



ELECTROMAGNETIC AND PHOTONIC SIMULATION FOR THE BEGINNER

Finite-Difference Frequency-Domain in MATLAB®

RAYMOND C. RUMPF

Electromagnetic and Photonic Simulation for the Beginner

Finite-Difference Frequency-Domain in MATLAB®

For a listing of recent titles in the
Artech House Applied Photonics Series,
turn to the back of this book.

Electromagnetic and Photonic Simulation for the Beginner

Finite-Difference Frequency-Domain in MATLAB®

Raymond C. Rumpf



**ARTECH
HOUSE**

BOSTON | LONDON
artechhouse.com

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the U.S. Library of Congress

British Library Cataloguing in Publication Data

A catalog record for this book is available from the British Library.

ISBN-13: 978-1-63081-926-2

Cover design by Charlene Stevens

© 2022 Artech House

685 Canton St.

Norwood, MA 02062

Supplemental material available at: <https://empossible.net/fdfdbook/>

All rights reserved. Printed and bound in the United States of America. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system, without permission in writing from the publisher.

All terms mentioned in this book that are known to be trademarks or service marks have been appropriately capitalized. Artech House cannot attest to the accuracy of this information. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

10 9 8 7 6 5 4 3 2 1

*This book is dedicated to all those who are struggling to get
started in computational electromagnetics*

Contents

Foreword	xiii
Preface	xv
Acknowledgments	xvii
Introduction	xix
References	xxii

CHAPTER 1

MATLAB Preliminaries	1
1.1 Basic Structure of an FDFD Program in MATLAB	1
1.1.1 MATLAB Code for Ideal Structure of a Program	2
1.2 MATLAB and Linear Algebra	3
1.2.1 Special Matrices	6
1.2.2 Matrix Algebra	8
1.3 Setting Up a Grid in MATLAB	8
1.3.1 MATLAB Array Indexing	8
1.3.2 Parameters Describing a Grid in MATLAB	10
1.3.3 Calculating the Grid Parameters	11
1.4 Building Geometries onto Grids	15
1.4.1 Adding Rectangles to a Grid	16
1.4.2 The Centering Algorithm	17
1.4.3 The Meshgrid	19
1.4.4 Adding Circles and Ellipses to a Grid	20
1.4.5 Grid Rotation	22
1.4.6 Boolean Operations	23
1.5 Three-Dimensional Grids	25
1.6 Visualization Techniques	27
1.6.1 Visualizing Data on Grids	27
1.6.2 Visualizing Three-Dimensional Data	29
1.6.3 Visualizing Complex Data	31
1.6.4 Animating the Fields Calculated by FDFD	32
Reference	32

CHAPTER 2

Electromagnetic Preliminaries	33
2.1 Maxwell's Equations	33
2.2 The Constitutive Parameters	37

2.2.1	Anisotropy, Tensors, and Rotation Matrices	37
2.2.2	Rotation Matrices and Tensor Rotation	39
2.3	Expansion of Maxwell's Curl Equations in Cartesian Coordinates	42
2.4	The Electromagnetic Wave Equation	43
2.5	Electromagnetic Waves in LHI Media	45
2.5.1	Wave Polarization	46
2.6	The Dispersion Relation for LHI Media	49
2.7	Scattering at an Interface	49
2.7.1	Reflectance and Transmittance	52
2.8	What is a Two-Dimensional Simulation?	55
2.9	Diffraction from Gratings	56
2.9.1	The Grating Equation	57
2.9.2	Diffraction Efficiency	59
2.9.3	Generalization to Crossed Gratings	60
2.10	Waveguides and Transmission Lines	62
2.10.1	Waveguide Modes and Parameters	63
2.10.2	Transmission Line Parameters	66
2.11	Scalability of Maxwell's Equations	68
2.12	Numerical Solution to Maxwell's Equations	69
	References	70

CHAPTER 3

	The Finite-Difference Method	71
3.1	Introduction	71
3.2	Finite-Difference Approximations	72
3.2.1	Deriving Expressions for Finite-Difference Approximations	73
3.2.2	Example #1—Interpolations and Derivatives from Three Points	76
3.2.3	Example #2—Interpolations and Derivatives from Two Points	78
3.2.4	Example #3—Interpolations and Derivatives from Four Points	79
3.3	Numerical Differentiation	80
3.4	Numerical Boundary Conditions	81
3.4.1	Dirichlet Boundary Conditions	81
3.4.2	Periodic Boundary Conditions	82
3.5	Derivative Matrices	82
3.6	Finite-Difference Approximation of Differential Equations	85
3.7	Solving Matrix Differential Equations	87
3.7.1	Example—Solving a Single-Variable Differential Equation	87
3.8	Multiple Variables and Staggered Grids	89
3.8.1	Example—Solving a Multivariable Problem	92
	References	94

CHAPTER 4

	Finite-Difference Approximation of Maxwell's Equations	95
4.1	Introduction to the Yee Grid Scheme	95
4.2	Preparing Maxwell's Equations for FDFD Analysis	97
4.3	Finite-Difference Approximation of Maxwell's Curl Equations	99

4.4	Finite-Difference Equations for Two-Dimensional FDFD	103
4.4.1	Derivation of E Mode Equations When Frequency Is Not Known	105
4.4.2	Derivation of H Mode Equations When Frequency Is Not Known	105
4.4.3	Derivation of E Mode Equations When Frequency Is Known	106
4.4.4	Derivation of H Mode Equations When Frequency Is Known	106
4.5	Derivative Matrices for Two-Dimensional FDFD	107
4.5.1	Derivative Matrices Incorporating Dirichlet Boundary Conditions	108
4.5.2	Periodic Boundary Conditions	112
4.5.3	Derivative Matrices Incorporating Periodic Boundary Conditions	115
4.5.4	Relationship Between the Derivative Matrices	119
4.6	Derivative Matrices for Three-Dimensional FDFD	120
4.6.1	Relationship Between the Derivative Matrices	123
4.7	Programming the <code>yeder2d()</code> Function in MATLAB	124
4.7.1	Using the <code>yeder2d()</code> Function	126
4.8	Programming the <code>yeder3d()</code> Function in MATLAB	128
4.8.1	Using the <code>yeder3d()</code> Function	129
4.9	The $2\times$ Grid Technique	131
4.10	Numerical Dispersion	134
	References	139

CHAPTER 5

	The Perfectly Matched Layer Absorbing Boundary	141
5.1	The Absorbing Boundary	141
5.2	Derivation of the UPML Absorbing Boundary	143
5.3	Incorporating the UPML into Maxwell's Equations	146
5.4	Calculating the UPML Parameters	147
5.5	Implementation of the UPML in MATLAB	149
5.5.1	Using the <code>addupml2d()</code> Function	150
5.6	The SCPML Absorbing Boundary	153
5.6.1	MATLAB Implementation of <code>calcpml3d()</code>	155
5.6.2	Using the <code>calcpml3d()</code> Function	155
	References	159

CHAPTER 6

	FDFD for Calculating Guided Modes	161
6.1	Formulation for Rigorous Hybrid Mode Calculation	161
6.2	Formulation for Rigorous Slab Waveguide Mode Calculation	166
6.2.1	Formulation of E Mode Slab Waveguide Analysis	167
6.2.2	Formulation of H Mode Slab Waveguide Analysis	168
6.2.3	Formulations for Slab Waveguides in Other Orientations	168
6.2.4	The Effective Index Method	169

6.3	Implementation of Waveguide Mode Calculations	171
6.3.1	MATLAB Implementation of Rib Waveguide Analysis	172
6.3.2	MATLAB Implementation of Slab Waveguide Analysis	179
6.3.3	Animating the Slab Waveguide Mode	185
6.3.4	Convergence	187
6.3.5	MATLAB Implementation for Calculating SPPs	188
6.4	Implementation of Transmission Line Analysis	192
	References	197

CHAPTER 7

	FDFD for Calculating Photonic Bands	199
7.1	Photonic Bands for Rectangular Lattices	199
7.2	Formulation for Rectangular Lattices	201
7.3	Implementation of Photonic Band Calculation	203
7.3.1	Description of MATLAB Code for Calculating Photonic Band Diagrams	205
7.3.2	Description of MATLAB Code for Calculating IFCs	210
	References	214

CHAPTER 8

	FDFD for Scattering Analysis	215
8.1	Formulation of FDFD for Scattering Analysis	215
8.1.1	Matrix Wave Equations for Two-Dimensional Analysis	215
8.2	Incorporating Sources	217
8.2.1	Derivation of the QAAQ Equation	218
8.2.2	Calculating the Source Field $f_{\text{src}}(x,y)$	220
8.2.3	Calculating the SF Masking Matrix Q	222
8.2.4	Compensating for Numerical Dispersion	224
8.3	Calculating Reflection and Transmission for Periodic Structures	226
8.4	Implementation of the FDFD Method for Scattering Analysis	228
8.4.1	Standard Sequence of Simulations for a Newly Written FDFD Code	231
8.4.2	FDFD Analysis of a Sawtooth Diffraction Grating	233
8.4.3	FDFD Analysis of a Self-Collimating Photonic Crystal	239
8.4.4	FDFD Analysis of an OIC Directional Coupler	246
	References	252

