It will be convenient to start in the frequency domain (Fig. A.2b), from which we obtain the

**TABLE A.2** Properties of the Z Transforms

	Sampled-Time Domain	Z Domain
Linearity	$f(n)$	$F(z)$
Time shift	$\alpha \cdot f(n) + \beta \cdot g(n)$	$\alpha \cdot F(z) + \beta \cdot F(z)$
Initial value	$f(n - m)$	$z^{-m} F(z)$
Final value	$f(0)$	$\lim_{z \rightarrow 0} F(z)$
	$f(\infty)$	$\lim_{z \rightarrow 1} (z - 1) \cdot F(z)$
Integration	$\sum_{n=0}^{\infty} f(n)$	$\frac{1}{1 - z^{-1}} F(z)$
Convolution	$\sum_{n=0}^{\infty} f(n)h(m - n)$	$F(z) \cdot H(z)$
Complex convolution	$f(n) \cdot g(n)$	$\frac{1}{2\pi j} \oint_C G(v)F\left[\frac{z}{v}\right]v^{-1}\delta v$



**Figure A.2** (a) An RLC circuit. (b) The RLC circuit with component values expressed in the frequency domain.

following transfer function:

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} = \frac{1/j\omega C}{j\omega L + 1/j\omega C + R} = \frac{1/LC}{1/LC + j\omega R/L - \omega^2}. \tag{A.23}$$

To get this into a recognizable form, we will use the following change of parameters:

$$\alpha = \frac{R}{2L} = .5 \times 10^6 \quad \beta = \sqrt{\frac{1}{LC} - \alpha^2} = .866 \times 10^6$$

$$\gamma\beta = \frac{1}{LC} \Rightarrow \gamma = 1.155 \times 10^6.$$

The Z transform can now be read from the frequency domain expression in Table A.1:

$$H(z) = Z \left[ \gamma \frac{\beta}{(\alpha^2 + \beta^2) + j2\alpha\omega - \omega^2} \right]$$

$$= 1.155 \times 10^6 \frac{e^{-\alpha \cdot \Delta t} \sin(\beta \cdot \Delta t) \cdot z^{-1}}{1 - 2e^{-\alpha \cdot \Delta t} \cdot \cos(\beta \cdot \Delta t) \cdot z^{-1} + e^{-2\alpha \cdot \Delta t}}. \tag{A.24}$$

At first, we may be somewhat startled to see the magnitude of the multiplier resulting from the  $\gamma$  term. But remember, when it is convolved with another function, it will be multiplied by  $\Delta t$ ! Since  $\beta = .866 \times 10^6$ , we will want  $\Delta t$  to be much smaller, so choose  $\Delta t = 10^{-7}$ .

Notice now that

$$\begin{aligned} e^{-\alpha \cdot \Delta t} &= e^{-.05} = .951, \\ e^{-2\alpha \cdot \Delta t} &= e^{-.1} = .904, \\ \sin(\beta \cdot \Delta t) &= \sin(.086) = .086, \end{aligned}$$

and

$$\cos(\beta \cdot \Delta t) = \cos(.086) = .9963.$$

Now take the convolution of  $h(n)$  with an unknown function  $x(n)$ :

$$\begin{aligned} Y(z) = H(z)X(z) \cdot \Delta t &= \frac{1.155 \times 10^6 \cdot (0.951) \cdot (0.086) \cdot 10^{-7}}{1 - 2 \cdot (0.951) \cdot (0.9963) \cdot z^{-1} + (.904) z^{-2}} z^{-1} X(z) \\ &= \frac{.0094}{1 - 1.895z^{-1} + .9044z^{-2}} z^{-1} X(z) \end{aligned} \quad (\text{A.25})$$

from which we get

$$\begin{aligned} Y(z) &= 1.895 \cdot z^{-1} Y(z) - .9044z^{-2} Y(z) + .0094 \cdot z^{-1} X(z) \\ y(n) &= 1.895 \cdot y(n-1) - 0.9044 \cdot y(n-2) + 0.0094 \cdot x(n-1). \end{aligned} \quad (\text{A.26})$$

Note that the  $z^{-1}$  in the numerator of  $H(z)$  in Eq. (A.25) resulted in the  $x(n-1)$ , i.e., a delay in the input in Eq. (A.26). The following computer code implements Eq. (A.26):

```
dt = 1e-7;
gamma = 1.155e6;
for { n=0; n< nmax; n++ } {
    y[n] = 2*exp(-alpha*dt)*cos(beta*dt)*y[n-1]
        - exp(-2*alpha*dt)*y[n-2] + dt*gamma*sin(beta*dt)*x[n-1];
}
```

### A.3.2 Sum of Two Parallel Systems

The diagram of a system composed of two branches is given in Fig. A.3. Assume the two transfer functions are

$$H_1(\omega) = \frac{1}{j\omega + \alpha_1} \quad (\text{A.27})$$

and

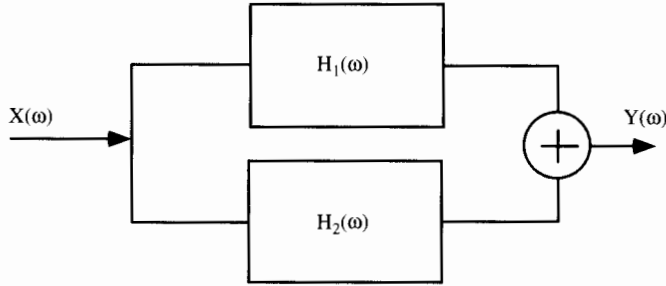
$$H_2(\omega) = \frac{1}{j\omega + \alpha_2}. \quad (\text{A.28})$$

Suppose we want to design a digital simulation of this system. The overall transfer function of the system is given by

$$Y(\omega) = [H_1(\omega) + H_2(\omega)] \cdot X(\omega). \quad (\text{A.29})$$

Going to the Z domain gives

$$\begin{aligned} Y(z) &= [H_1(z) + H_2(z)] \cdot X(z) \cdot \Delta t \\ &= \left[ \frac{1}{1 - e^{-\alpha_1 \cdot \Delta t} z^{-1}} + \frac{1}{1 - e^{-\alpha_2 \cdot \Delta t} z^{-1}} \right] \cdot X(z) \cdot \Delta t \\ &= \left[ \frac{2 - (e^{-\alpha_1 \cdot \Delta t} + e^{-\alpha_2 \cdot \Delta t}) z^{-1}}{1 - (e^{-\alpha_1 \cdot \Delta t} + e^{-\alpha_2 \cdot \Delta t}) z^{-1} + z^{-2}} \right] \cdot X(z) \cdot \Delta t, \end{aligned} \quad (\text{A.30})$$



**Figure A.3** A system diagram consisting of two separate branches.

from which we get

$$Y(z) = (e^{-\alpha_1 \cdot \Delta t} + e^{-\alpha_2 \cdot \Delta t}) z^{-1} Y(z) - z^{-2} Y(z) + 2 \cdot \Delta t \cdot X(z) - \Delta t \cdot (e^{-\alpha_1 \cdot \Delta t} + e^{-\alpha_2 \cdot \Delta t}) z^{-1} X(z). \quad (\text{A.31})$$

Note that two terms of the input are used:  $X(z)$  corresponding to  $x(n)$ , and  $z^{-1} X(z)$  corresponding to  $x(n-1)$ . This does not present any particular difficulty.

Going back to Eq. (A.30), instead of cross-multiplying, suppose we define two auxiliary functions

$$S_1(z) = \frac{\Delta t}{1 - e^{-\alpha_1 \cdot \Delta t} z^{-1}} \cdot X(z)$$

$$S_2(z) = \frac{\Delta t}{1 - e^{-\alpha_2 \cdot \Delta t} z^{-1}} \cdot X(z).$$

So instead of the results of Eq. (A.31), we get

$$S_1(z) = e^{-\alpha_1 \cdot \Delta t} z^{-1} S_1(z) + \Delta t \cdot X(z) \quad (\text{A.32a})$$

$$S_2(z) = e^{-\alpha_2 \cdot \Delta t} z^{-1} S_2(z) + \Delta t \cdot X(z) \quad (\text{A.32b})$$

$$Y(z) = S_1(z) + S_2(z). \quad (\text{A.32c})$$

The results of Eq. (A.32) present a simpler formulation. If, for instance,  $H_1$  and  $H_2$  were each second-order systems, the cross multiplication similar to Eq. (A.30) would produce a fourth-order system. It would be far better to define two second-order auxiliary parameters and make two second-order calculations similar to (A.32a) and (A.32b). The following is the digital simulation of the transfer function:

```

for ( n=0; n < nmax; n++) {
    s1[n] = exp(-alpha1*dt)*s1[n-1] + dt*x[n]
    s2[n] = exp(-alpha2*dt)*s2[n-1] + dt*x[n]
    y[n] = s1[n] + s2[n]
}