CHAPTER 9

	Parameter Sweeps with FDFD	255
9.1	Introduction to Parameter Sweeps	255
9.2	Modifying FDFD for Parameter Sweeps	257
9.2.1	Generic MATLAB Function to Simulate Periodic Structures	258
9.2.2	Main MATLAB Program to Simulate the GMRF	260
9.2.3	Main MATLAB Programs to Analyze a Metal Polarizer	262
9.3	Identifying Common Problems in FDFD	266
	References	267

CHAPTER 10

FDFD Analysis of Three-Dimensional and Anisotropic Devices	269
10.1 Formulation of Three-Dimensional FDFD	269
10.1.1 Finite-Difference Approximation of Maxwell's Curl Equations	270
10.1.2 Maxwell's Equations in Matrix Form	273
10.1.3 Interpolation Matrices	274
10.1.4 Three-Dimensional Matrix Wave Equation	275
10.2 Incorporating Sources into Three-Dimensional FDFD	277
10.3 Iterative Solution for FDFD	278
10.4 Calculating Reflection and Transmission for Doubly Periodic Structures	280
10.5 Implementation of Three-Dimensional FDFD and Examples	282
10.5.1 Standard Sequence of Simulations for a Newly Written Three-Dimensional FDFD Code	282
10.5.2 Generic Three-Dimensional FDFD Function to Simulate Periodic Structures	285
10.5.3 Simulation of a Crossed-Grating GMRF	287
10.5.4 Simulation of a Frequency Selective Surface	290
10.5.5 Parameter Retrieval for a Left-Handed Metamaterial	294
10.5.6 Simulation of an Invisibility Cloak	300
References	303

APPENDIX A

A.1 Best Practices for Building Devices onto Yee Grids	305
A.2 Method Summaries	308
List of Acronyms and Abbreviations	313
About the Author	315
Index	317

Foreword

For anyone wanting to investigate electromagnetic or optical phenomena, or go further to design components and devices, this is an ideal book. In most cases, analytical solutions to hard problems do not exist, and it is necessary to find solutions by numerical means. The author, Dr. Rumpf, is very well-known in the field of computational electromagnetics (CEM). He has pioneered and promoted the advantages of using the finite-difference frequency-domain (FDFD) method, which, as he demonstrates here, is both simple and versatile. While it is somewhat of a brute-force method and may not be as fast and efficient as other techniques, it can be very broadly applied to almost all important electromagnetic problems. Many examples ranging from waveguides, metasurfaces, and guided mode filters are presented, illustrating the power and versatility of FDFD.

The book is organized and written in a style that lends itself to the self-navigator. The level is appropriate for undergraduates, graduates, or professionals in the field. It can also be used by instructors to teach from. Apart from an assumption of some basic knowledge of electromagnetics and an appreciation for a tool such as MATLAB®, this book unfolds as an adventure through a planned series of electromagnetic modeling examples which the reader can try and then build on. The reader's confidence is built not only by completing the assigned examples, but through the conceptual help and insights provided. The content flow of the book reinforces the reader's understanding of foundational physical concepts and the key relationships that are embedded in Maxwell's equations. FDFD is an excellent method for CEM simulations and also naturally lends itself to visualizing near-field and far-field effects. Visualization is the key to building physical intuition and tackling more challenging simulations, but confidence in the results of simulations is of key importance. Dr. Rumpf emphasizes throughout his book the need for "benchmarking." Devices and structures that have known characteristics and electromagnetic responses are simulated first to ensure the code is trustworthy. As Dr. Rumpf notes, the ability to calculate and visualize fields and waves as they interact with material structures gives one the sense of having superpowers. CEM can become almost addictive after a few early successes, and it is only natural to want to build on new insights and investigate increasingly interesting and more complex problems! The importance of benchmarking and tracking convergence of solutions is critical to ensure accuracy, and this is very well covered throughout the book.

Dr. Rumpf has been widely praised for his excellent teaching and writing skills. In the field of CEM, there are few others with his depth of understanding and clarity of purpose. This shines through in his writing style. He systematically introduces topics in an engaging, non-intimidating, and thoroughly informative way. The book

begins with a review of the relevant MATLAB materials including linear algebra as the tool to keep track of everything being simulated. The second chapter reviews electromagnetism and Maxwell's equations. This is an excellent chapter providing all of the necessary backgrounds for success, with tutorials on gratings, waveguides, and transmission lines. He also shows how the guiding principles of Maxwell's equations scale with physical size and material properties, a critical concept to grasp and understand in CEM. The next two chapters, Chapters 3 and 4, provide the basic understanding of the implementation issues for successful CEM. These are the fundamentals of finite-difference approaches for computation, the need for boundary conditions and the organization of the (Yee) grid in an FDFD simulation, essential to capture the physics of the problem at hand, and to obtain quantitative accuracy estimates for simulation results. Chapter 5 explains the inevitable constraints of finite computational modeling and the need for perfectly matched boundaries to ensure the solution volume is not corrupted by computational artifacts from outgoing waves. Chapter 6 explicitly considers guided modes, slab waveguide structures, and includes a section on the simulation of surface plasma waves and surface plasmon polaritons. Chapters 7 and 8 explain very clearly and elegantly the role of, and the computation of, so-called photonic bands and equifrequency surfaces for structured materials, both important for characterizing and visualizing electromagnetic propagation through, as well as reflection or transmission from, periodic structures. The benefits of sweeping variables and parameters in CEM cannot be overstated, and this is covered in Chapter 9. Parameter sweeps can provide deep physical insights into the performance of devices and trends for their optimization. The final chapter includes a number of state-of-the-art and highly stimulating examples involving modern-day metamaterials and design strategies based on transformation optics, such as how to simulate an invisibility cloak.

In summary, this book will be of value to anyone interested in CEM modeling. While alternative simulation approaches exist, for example, for very large problems, FDFD can be comprehensively applied to model almost all electromagnetic phenomena. For example, transient and time-evolving phenomena can be studied by taking a frequency sweep followed by Fourier transformation. Also, being a frequency-domain method, dispersion studies are very straightforward, and physical phenomena such as skin depth and plasmonic effects are automatically incorporated. The many MATLAB codes associated with simulations presented in the book are available for download, granting those superpowers immediately!

Michael Fiddy

*Professor, Optical Science and Engineering and of Electrical Engineering
at the University of North Carolina at Charlotte (UNCC)*

*Research physicist, Army Research Laboratory in
the Directed Energy Bioeffects Group*

January 2022

Preface

Over my career, I have implemented more numerical methods for electromagnetics than most people can even name. Of all these, finite-difference frequency-domain (FDFD) has contributed the most to my career. When I wrote my first FDFD code around the year 2000, I mistakenly thought I had invented a new computational electromagnetic method. Shortly after, I became aware of some research that implemented the method, but the papers describing the research said little about how the method worked. Due to a lack of learning materials, I had to figure out how to implement FDFD completely on my own. I made every possible mistake there is to make while doing so.

For many different methods in *computational electromagnetics* (CEM), I became frustrated with the overall lack of helpful information. It is still very common for me to read a journal article with dozens of equations and almost no description of how to actually implement the method. Eventually, I learned there is a huge difference between deriving the necessary equations and describing how the method is implemented. The latter is seriously lacking in the literature. With insane tenacity and quickly developing visualization skills, I managed to implement a large number of different methods. My frustrating experience getting started in CEM has profoundly impacted how I teach the topic. I have created lots of online content and even some online courses (<https://empossible.net/>) covering electromagnetics and computation. I hope the style I developed for teaching CEM comes through in this book. My goal for this book is to make it easier for others to get started in CEM and to popularize FDFD.

Over the last 20 years, more materials have become available to help a beginner get started in CEM. Unfortunately, there are still almost no materials for learning FDFD and the method remains relatively obscure. I believe FDFD should be one of the most popular methods due to its simplicity and versatility. I cannot think of a better first method to learn. With a good understanding of FDFD, it will be easier to learn new methods in CEM. Every sentence in this book comes from my heart and is intended to help you. I hope this book will help many people get started in CEM that will go on to use the skills and knowledge to discover new things and achieve new breakthroughs that help humanity.

Acknowledgments

Foremost, I gratefully and lovingly acknowledge my wife Krissy for tolerating me being locked in my office for countless hours instead of being with her and helping around the house. I acknowledge my daughter Sydney for keeping me in good physical shape from the karate classes we take together. I acknowledge my son Charlie for making me practice my graphics skills with him using Blender and other tools. I affectionately acknowledge my dog Ardie, who forced me to take breaks by using his nose to push my hands off of my keyboard so I could take him on our daily walks on our mountain. I appreciatively acknowledge my uncle, Dr. Ron Daggett, who supported and encouraged me when I was first getting started in computer programming. Our trips to the Philadelphia Area Computer Society are great memories. I humbly acknowledge my PhD Advisor Dr. Eric Johnson. I learned much from him about building a lab and applying computational electromagnetics to design devices. I will always be his student. I acknowledge Edgar Bustamante, who was my student but still taught me many things about computational electromagnetics, some of which found its way into this book. I appreciatively acknowledge Dr. Ubaldo Robles and Gilbert Carranza who kept the EM Lab running during the time I was mentally focused on writing this book. They did an incredible job covering for me! I must acknowledge Cesar Valle and Sarah Manzano who reviewed early drafts of the book and contributed their thoughts and expertise. This book is better because of them. Last, I wish to acknowledge my boss Dr. Miguel Velez-Reyes who released me from some administrative duties while I was writing this book and likely took those duties on himself. This book could not have happened without the contributions from all of these individuals and many others.

Introduction

Computational electromagnetics (CEM) deals with using a computer to figure out how electric and magnetic fields will behave when interacting with materials and devices. It is especially useful when it is not possible to obtain analytical solutions. I have found skills in CEM to be like having superpowers. The ability to analyze devices and predict how they will behave is an incredible skill to have and will surely accelerate your career. This is the book I wish I had when I was trying to get started in CEM.

The FDFD method may be the easiest numerical method to derive and implement that is able to obtain a rigorous solution to Maxwell's equations. It uses the finite-difference method to solve Maxwell's equations so the underlying mathematics is mature and well-understood. It is the frequency-domain brother of the hugely popular finite-difference time-domain (FDTD) method [1, 2]. For this reason, much of the literature on FDTD can be directly applied to FDFD. FDFD offers many significant advantages over FDTD. FDFD is easier to implement than FDTD, especially when a perfectly matched layer (PML) is incorporated to absorb outgoing waves. For small simulations, FDFD is faster than FDTD. Being a frequency-domain method, FDFD is superior for simulating highly resonant devices or devices exhibiting abrupt features in the spectral response. FDFD can also incorporate oblique angle of incidence into simulations of periodic structures much more easily than FDTD. The benefits of FDFD, however, come at a price. It is a very brute-force method so it is not as fast and efficient as other techniques like the finite element method (FEM) [3] or rigorous coupled-wave analysis [4]. FDFD is an excellent method for visualizing electromagnetic fields and learning about the physics of devices. Other numerical methods often require modification or computationally intensive post-processing steps to visualize the fields. FDFD inherently calculates the fields so they can always be visualized. Visualizing the fields is good for troubleshooting codes, learning about devices, and for showcasing the results of a simulation. Why just plot reflection versus frequency when you can also animate the electromagnetic fields interacting with your device?

FDFD can be compared to the FEM [3, 5]. While the FEM offers better efficiency due to the elegant use of an unstructured grid, it is much more difficult to formulate and implement. To benefit from an unstructured grid, a FEM program must also generate a mesh specific to the device being simulated. Generating meshes adds another large step to the simulation process and is just as complicated of a topic as the FEM itself [6–8]. There is no meshing step for uniform grids in FDFD, greatly simplifying the implementation. The main price paid for the simplicity of FDFD is poorer efficiency compared to other methods. Despite this, FDFD is easily

able to simulate an incredible variety of devices on modern computers and is one of the most versatile methods available.

Working through all of my struggles in CEM allowed me to come up with five key steps for the ideal CEM process. These steps are laid out in the block diagram in Figure i.1 where the first two steps are performed on paper and the remaining steps are performed on a computer. The first step is the *formulation* where all of the necessary equations are derived. The ideal formulation starts at Maxwell's equations and ends with the final equations that should be entered into a computer code to implement the method. The second step is the *implementation* where the algorithm is constructed step-by-step, often in a block diagram. The third step is coding the algorithm in a computer language such as MATLAB, Julia, Python, or C/C++. When the algorithm is running without error, the fourth step is to ensure the method is properly converged before making any conclusions about the results. *Convergence* identifies the proper resolution of the simulation in order to reduce numerical error below an acceptable threshold. The last step is *benchmarking* where devices and structures that have known answers are simulated. This is the only true way to get practice and to conclude that the code and how you are using the code is correct.

One of my big mistakes when I was trying to learn CEM was interpreting the formulation as if that was the implementation. This is mostly due to the literature usually only ever covering the derivation of the equations and not discussing how the algorithms were actually implemented. I could not understand how a code was going to do all of the things happening in the derivations. The answer was that the code does not do these things at all! In fact, I learned that the steps performed on a computer are very different than the steps performed in the formulation. Learning to mentally separate the formulation from the implementation was a big breakthrough for me. Another big mistake was trusting the results of my algorithm the moment it ran without error. Those that simulate the most, trust the simulations the least. I learned that benchmarking was the best way to troubleshoot my algorithms and my own simulation practices. This book and the literature are full of examples to benchmark your own codes. The last big mistake that I will mention is making conclusions about a simulation before the results are converged. The simulated behavior of devices can change dramatically as the resolution of the simulation is increased. Never conclude anything about a device or a simulation until the results are converged.

I am not a detail-oriented person and this caused me to struggle even more in CEM. In fact, I do not possess any special skills for CEM other than passion. For me to be successful, I had to take a different approach that minimized the tediousness of the methods. In the end, I learned to build large and complicated matrices from smaller and simpler matrices. The simpler matrices are easy to build and easy to test. Often, building the big matrices becomes trivial after the simpler matrices are

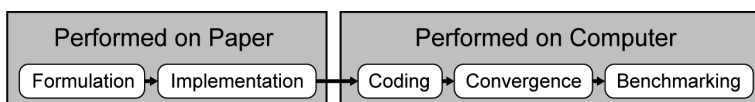


Figure i.1 Block diagram of the ideal CEM process.

calculated. You will observe this practice in the formulations in this book. A person hardly benefits from clean code if the code is only cleaned up after it is made to work. I have learned to write clean code from the very start. It helps to clear my chaotic mind. I found that clean code just seems to run better. Further, I developed and practiced strong visualization skills. Over the years, I have learned that the key to computation is visualization. A person can stare at an array of numbers all day without seeing the trends or errors. Our brains are already programmed to interpret images and catch inconsistencies, so visualizing the data graphically is the most powerful tool for learning and troubleshooting numerical methods. Producing exciting graphics is fun and an excellent way to showcase your simulations!

It is with a tremendous amount of personal debate and guilt that I have omitted some topics from this book. I have left out wonderful subjects like near-field-to-far-field transformation [2, 9], unstructured grids [10, 11], nonorthogonal grids [12], FDFD in different coordinates systems [13], and more. I have left out variants of the FDFD method like the beam propagation method [14], method of lines [15–17], slice absorption method [18–21], and others. There are countless types of devices that I have not demonstrated in this book. I chose the examples in this book to be representative of the broadest array of devices possible. For example, simulating metasurfaces [22, 23] is identical to simulating frequency selective surfaces [24]. So, I only included an example of a frequency selective surface to keep the book as concise as possible. With some thought, a beginner will be able to simulate almost anything with simple modifications to the examples in this book.

The chapters in this book are written in a cumulative order, where later chapters rely on topics covered in previous chapters. Chapters 1 to 5 are background chapters needed to understand how the various forms of the FDFD method are formulated and implemented. Chapter 1 does not teach MATLAB, but it covers special topics and skills in MATLAB that are important for implementing FDFD and understanding why some things in this book are done the way they are. Chapter 2 does not teach electromagnetics, but reviews some key concepts needed to understand and implement the FDFD method. This includes Maxwell's equations, parameters describing the properties of materials, waves, polarization, and some limited device theory. With great effort, I have tried to make the notation consistent between the analytical equations, the numerical equations, and the computer codes as much as possible. Chapter 3 covers the finite-difference method and how it will be used to solve Maxwell's equations. It is quite different than how the finite-difference method is taught in most texts. With practice, the techniques taught in Chapter 3 will let you solve completely new differential equations in mere seconds to minutes. Chapter 4 covers how to make the fields and materials in Maxwell's equations discrete functions using the Yee grid scheme [25]. The Yee grid makes approximating Maxwell's equations with finite-differences simple and accurate. The Yee grid enforces the divergence equations implicitly so only the curl equations have to be solved explicitly. Chapter 5 covers the PML absorbing boundary that will be used to absorb outgoing waves in the simulation. Both a uniaxial PML (UPML) and a stretched-coordinate PML (SCPML) will be presented. Both types of PMLs absorb outgoing waves equally, but the UPML is easier to implement. The SCPML has advantages for large problems when the final matrix equation must be solved iteratively.

Chapters 6 to 10 cover many of the variations of the FDFD method. Chapter 6 covers how to calculate guided modes and analyze transmission lines. Examples include a dielectric rib waveguide, a dielectric slab waveguide, an interface supporting a surface plasmon polariton, and a microstrip transmission line. Chapter 7 describes how to calculate photonic bands of periodic structures using the FDFD method. Examples include calculating and plotting the photonic band diagram of a photonic crystal as well as the isofrequency contours. Chapter 8 describes how to use FDFD to simulate scattering from various types of devices. Simulation examples include calculating the diffraction orders from a sawtooth diffraction grating, simulating the transmission of a Gaussian beam through a self-collimating photonic crystal, and directional coupling of a guided mode through an optical integrated circuit. Chapter 9 explains the best practices for performing parameter sweeps. The reflection and transmission from a guided-mode resonance filter (GMRF) as a function of frequency is simulated and plotted. The extinction ratio of a terahertz polarizer is plotted as a function of grating depth. Last, the frequency response of the terahertz polarizer is plotted.