```

## A.4 ALTERNATIVE METHODS TO FORMULATE THE Z TRANSFORM

We have seen examples in which problems were stated in the frequency domain and we solved them in the sampled time domain. Our approach has been to take the partial fraction expansion of the frequency domain expression, find the corresponding Z transforms, and solve the problem in the Z domain. Our success depended upon the ability to manipulate the frequency domain expression in a form that could be found in a table like Table A.1. In this section, we present some alternatives.

### A.4.1 Backward Rectangular Approximation

Fourier transform theory tells us that a multiplication by  $j\omega$  in the frequency domain becomes a derivative in the time domain [1, 2]. In going from the time domain to the sampled time domain, the derivative may be approximated by

$$\frac{df(t)}{dt} \cong \frac{f[n \cdot \Delta t] - f[(n-1) \cdot \Delta t]}{\Delta t}$$

Taking the Z transform of the right side gives

$$Z \left\{ \frac{f[nT] - f[(n-1)T]}{\Delta t} \right\} = \frac{F(z) - z^{-1}F(z)}{\Delta t} = \frac{1 - z^{-1}}{\Delta t} F(z).$$

Suppose this is taken one step further and the transition from the frequency domain to the Z domain is made by simply making the replacement

$$j\omega \Rightarrow \frac{1 - z^{-1}}{\Delta t}. \quad (\text{A.33})$$

As an example, the transition from the frequency domain to the Z domain for the one-pole filter becomes

$$\frac{1}{\alpha + j\omega} \Rightarrow \frac{1}{\alpha + \frac{1-z^{-1}}{\Delta t}} = \frac{\Delta t}{\alpha \cdot \Delta t + 1 - z^{-1}}. \quad (\text{A.34})$$

At first glance, this does not seem in any way to represent the Z transform in Table A.1. An approximation that is useful here and elsewhere is

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

Utilizing this approximation Eq. (A.34) becomes

$$\frac{\Delta t}{\alpha \cdot \Delta t + 1 - z^{-1}} = \frac{\Delta t / (1 + \alpha \cdot \Delta t)}{1 - z^{-1} / (1 + \alpha \cdot \Delta t)} \cong \frac{\Delta t \cdot e^{-\alpha \cdot \Delta t}}{1 - e^{-\alpha \cdot \Delta t} z^{-1}}. \quad (\text{A.35})$$

There are two points worth noting. First, the factor  $\Delta t$  that we usually add in the convolution theorem is already there because the substitution of Eq. (A.35) is essentially "Riemann integration," i.e., approximating an integral by a summation at specific intervals of size  $\Delta t$ . Furthermore, the amplitude has been changed by a factor of

$$\frac{1}{1 + \alpha \cdot \Delta t} \cong e^{-\alpha \cdot \Delta t}.$$

Once again, if  $\Delta t$  is small, this term will be very close to 1. This means that more accuracy can be obtained by making  $\Delta t$  smaller.

The practical reasoning for this approach is a little clearer if we go back to the RLC circuit of Fig. A.2, which had the transfer function

$$H(\omega) = \frac{\omega_0^2}{\omega_0^2 + j\omega 2\delta_0\omega_0 - \omega^2},$$

where

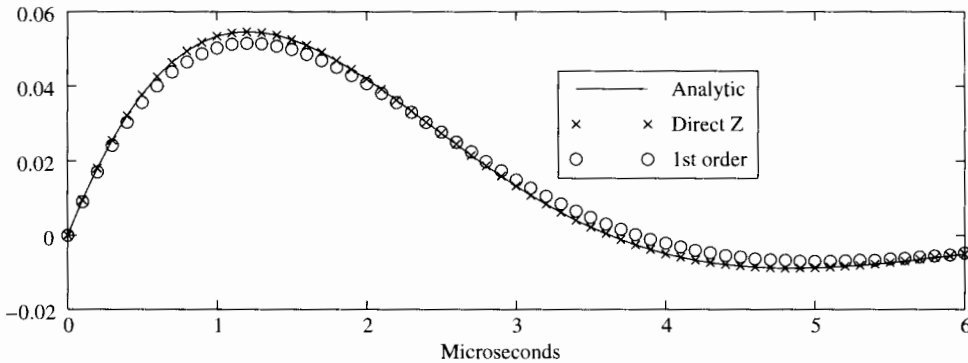
$$\omega_0 = 1/LC \quad \delta_0 = R/2L\omega_0.$$