While this book is primarily intended for the beginner in CEM, Chapter 10 is advanced. FDFD for three-dimensional devices and structures composed of anisotropic materials are covered at the same time. Device examples include a crossed grating GMRF, a frequency selective surface, parameter retrieval of a left-handed metamaterial, and an invisibility cloak designed using transformation optics.

To download the MATLAB codes for this book or to access more learning resources, see <https://empossible.net/fdfdbook/>.

References

- [1] Sullivan, D. M., *Electromagnetic Simulation Using the FDTD Method*, Hoboken, NJ: John Wiley & Sons, 2013.
- [2] Taflove, A., and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Norwood, MA: Artech House, 2005.
- [3] Özgün, Ö., and M. Kuzuoğlu, *MATLAB-Based Finite Element Programming in Electromagnetic Modeling*, New York: CRC Press, 2018.
- [4] Rumpf, R., “Design and Optimization of Nano-Optical Elements by Coupling Fabrication to Optical Behavior,” 2006.
- [5] Jin, J.-M., *The Finite Element Method in Electromagnetics*, Hoboken, NJ: John Wiley & Sons, 2015.
- [6] Ho-Le, K., “Finite Element Mesh Generation Methods: A Review and Classification,” *Computer-Aided Design*, Vol. 20, No. 1, 1988, pp. 27–38.
- [7] Lo, D. S., *Finite Element Mesh Generation*: CRC Press, 2014.
- [8] Mirebeau, J.-M., “Optimal Meshes for Finite Elements of Arbitrary Order,” *Constructive Approximation*, Vol. 32, No. 2, 2010, pp. 339–383.
- [9] Luebbers, R. J., *et al.*, “A Finite-Difference Time-Domain Near Zone to Far Zone Transformation (Electromagnetic Scattering),” *IEEE Trans. on Antennas and Propagation*, Vol. 39, No. 4, 1991, pp. 429–433.
- [10] Fernandez, F., and L. Kulas, “A Simple Finite Difference Approach Using Unstructured Meshes from FEM Mesh Generators,” *15th International Conference on Microwaves, Radar and Wireless Communications*, Warsaw, Poland, May 17–19, 2004, pp. 585–588.

- [11] Gedney, S., F. S. Lansing, and D. L. Rascoe, "Full Wave Analysis of Microwave Monolithic Circuit Devices Using a Generalized Yee-Algorithm Based on an Unstructured Grid," *IEEE Trans. on Microwave Theory and Techniques*, Vol. 44, No. 8, 1996, pp. 1393–1400.
- [12] Lavranos, C., and G. Kyriacou, "Eigenvalue Analysis of Curved Waveguides Employing FDFD Method in Orthogonal Curvilinear Co-Ordinates," *Electronics Letters*, Vol. 42, No. 12, 2006, pp. 702–704.
- [13] Xiao, J., H. Ni, and X. Sun, "Full-Vector Mode Solver for Bending Waveguides Based on the Finite-Difference Frequency-Domain Method in Cylindrical Coordinate Systems," *Optics Letters*, Vol. 33, No. 16, 2008, pp. 1848–1850.
- [14] El-Refaei, H., D. Yevick, and I. Betty, "Stable and Noniterative Bidirectional Beam Propagation Method," *IEEE Photonics Technology Letters*, Vol. 12, No. 4, 2000, pp. 389–391.
- [15] Helfert, S. F., and R. Pregla, "The Method of Lines: A Versatile Tool for the Analysis of Waveguide Structures," *Electromagnetics*, Vol. 22, No. 8, 2002, pp. 615–637.
- [16] Pregla, R., and W. Pascher, "The Method of Lines," *Numerical Techniques for Microwave and Millimeter Wave Passive Structures*, Vol. 1, 1989, pp. 381–446.
- [17] Sadiku, M. N., and C. Obiozor, "A Simple Introduction to the Method of Lines," *International Journal of Electrical Engineering Education*, Vol. 37, No. 3, 2000, pp. 282–296.
- [18] Deng, H., and S. Chen, "Efficient Implementation of Fourier Modal Slice Absorption Method for Lamellar Crossed Gratings," *Optical Engineering*, Vol. 52, No. 6, 2013, p. 068201.
- [19] Deng, H., and S. Chen, "Convergence Improvement of the Fourier Modal Slice Absorption Method for Crossed Gratings," *Optik*, Vol. 126, No. 24, 2015, pp. 5310–5315.
- [20] Deng, H., S.-Q. Chen, and L. Ma, "Calculation of Zero-Order Diffraction by Using the Modal Slice Absorption Method for Gratings," *Journal of University of Electronic Science and Technology of China*, 2015, p. 06.
- [21] Rumpf, R. C., A. Tal, and S. M. Kuebler, "Rigorous Electromagnetic Analysis of Volumetrically Complex Media Using the Slice Absorption Method," *Journal of the Optical Society of America A*, Vol. 24, No. 10, 2007, pp. 3123–3134.
- [22] Simovski, C., "Material Parameters of Metamaterials (A Review)," *Optics and Spectroscopy*, Vol. 107, No. 5, 2009, pp. 726–753.
- [23] Vahabzadeh, Y., *et al.*, "Computational Analysis of Metasurfaces," *IEEE J. on Multiscale and Multiphysics Computational Techniques*, Vol. 3, 2018, pp. 37–49.
- [24] Farahat, N., and R. Mittra, "Analysis of Frequency Selective Surfaces Using the Finite Difference Time Domain (FDTD) Method," *IEEE Antennas and Propagation Society International Symposium*, San Antonio, TX, June 16–21, 2002, pp. 568–571.
- [25] Yee, K., "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Trans. on Antennas and Propagation*, Vol. 14, No. 3, 1966, pp. 302–307.

MATLAB Preliminaries

This chapter covers some basic topics related to MATLAB that are useful for implementing the finite-difference frequency-domain (FDFD) method. It is assumed that the reader has a basic familiarity with MATLAB® and computer programming. If not, visit the MathWorks website for excellent introductory and tutorial materials. If MATLAB is not available to you, Octave is an excellent open-source alternative that offers close to 100% compatibility with the MATLAB language.

1.1 Basic Structure of an FDFD Program in MATLAB

A block diagram of the basic structure for an FDFD program in MATLAB is illustrated in Figure 1.1. The MATLAB code corresponding to the block diagram is provided in Section 1.1.1. It is a common practice to begin a MATLAB program with a commented section that summarizes the purpose of the program, how to use it, and any other information that may be useful to know. This is especially helpful if the code may be used by others or put away and used again years later. When `help` followed by the program name is entered at the command prompt, the commented section at the start of the program will be displayed in the command window. It is useful for the first line of the program to be the name of the file and any other information provided immediately below. The header for the program in Section 1.1.1 extends from lines 1 to 8.

Immediately after the header, MATLAB is initialized on lines 10 to 13. Throughout this book, MATLAB will be initialized in three steps. First, all open figure windows are closed with the command `close all`. Second, all texts are cleared from the command window with the command `clc`. Third, and most important, all variables and functions are cleared from memory with the command `clear all`.

After MATLAB is initialized, the simulation itself is initialized by defining all of the units and constants that will be used by the program. This code extends from lines 15 to 31. Here, `meters` is set to the value 1 and all other length units are

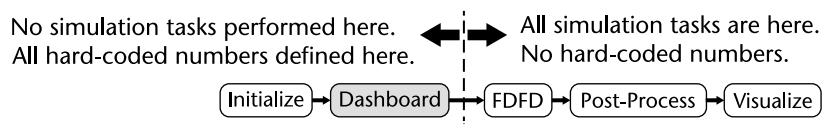


Figure 1.1 Basic structure of an FDFD program in MATLAB.

defined relative to this. This makes it easier to keep the units consistent and allows for much cleaner code that is easy to read and understand. For example, the dimension a can be defined as 2.0 mm with the following line of code.

```
a = 2.0 * millimeters;
```

For photonic simulations, it is best to define `micrometers` to be 1 and then define all other length scales relative to this. For radio frequency simulations, frequency is often used instead of wavelength and so it is convenient to define frequency units in addition to dimensional units. With all units defined, any physical constants that will be needed can be defined immediately afterward. These should be defined in terms of the specified units, as done in lines 28 to 31.

The next part of the MATLAB program is a major new section. Observe the section header inserted into lines 33 to 35 to identify that this is the start of what will be called the dashboard. A nice visual break between sections helps our brains understand the code more easily. The dashboard is where all aspects of the simulation are defined and controlled. All hard-coded numbers in the program should be defined here. No tasks should be performed in the dashboard, such as building arrays, calculating matrices, or generating figures. This practice will keep the dashboard simple and restricted to just defining parameters that control the rest of the code. For FDFD analysis, the dashboard will define all the needed parameters for the source, dimensions of the device, material properties of the device, and numerical parameters controlling things like the size and resolution of the grid.