Instead of having to transform this to the form in Table A.1, replace each  $j\omega$  with  $(1 - z^{-1}/\Delta t)$ :

$$\begin{aligned}
 H(z) &= \frac{\omega_0^2}{\omega_0^2 + \left(\frac{1 - z^{-1}}{\Delta t}\right) 2\delta_0\omega_0 + \left(\frac{1 - z^{-1}}{\Delta t}\right)^2} \\
 &= \frac{\omega_0^2 \Delta t^2}{\omega_0^2 \Delta t^2 + (1 - z^{-1}) 2\delta_0\omega_0 \cdot \Delta t + (1 - 2z^{-1} + z^{-2})} \\
 &= \frac{\omega_0^2 \Delta t^2}{(\omega_0^2 \Delta t^2 + 2\delta_0\omega_0 \cdot \Delta t + 1) - 2(1 + \delta_0\omega_0 \cdot \Delta t) z^{-1} + z^{-2}}.
 \end{aligned} \tag{A.36}$$

The resemblance to the Z transform of Table A.1 is not as obvious, but it is there. Figure A.4 shows the impulse response using Eq. (A.36) compared to the results obtained from Eq. (A.26). These plots were made using a time step of 0.1 microseconds. If the time step is reduced to .01 microseconds, the results correspond almost exactly.



**Figure A.4** Impulse response of the second-order RLC circuit as calculated analytically (—), by the direct Z transform formulation (×), and by the first-order backward rectangular approximation (○).

#### A.4.2 Trapezoidal Approximation (Bilinear Transform)

Equation (A.33) can be improved upon by the following transformation:

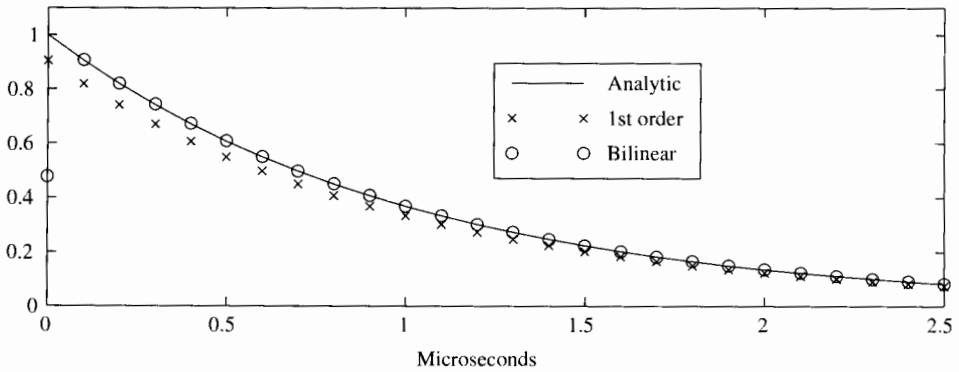
$$j\omega \Rightarrow \frac{2}{\Delta t} \frac{1 - z^{-1}}{1 + z^{-1}}. \tag{A.37}$$

This is referred to as the “bilinear transform.” While the use of Eq. (A.33) can be thought of as approximating the time domain convolution integral with rectangular step Reimann integration, Eq. (A.37) represents trapezoidal integration. Equation (A.37) is preferred by theoreticians in the signal processing field because it is unconditionally stable, whereas (A.33) is not [3]. From a somewhat more intuitive view, it is more accurate because trapezoidal integration is more accurate than rectangular integration. The disadvantage is obvious: the order of the system in the Z domain is doubled!