After the dashboard is the rest of the program where the real work happens. Absolutely no hard-coded numbers should appear outside of the dashboard. If changing a value inside of the dashboard requires you to change something else outside of the dashboard, you have written your program incorrectly. When multiple edits to the program are needed to change one thing, there is a good chance you will miss some of the edits. The absolute best outcome of this is that the simulation will fail. An even worse outcome is that the simulation appears to work but gives you the wrong answer that you think is correct. Also, perhaps the most powerful tools in electromagnetics are parameter sweeps where the result of the simulation is plotted as a parameter is varied over some range of values. For example, plotting reflection and transmission as a function of frequency is one of the most common parameter sweeps. Imagine how many changes your code would require if you had to change multiple items in your code for every frequency in the sweep. If your MATLAB program is written properly, parameter sweeps are as easy as placing a big `for` loop around your code. Procedures and best practices for parameter sweeps are covered in Chapter 9.

1.1.1 MATLAB Code for Ideal Structure of a Program

```
1 % basic_programstructure.m
2 %
3 % The purposes of this program are to:
4 %
5 % 1. Demonstrate the function of a header.
```

```

6 % 2. Demonstrate how to initialize MATLAB.
7 % 3. Demonstrate how to define units.
8 % 4. Demonstrate the dashboard.
9
10 % INITIALIZE MATLAB
11 close all;
12 clc
13 clear all;
14
15 % DEFINE UNITS
16 meters = 1;
17 centimeters = 1e-2 * meters;
18 millimeters = 1e-3 * meters;
19 micrometers = 1e-6 * meters;
20 nanometers = 1e-9 * meters;
21 seconds = 1;
22 hertz = 1/seconds;
23 kilohertz = 1e3 * hertz;
24 megahertz = 1e6 * hertz;
25 gigahertz = 1e9 * hertz;
26
27 % CONSTANTS
28 c0 = 299792458 * meters/seconds;
29 e0 = 8.854187812813e-12 * 1/meters;
30 u0 = 1.256637062121e-6 * 1/meters;
31 N0 = 376.7303136686;
32
33 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
34 %% DASHBOARD
35 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
36
37 % DEFINE ALL HARD CODED NUMBERS HERE
38
39 % NO WORK HAPPENS HERE
40
41 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
42 %% REST OF THE PROGRAM
43 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
44
45 % ALL THE WORK HAPPENS HERE
46
47 % NO HARD-CODED NUMBERS APPEAR HERE

```

1.2 MATLAB and Linear Algebra

The name MATLAB is an acronym for MATrix LABoratory. The original purpose of MATLAB was to provide an easy interface to matrix libraries like LINPACK and EISPACK. A *matrix* is nothing more than a table of numbers that holds information

from large sets of linear algebraic equations. For example, define the following set of four equations.

$$\begin{aligned}
 a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\
 a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\
 a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\
 a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4
 \end{aligned} \tag{1.1}$$

In these equations, the a_{mn} terms are *coefficients* that will have numerical values assigned to them. The terms b_1 , b_2 , b_3 , and b_4 are constants representing the *excitation* and will also have numerical values assigned to them. Last, x_1 , x_2 , x_3 , and x_4 are the *unknowns* that are to be determined by solving the system of equations.

In FDFD, instead of having four unknown terms, there will be hundreds of thousands of unknown terms. It is quite tedious to keep writing all of the equations in (1.1) and to manipulate many equations at every step of a derivation. To get around this, it is much more convenient to express and manipulate large sets of equations as matrices. Using matrices, (1.1) is written compactly as

$$\mathbf{Ax} = \mathbf{b} \tag{1.2}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \tag{1.3}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \tag{1.4}$$

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix} \tag{1.5}$$

All of the terms from the large set of equations have been assembled into the matrix \mathbf{A} and the column vectors \mathbf{x} and \mathbf{b} . In this book, bold uppercase letters represent matrices while bold lowercase letters represent column vectors. This allows all of the data associated with the large set of equations to be manipulated using just the terms \mathbf{A} , \mathbf{x} , and \mathbf{b} using a modified set of rules for matrix algebra [1]. To solve (1.2) for \mathbf{x} , both sides of the equation are predivided by \mathbf{A} to get $\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$. The product $\mathbf{A}^{-1}\mathbf{A}$ equals the identity matrix \mathbf{I} which vanishes from the equation to get

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad (1.6)$$

It is these types of calculations that MATLAB makes incredibly easy to do. For example, the following code enters values for the matrix \mathbf{A} and column vector \mathbf{b} and then solves for \mathbf{x} using (1.6). Note that commas separate elements along a row while semicolons separate elements along the columns. Inserting the commas is optional in MATLAB, but can help make the code more readable.

```
A = [ -0.8 , 1.3 , -1.4 , -1.5 ; ...
      -1.4 , -0.3 , -1.3 , -0.3 ; ...
       0.5 , 0.1 , 0.1 , -1.3 ; ...
       0.5 , 1.0 , 0.5 , -0.2 ];
b = [ -2.66 ; -4.86 ; 1.56 ; 2.60 ];
x = A\b;
```

Observe how the column vector \mathbf{x} is calculated in the above MATLAB code. From a computer coding perspective, it is a mistake to read (1.6) as the inverse of \mathbf{A} times the column vector \mathbf{b} . This interpretation leads to obtaining the solution as $\mathbf{x} = \text{inv}(\mathbf{A}) * \mathbf{b}$. While a correct solution can be obtained this way, there is a significant difference between calculating a matrix inverse and performing a matrix division in MATLAB. Matrix division does not calculate the inverse of matrix \mathbf{A} and usually involves significantly fewer calculations. Calculating \mathbf{x} as $\mathbf{x} = \mathbf{A} \backslash \mathbf{b}$ is called *backward division* and is the preferred way in FDFD to solve the system of equations defined by $\mathbf{Ax} = \mathbf{b}$.

Matrices are tables of numbers arranged by rows and columns, as illustrated in Figure 1.2. MATLAB makes inserting, extracting, and manipulating rows and columns in matrices very easy. For example, if the second row of matrix \mathbf{A} is to be replaced with all numerical values of 7, the MATLAB command to do this is $\mathbf{A}(2, :) = 7$. Similarly, if the third column is to be replaced with all numerical values of 4, the MATLAB command to do this is $\mathbf{A}(:, 3) = 4$.

Solving differential equations by numerical methods very often leads to matrices with numbers placed along the diagonals of a matrix. Figure 1.3 illustrates

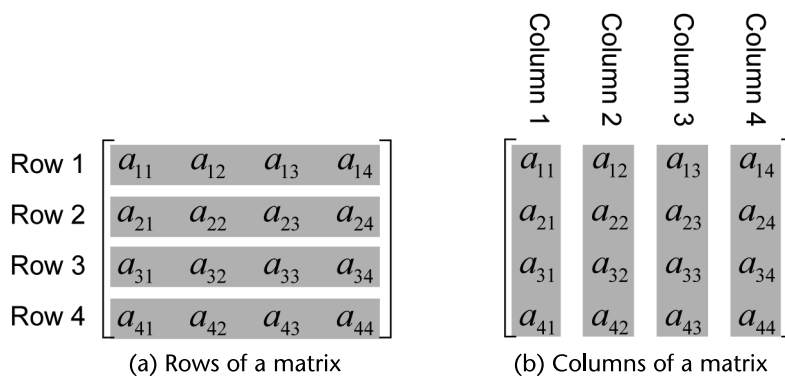


Figure 1.2 Rows and columns of a matrix.

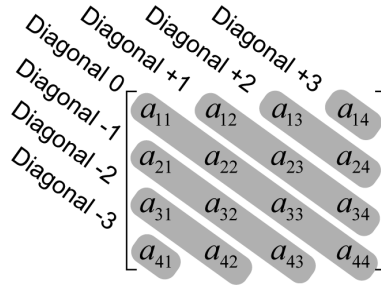


Figure 1.3 Diagonals of a matrix.

how the diagonals are referenced by number. The center diagonal is assigned the index of 0 and is the diagonal being referred to when reading or hearing the label “the diagonal.” Diagonals above the center diagonal are given positive indices and diagonals below the center diagonal are given negative indices. MATLAB makes inserting, extracting, and manipulating diagonals in matrices very easy using the functions `diag()` and `spdiags()`. The diagonals of a matrix will be the primary means for constructing the matrices used in FDFD and will be covered in detail in Chapters 3 and 4.

It is very common in numerical methods to have very large matrices with numbers placed along only a single diagonal or just a handful of diagonals. A matrix with non-zero elements along only the center diagonal is called a *diagonal matrix*. A matrix with non-zero elements along a small set of diagonals is called a *banded matrix*. It is common to have 99.9% or more of the elements in a diagonal or banded matrix be all zeros. Storing every element in these matrices as double-precision floating-point numbers is unnecessary and consumes a lot of memory. Tremendous memory savings can be achieved if only the non-zero elements of a matrix are stored. This is called a *sparse matrix* and a plethora of algorithms exists that are optimized for working with sparse matrices. In MATLAB, a matrix is simply declared as sparse and everything else is handled automatically. Thank you MATLAB! While a matrix can be declared to be sparse using the function `sparse()`, this is bad practice because a full matrix will be created and stored before declaring it as sparse. Instead, it is best to initialize the matrices as sparse from the very beginning so large and full matrices never have to be created or stored.