As a simple example, take the Z transform of the one-pole function using the bilateral transform [4]:

$$\begin{aligned} \frac{1}{\alpha + j\omega} &= \frac{1}{\alpha + \frac{2}{\Delta t} \frac{1 - z^{-1}}{1 + z^{-1}}} = \frac{(1 + z^{-1}) \cdot \Delta t}{(1 + z^{-1}) \cdot \alpha \cdot \Delta t + 2 \cdot (1 - z^{-1})} \\ &= \frac{(1 + z^{-1}) \cdot \Delta t}{(\alpha \cdot \Delta t + 2) + (\alpha \cdot \Delta t - 2) z^{-1}} = \frac{(1 + z^{-1}) \cdot \frac{\Delta t}{2 + \alpha \cdot \Delta t}}{1 - \frac{2 - \alpha \cdot \Delta t}{2 + \alpha \cdot \Delta t} z^{-1}}. \end{aligned} \quad (\text{A.38})$$

Figure A.5 compares the first-order formulation of Eq. (A.35) with the new bilateral formulation from Eq. (A.38) and with the analytic formulation of Eq. (A.11). Clearly Eq. (A.38) is more accurate, after the first pulse. (Notice that  $1 + z^{-1}$  in the numerator of Eq. (A.38) means that the impulse response is calculated by averaging the impulse over the first two time steps.)



**Figure A.5** Impulse response of the single pole filter as calculated analytically (—), by the first-order backward rectangular approximation (x), and by the second-order bilateral approximation (o).

The different approaches used to change from the Fourier domain to the Z domain can be summarized as follows: the direct transform, i.e., converting from terms in the frequency domain to those in the Z domain by looking them up in a table, is the most accurate and also gives the lowest-order expression in the Z domain. The disadvantage of the direct transform is that it often requires a partial fraction expansion and other mathematical manipulations to put it into terms that can be found in a table. For third- and higher-order systems, this is not trivial. Using direct substitution via either the backward approximation or the bilateral transform is usually easier; however, these are approximations. The bilateral transform is better than the backward approximation, at the cost of increased complexity.

Most authors describe these transforms starting from the Laplace domain [3, 4]. The concepts are the same.

## A.5 SUMMARY

The Z transform plays the same role in discrete time that the Laplace and Fourier transforms play in continuous time. As shown earlier in this appendix, it can be used to analyze discrete time equations or develop iterative solutions to discrete time equations. It has even found application in electromagnetic simulation when modeling complicated media [5–8].

**REFERENCES**

- [1] R. E. Ziemer, W. H. Tranter, and D. R. Fannin, *Signals and Systems: Continuous and Discrete*. New York: Macmillan, 1983.
- [2] C. L. Phillips and J. M. Parr, *Signals, Systems, and Transforms*. Englewood Cliffs, NJ: Prentice-Hall, 1995.
- [3] E. P. Cunningham, *Digital Filtering: An Introduction*. Princeton, NJ: Houghton Mifflin, 1992.
- [4] A. V. Oppenheim and R. W. Schaffer, *Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1975.
- [5] D. M. Sullivan, A frequency-dependent FDTD method using Z transforms, *IEEE Trans. on Antennas and Propagat.* vol. AP-40, Oct. 1992, pp. 1223–1230.
- [6] D. M. Sullivan, Z transform theory and the FDTD method, *IEEE Trans. on Antennas and Propagat.* vol. AP-44, Jan. 1996, pp. 28–34.
- [7] D. M. Sullivan, Digital filtering techniques for use with the FDTD method, *Int. J. Numer. Model.*, vol. 12, 1999, pp. 93–106.
- [8] D. M. Sullivan, Nonlinear FDTD formulations using Z tranforms, *IEEE Trans. Microwave Theory and Tech.*, vol. MTT-43, March 1995, pp. 676–682.