1.2.1 Special Matrices

FDFD will make use of some special matrices. The zero matrix \mathbf{Z} is a matrix filled entirely with zeros and is the closest thing to the numerical value of zero that exists that is still a matrix. Any matrix multiplied by the zero matrix will give another zero matrix. There also exists a zero column vector $\mathbf{0}$ indicated by a bold zero.

$$\mathbf{Z} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & & \\ \vdots & & \ddots & \\ 0 & & & 0 \end{bmatrix} \quad (1.7)$$

$$\mathbf{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (1.8)$$

A sparse zero matrix can be initialized using `sparse()`. The following line of MATLAB code creates a sparse zeros matrix of size $M \times M$.

```
Z = sparse(M,M);
```

The identity matrix \mathbf{I} is a diagonal matrix with ones placed along the center diagonal and zeros everywhere else. This is the closest thing to the numerical value of 1 that is still a matrix. Any matrix \mathbf{A} multiplied or divided by the identity matrix is the matrix \mathbf{A} again.

$$\mathbf{I} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \quad (1.9)$$

A sparse identity matrix can be initialized using the MATLAB command `speye()`. The following line of MATLAB code creates a sparse identity matrix of size $M \times M$.

```
I = speye(M,M);
```

A *block matrix* is a matrix composed of multiple smaller matrices. Forming large and complicated matrices by assembling combinations of smaller and simpler matrices is a practice that will be used throughout this book to formulate and implement FDFD. MATLAB makes forming block matrices very easy. For example, suppose the following block matrix \mathbf{G} is to be formed.

$$\mathbf{G} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 2 & 2 \\ 1 & 1 & 2 & 2 \\ 3 & 3 & 4 & 4 \\ 3 & 3 & 4 & 4 \end{bmatrix} \quad (1.10)$$

$$\mathbf{A} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} 3 & 3 \\ 3 & 3 \end{bmatrix} \quad \mathbf{D} = \begin{bmatrix} 4 & 4 \\ 4 & 4 \end{bmatrix} \quad (1.11)$$

The following MATLAB code builds the small matrices \mathbf{A} , \mathbf{B} , \mathbf{C} , and \mathbf{D} and then calculates the large matrix \mathbf{G} as a block matrix. This practice will make more sense to perform when matrices are larger and more complicated.

```
A = 1*ones(2,2);
B = 2*ones(2,2);
C = 3*ones(2,2);
D = 4*ones(2,2);
G = [ A B ; C D ];
```

1.2.2 Matrix Algebra

To formulate FDFD, it will be necessary to derive and manipulate matrix equations. It is critical that proper matrix algebra rules [1] are followed so that correct equations are derived and correct quantities are calculated. The most important concept to remember is that the order of multiplication of matrices cannot be reversed. The commutative laws for matrix algebra are

$$\begin{aligned} \mathbf{A} + \mathbf{B} &= \mathbf{B} + \mathbf{A} \\ \mathbf{AB} &\neq \mathbf{BA} \end{aligned} \tag{1.12}$$

The associative laws are

$$\begin{aligned} (\mathbf{A} + \mathbf{B}) + \mathbf{C} &= \mathbf{A} + (\mathbf{B} + \mathbf{C}) \\ (\mathbf{AB})\mathbf{C} &= \mathbf{A}(\mathbf{BC}) \end{aligned} \tag{1.13}$$

The distributive laws are

$$\begin{aligned} (\mathbf{A} + \mathbf{B})\mathbf{C} &= \mathbf{AC} + \mathbf{BC} \\ \mathbf{A}(\mathbf{B} + \mathbf{C}) &= \mathbf{AB} + \mathbf{AC} \end{aligned} \tag{1.14}$$

The commutative laws apply to matrix division as well

$$\mathbf{A}^{-1}\mathbf{B} \neq \mathbf{BA}^{-1} \tag{1.15}$$

The operation $\mathbf{A}^{-1}\mathbf{B}$ is called *predivision* and is calculated in MATLAB as $\mathbf{A} \setminus \mathbf{B}$. The operation \mathbf{BA}^{-1} is called *postdivision* and is calculated in MATLAB as \mathbf{B} / \mathbf{A} . Note the direction of the divide symbol. The backward slash \setminus is used for predivision, or *backward division*, and the forward slash $/$ is used for postdivision, or *forward division*. It is very important to not change the order of the division when dividing matrices. It is also important to not calculate a matrix division by calculating the inverse of a matrix and then multiplying. Explicitly calculating matrix inverses is surprisingly rare in numerical methods.

1.3 Setting Up a Grid in MATLAB

1.3.1 MATLAB Array Indexing

MATLAB treats every variable as if it were a matrix. This leads to some problems when arrays are being manipulated because there are profound differences between matrices and arrays. *Matrices* are sets of numbers cast into a variable that are used in linear algebra equations and algorithms. When visualized, matrices do not provide much insightful information. *Arrays* are also sets of numbers, but they can be composed of any number of dimensions and convey more meaning when they are visualized. The most significant difference between matrices in MATLAB and traditional arrays in other programming languages is how the elements are referenced. This comparison is illustrated in Figure 1.4. From long-standing convention, the elements of a matrix a_{mn} are referenced using the index m for row number and n

for column number, as illustrated in Figure 1.4(a). Both m and n are integers with a starting value of 1. This means the first index m is the vertical position with values increasing downward. The second index n is the horizontal position with values increasing toward the right. The manner in which a traditional array is visualized is shown in Figure 1.4(b). The elements of a two-dimensional array are referenced using two integer indices, i and j , but they are defined and interpreted differently than the indices of a matrix. The first index i starts at 0 and describes the horizontal position with values increasing toward the right. The second index j also starts at 0 and describes the vertical position with values increasing upward.

The most confusing aspect of the inconsistency between matrices and arrays is the direction that the indices access the elements. For matrices, the first index is vertical position and the second is horizontal position. However, people tend to think of functions as $f(x,y)$, where the first argument is the horizontal position and the

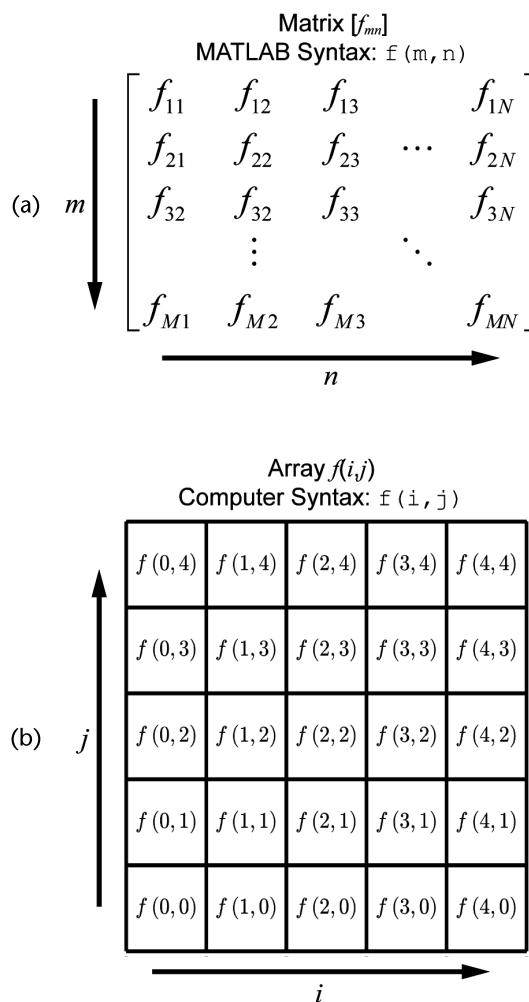


Figure 1.4 Matrix versus array in MATLAB. (a) A matrix and the MATLAB syntax to access elements in the matrix. (b) A traditional array and the representative syntax used by other programming languages to access elements in the array.

second argument is the vertical position. There are multiple ways to work around this inconsistency, but MATLAB will always treat arrays as if they are matrices. In the author's experience in teaching this subject, he has observed students struggle much more with building geometries into arrays than they struggle with numerical algorithms. For this reason, it is recommended to program MATLAB as if the arrays are actually traditional arrays where the first index references horizontal position and the second the vertical position. This practice will make building devices into arrays more intuitive. Treating arrays in this manner will need to be considered when the arrays are visualized because MATLAB will try to visualize those arrays as if they were matrices. Unlike arrays in most other programming languages, in MATLAB the array indices will start at 1 instead of 0 and the vertical array index will have increasing values in the downward direction instead of upward. This second point is actually a nice coincidence that will work in your favor. In electromagnetics, it is preferred to have waves propagate in positive directions. It is also preferred to have waves flow either left-to-right or top-to-bottom. So, having the positive direction be downward is convenient.

1.3.2 Parameters Describing a Grid in MATLAB

Continuous functions that are defined over some area of space contain an infinite amount of information. The standard FDFD algorithm cannot process continuous functions so they must be made discrete. To do this, space is divided into an array of *cells*, as illustrated in Figure 1.5. The set of cells and parameters describing how space is represented will collectively be called *the grid*. The physical size of the grid in the x -direction will be given the variable name S_x in MATLAB. It will have units of length such as meters or micrometers. Similarly, the physical size of the grid in the y -direction will be given the variable name S_y . Space will be divided into an array of discrete cells. In MATLAB, the variable N_x will be the number of cells in the x -direction and N_y will be the number of cells in the y -direction. The size of a single cell in the x -direction will be given the variable name dx and the size of a single cell in the y -direction given the variable name dy . The variables dx and dy will be called the *grid resolution parameters*. If all of the grid parameters are calculated correctly, the following two equations will be satisfied.

$$S_x = N_x * dx \quad (1.16)$$

$$S_y = N_y * dy \quad (1.17)$$

Continuous functions still contain an infinite amount of information within a single cell. To make the function truly discrete, the value of the function will be known at only one infinitely small point in each cell. It is incorrect to think of the function as having a constant value throughout the area of each cell. Instead, it is more correct to think of the function value as varying linearly from point to point through the cells. For now, think of the position of those points to be at the center of the cells, but more will be discussed about this in Chapters 3 and 4 when the Yee grid scheme is introduced. As more points (N_x and N_y) are used on a grid covering the same physical amount of space (S_x and S_y), the function will be resolved more accurately, but the simulation will be less computationally efficient due to requiring

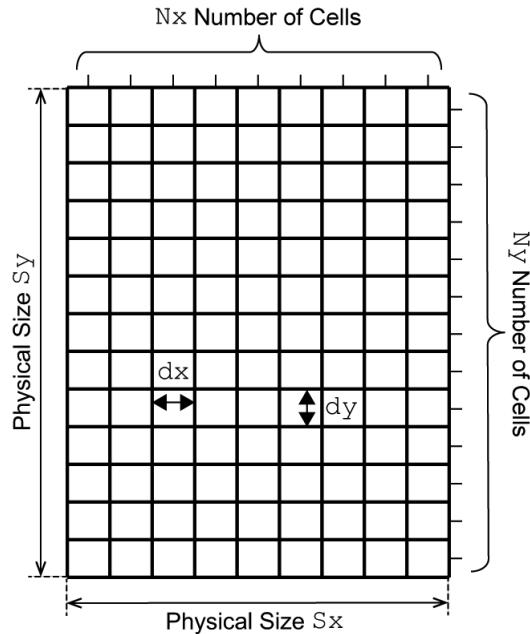


Figure 1.5 Definition of the grid parameters S_x , S_y , N_x , N_y , dx , and dy .

more memory and more calculations. This is a fundamental tradeoff in computation, accuracy versus efficiency. A huge portion of the research in the area of computation is aimed at achieving higher accuracy with greater efficiency.

1.3.3 Calculating the Grid Parameters

There are multiple ways to properly calculate the grid parameters S_x , S_y , N_x , N_y , dx , and dy . It is most common to first calculate the grid resolution parameters dx and dy . It is not necessary that these parameters be equal to each other. The grid resolution is calculated with two considerations in mind. First, the cells on the grid must be sufficiently small to resolve the wave accurately enough. To do this, the variable $NRES$ will be defined in the dashboard as the number of cells per smallest wavelength. $NRES$ typically has a value in the range of 10 to 40. Figure 1.6 shows once cycle of a sine wave resolved with an increasing number of points defined by $NRES$. Observe that somewhere around 10 points is where the wave begins to be resolved well. This is consistent with what you will observe in simulations. However, in real simulations, waves will be interfering and diffracting to produce fluctuations that vary more abruptly than a pure wave. Depending on the physics involved in the simulation, higher values of $NRES$ will be needed in order to obtain accurate results. This leads into the subject of *convergence* where the results from a simulation are analyzed as the value of $NRES$ is increased. A simulation is said to be *converged* when the numerical error in the result falls below an acceptable threshold. It is up to the designer to determine the converged value of $NRES$. Testing for convergence must become standard practice. Behind every simulation should be a proper convergence study. Never make any conclusions about a simulation

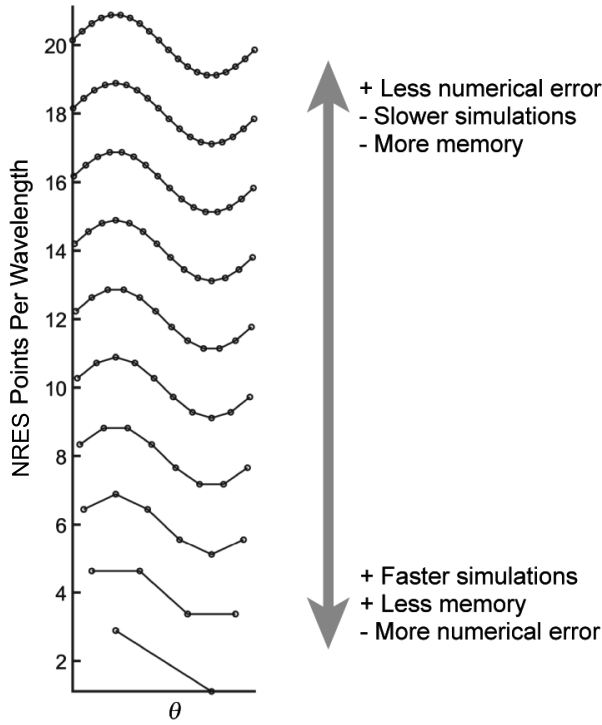


Figure 1.6 Resolving a wave with the increasing number of points.

until it is converged. More about how to perform proper convergence studies will be covered in Chapters 6 to 10.

Another variable that will be defined in the dashboard is the maximum refractive index n_{\max} found anywhere in the simulation. Given the free space wavelength λ_{am0} for the simulation, the smallest wavelength that the simulation will have to resolve is $\lambda_{\text{am0}}/n_{\max}$. It is this smallest wavelength that should be resolved with NRES points. Based on this, the preliminary calculation for dx is

$$dx = \lambda_{\text{am0}}/n_{\max}/\text{NRES} \quad (1.18)$$

The second consideration for grid resolution is that dx and dy must be sufficiently small to resolve the minimum feature size of the device to be simulated. This concept is illustrated in Figure 1.7. It is best to resolve the minimum features with at least one to two grid cells. This is particularly important when simulating small features composed of metals or high permittivity materials. Sometimes it is okay to be lazy and ignore the minimum feature size consideration when simulating devices with low permittivity or when it is known beforehand that the device will not have any small features. For a good preliminary calculation of the grid resolution, go with the smaller value of dx calculated for resolving the minimum wavelength and the minimum feature size.

After the preliminary values of dx and dy are calculated, they can be adjusted, if necessary, so that a critical dimension is resolved on the grid with an exact integer number of cells. For periodic structures, resolving the period exactly is often

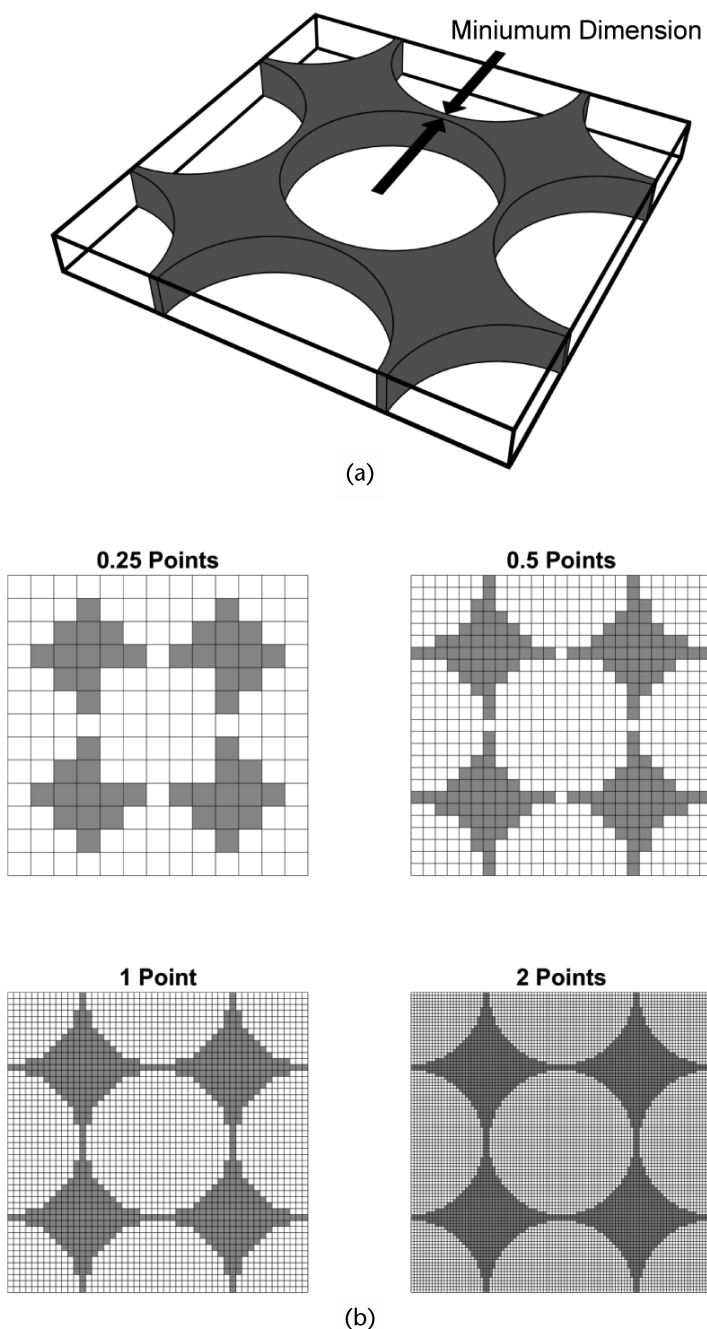


Figure 1.7 (a) Lattice to build into an array. (b) Grids where lattices are resolved with a different number of points representing the minimum dimension.

very important. For diffraction gratings, resolving the groove depth is often very important. It is up to you as the device expert to identify any critical dimensions and the need to resolve them exactly. If a wrong guess is made for the most critical dimension, the simulation may exhibit slow convergence but will still give the correct answer if proper convergence habits are followed.