# Index

## A

- Absorbing boundary conditions (ABCs), viii
  - one-dimensional simulation, 4–5
  - perfectly matched layer, viii, 51–57, 83–85
- Acoustic FDTD formulation, 133–136
- Antennas
  - aperture antenna, calculating far field of, 118–121
  - microstrip antenna, simulation, 109–113
- Aperture antenna, calculating far field of, 118–121
- Auxiliary differential equation method, 26–27

## B

- Backward rectangular approximation, Z transform, 156–157
- Bilinear transform, 157–158
- Boundary conditions. See also Absorbing boundary conditions
  - microstrip antenna, simulation, 111–112

## C

- Cell size, one-dimensional simulation, 8–9
- Convolution
  - of sampled signals, 152–155
  - using Z transform, 148–152
- Convolution theorem, proof of, 149–150
- Courant condition, 4

## D

- Debye formulation, 23
- Dielectric cylinder, plane wave impinging on, 59–61

- Dielectric medium, propagation in, 5–6
- Dielectric sphere, plane wave impinging on, 85–89

## F

- Far field, of aperture antenna, calculating, 118–121
- Finite-difference time-domain (FDTD) method, vii
  - absorbing boundary conditions, viii, 45, 51–57, 83–85
  - acoustic formulation, 133–136
  - aperture antenna, calculating far field of, 118–121
  - auxiliary differential equation method, 26–27
  - frequency-dependent media, 23–26
  - frequency domain output, 21–23
  - Lorentz medium, 31–34
  - microstrip antenna, simulation, 109–113
  - one-dimensional simulation, 1–18, 19–47
  - perfectly matched layer (PML), viii, 51–57, 83–85
  - of Schroedinger equation, 136–140
  - stability and, 4
  - three-dimensional simulation, 79–108
  - two-dimensional simulation, 49–78
  - unmagnetized plasma, 28–30
- Flux density, one-dimensional simulation, 19–21
- Free space formulation
  - one-dimensional simulation, 1–4
  - three-dimensional simulation, 79–82

Frequency-dependent media, 23–26  
 Frequency domain output, one-dimensional simulation, 21–23

**H**

Heavyside function, 151

**I**

Incident array, 58

**K**

Kinetic energy, 138

**L**

Lorentz formulation, one-dimensional simulation, 31–34

Low-pass filter, 150–152

**M**

Microstrip antenna, simulation of, 109–113

**O**

One-dimensional simulation, 1–18, 19–47  
   absorbing boundary condition, 4–5  
   auxiliary differential equation method, 26–27  
   cell size, 8–9  
   free space formulation, 1–4  
   frequency-dependent media, 23–26  
   frequency domain output, 21–23  
   Lorentz medium, 31–34  
   propagation in dielectric medium, 5–6  
   propagation in a lossy dielectric medium, 8–9  
   reformulation using flux density, 19–21  
   simulating different sources, 7–8  
   unmagnetized plasma, 28–30  
   using Z transforms, 27–30

**P**

Perfectly matched layer (PML), viii  
   three-dimensional simulation, 83–85  
   two-dimensional simulation, 51–57

Plane wave

  on dielectric cylinder, 59–61  
   on dielectric sphere, 85–89

Potential energy, 139

Propagation

  in dielectric medium, 5–6  
   in lossy dielectric medium, 8–9

**S**

Scattered field, 58

Simulation

  FDTD. See Finite-difference time-domain (FDTD) method  
   of Schroedinger equation, 136–140  
   of two-pole digital filter, 152–154

**T**

Three-dimensional simulation, 79–108  
   free space formulation, 79–82  
   perfectly matched layer, 83–85  
   total/scattered field formulation, 85–89

Total field, 58

Total/scattered field formulation

  three-dimensional simulation, 85–89  
   two-dimensional simulation, 58–63

Trapezoidal approximation, Z transform, 157–158

Two-dimensional simulation, 49–78

  perfectly matched layer (PML), 51–57  
   total/scattered field formulation, 58–63

Two-pole digital filter, simulation, 152–154

**U**

Unmagnetized plasma, 28–30

**Y**

Yee cell, 79

**Z**

Z transform, viii–ix, 158

  alternative methods of formulation, 155–158  
   backward rectangular approximation, 156–157

  bilinear transform, 157–158

  convolution using, 148–152

  definition of, 147–148

  one-dimensional simulation, 27–30

  trapezoidal approximation, 157–158

# List of C Programs

FD1D_1.1.c	One-dimensional free space	13
FD1D_1.2.c	Boundary conditions added	15
FD1D_1.3.c	Dielectric medium added	16
FD1D_1.4.c	Sinusoidal source	17
FD1D_1.5.c	Lossy dielectric medium	18
FD1D_2.1.c	Flux density formulation	36
FD1D_2.2.c	Fourier Transform added	39
FD1D_2.3.c	Frequency dependent material	43
FD2D_3.1.c	Two-dimensional free space	64
FD2D_3.2.c	PML added	67
FD2D_3.3.c	Plane wave source	72
FD2D_3.4.c	Scattering from a cylinder	75
FD3D_4.1.c	Three-dimensional free space	90
FD3D_4.2.c	Plane wave and PML	94
FD3D_patch.c	Patch antenna simulation	122
FD3D_far_field.c	Aperture antenna simulation	131
US1.c	One-dimensional acoustic	141
SE.c	One-dimensional Schroedinger	144

## About the Author

**Dennis M. Sullivan** graduated from Marmion Military Academy in Aurora, Illinois, in 1966 with the distinction of being the only cadet private in the graduating class. He spent the next three years in the army, including a year as an artillery forward observer with the 173rd Airborne Brigade in Vietnam. He graduated from the University of Illinois with a B.S. in electrical engineering in 1973, and in 1974 entered the University of Utah as a double major in electrical engineering and ballet. After two years, the Ballet Department encouraged him to vigorously pursue his engineering studies. Over the next ten years, he worked and studied in the Salt Lake City area, at one time supporting his engineering studies by working as a professional actor. In 1987 he finished his doctorate in electrical engineering.

From 1987 to 1993, he was a research engineer with the Department of Radiation Oncology at Stanford Medical School, where he developed a treatment planning system for hyperthermia cancer therapy. Since 1993 he has been on the electrical engineering faculty at the University of Idaho. His main interests are electromagnetic and quantum simulation. In 1997 his paper, “Z Transform Theory and the FDTD Method,” won the R. P. W. King Award from the IEEE Antennas and Propagation Society for the “Best Paper by a Young Investigator” (at age 48).

Mr. Sullivan’s outside interests include street hockey, water polo, and scuba. But above all, he is still trying to get his ballet career going.

# ELECTROMAGNETIC SIMULATION USING THE FDTD METHOD

A volume in the IEEE Press Series on RF and Microwave Technology  
Roger D. Pollard and Richard Booton, *Series Editors*

You can immediately have the power to perform electromagnetic simulation. If you have a fundamental understanding of electromagnetic theory and the knowledge of at least one high-level computer language, you can begin writing simple electromagnetic simulation programs after reading the first chapter of this book.

*Electromagnetic Simulation Using the FDTD Method* describes the power and flexibility of the finite-difference time-domain method as a direct simulation of Maxwell's equations. The FDTD method takes advantage of today's advanced computing power because its computational requirements increase linearly with the size of the simulation problem.

This book begins with a simple one-dimensional simulation, and progresses to a three-dimensional simulation. Each chapter contains a concise explanation of an essential concept and instruction on its implementation into computer code. Projects that increase in complexity are included, ranging from simulations in free space to propagation in dispersive media. Peripheral topics that are pertinent to time-domain simulation, such as Z-transforms and the discrete Fourier transform, are also covered.

*Electromagnetic Simulation Using the FDTD Method* is written for anyone who would like to learn electromagnetic simulation using the finite-difference time-domain method. Appropriate as both a textbook and for self-study, this tutorial-style book will provide all the background you will need to begin research or other practical work in electromagnetic simulation.

## About the Author

**Dennis M. Sullivan**, award-winning author and researcher, has done extensive work in several areas of simulation, including EM dosimetry, hyperthermia cancer treatment, nonlinear optics, and quantum semiconductors. Dr. Sullivan has developed classes in electromagnetic simulation, nonlinear optics, and wavelets. He has taught undergraduate and graduate courses in several areas, such as system theory and electromagnetics. Presently, he is an associate professor of electrical engineering at the University of Idaho, Moscow.

In 1997 Dr. Sullivan won the R. P. W. King Award from the IEEE Antennas and Propagation Society for the "Best Paper by a Young Investigator" for his paper "Z Transform Theory and FDTD Method". He is a senior member of the IEEE (Institute of Electrical and Electronics Engineers).

IEEE Press  
445 Hoes Lane  
P.O. Box 1331  
Piscataway, NJ 08855-1331 U.S.A.  
+1 800 678 IEEE (Toll-free in U.S. and Canada)  
or +1 732 981 0060

Shop at the IEEE store! Visit <http://www.ieee.org/ieeestore>.

IEEE Order No. PC5400