Suppose it is desired to accurately resolve the dimension a in the x -direction. The equation $n_x = a/dx$ calculates how many grid cells will represent the dimension a given the grid resolution dx . The number n_x is very likely not going to be an integer here because the dimension a was not considered in the calculation of dx . To adjust the value of dx to make n_x an integer, the calculated value of n_x should be rounded up to the nearest integer using MATLAB's `ceil()` function.

$$n_x = \text{ceil}(a/dx) \quad (1.19)$$

Given that n_x has been made an integer, dx is adjusted according to

$$dx = a/n_x \quad (1.20)$$

After this recalculation of dx , the dimension a will be resolved on the grid exactly by n_x cells. This procedure ensures that dx is made smaller instead of bigger so grid resolution is improved instead of lost.

The concepts discussed above to calculate the grid resolution parameter dx are illustrated in Figure 1.8. At the top is a bar representing some critical dimension a that should be resolved exactly with an integer number of grid cells for best numerical efficiency and least error. The preliminary calculation of dx does not achieve this so it is adjusted in a second step. This will be called *snapping* the grid to critical dimensions. Equations (1.18) to (1.20) are repeated to calculate dy . Uniform grids can only snap to one critical dimension per axis so choose wisely which dimension is critical. Make snapping the grid standard practice when setting up a grid for FDFD analysis! This practice will provide your FDFD codes better accuracy and faster convergence.

At this point, only the grid resolution parameters dx and dy are known and they will not be modified again in the program. The next step is to calculate the overall size of the grid. This includes the physical dimensions S_x and S_y and the number of points N_x and N_y along the x - and y -directions, respectively. Usually, S_x and S_y are calculated first. A typical grid setup for an FDFD simulation is illustrated in Figure 1.9. Lengths 1, 5, 6, and 10 represent absorbing boundaries that will be incorporated onto the grid to absorb waves propagating outward so that they do not reflect back into the simulation. These will typically be 10 to 20 cells large. Lengths 3 and 8 are the dimensions of the device to be simulated. Lengths 2, 4, 7, and 9 are spacer regions placed between the device and the absorbing boundaries.

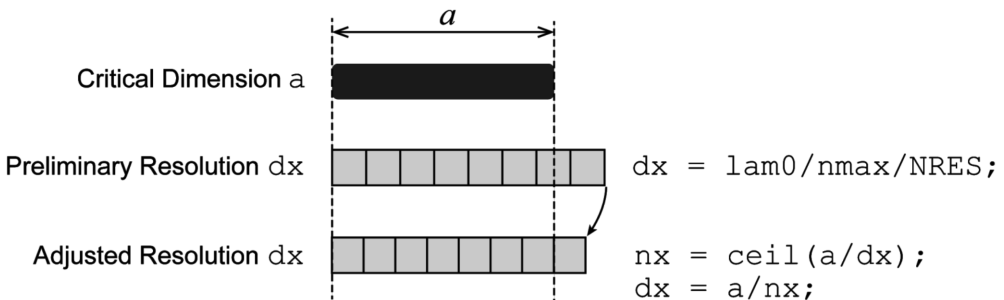


Figure 1.8 Illustration of snapping the grid to a critical dimension.

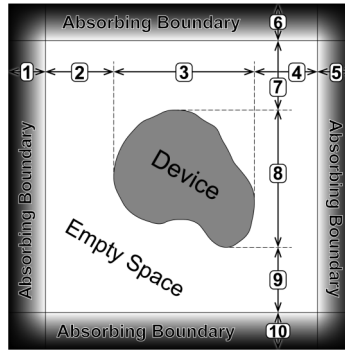


Figure 1.9 Typical grid setup for FDFD.

These are typically made to be at least a quarter to one wavelength large and are needed to obtain accurate results or to better visualize the field around the device. The grid must be made sufficiently large to fit all of this at the same time.

After S_x and S_y are calculated, the total number of points on the grid in the x -direction is calculated as S_x/dx . However, this is likely not going to be an integer quantity because the total size of the grid S_x was not considered in the calculation of dx . For this reason, the quantity S_x/dx is rounded up to the nearest integer to calculate the total number of points N_x .

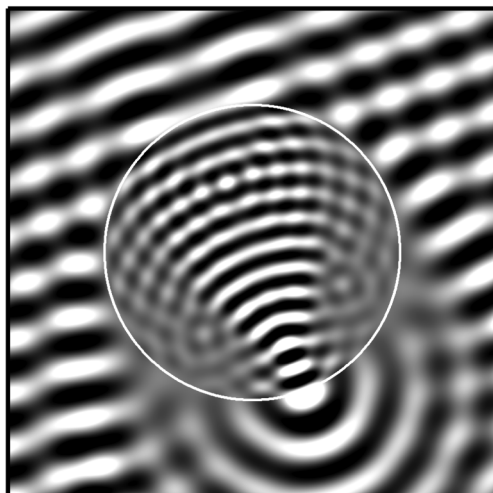
$$N_x = \text{ceil}(S_x/dx) \quad (1.21)$$

Since N_x was rounded up, S_x may no longer accurately represent the physical size of the grid. This is corrected simply by recalculating S_x according to (1.16). This process is repeated for the y -direction to calculate S_y and N_y . Now all of the grid parameters are consistent and the grid has been optimized to simulate the device.

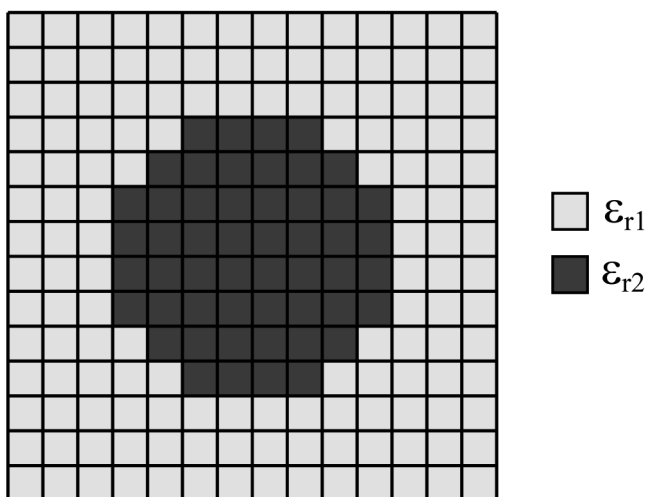
With the calculated grid, it is time to build the arrays ER and UR which contain the relative permittivity and permeability, respectively, across the grid. These are needed in order to fully describe the problem to be simulated. The following sections describe how to make some simple shapes and how to combine them with Boolean operations to make more complicated shapes.

1.4 Building Geometries onto Grids

Suppose it is desired to simulate a wave scattering from a cylinder as shown in Figure 1.10(a). How is the simulation told what it is simulating, the shape the device, the material properties, etc.? For the FDFD method, this information is conveyed through the relative permittivity $\epsilon_r(i,j)$ and relative permeability $\mu_r(i,j)$ at each point of the grid. To do this in MATLAB, two arrays will be constructed, one to describe the relative permittivity ER and one to describe the relative permeability UR . If done correctly, plotting either of these arrays will show a picture of what is to be simulated, as illustrated in Figure 1.10(b), which shows the relative permittivity array ER for a cylinder. In this case, the array ER is filled with the value ϵ_{r2} at all points that lie inside of the cylinder and the value ϵ_{r1} at all points outside of the cylinder.



(a) FDFD simulation



(b) Relative permittivity array ER

Figure 1.10 (a) FDFD simulation of scattering from a dielectric cylinder. (b) Relative permittivity array ER that completely describes the cylinder to be simulated.

Building geometries into arrays is often the most difficult for students. The following sections cover the techniques required to build a wide variety of different geometries into data arrays so that different devices can be simulated. Complicated shapes can usually be created from combinations of simple shapes. With practice, these techniques will become intuitive and easy.

1.4.1 Adding Rectangles to a Grid

The simplest shapes to build onto a grid are squares and rectangles. They are also very common shapes for real devices in electromagnetics and photonics. Diffraction

gratings and polarizers are often described well by rectangles in their cross section. Rectangles are also elements of more complicated geometries. For example, a triangle can be thought of as a stack of thin rectangles of varying lengths. Rectangles can be added to a grid simply by calculating the array indices where the shape begins and ends and then adding the numbers to the array. For example, the MATLAB code below builds a rectangle into the array `ER`. The result is illustrated in Figure 1.11.

```

1  Nx = 10;
2  Ny = 15;
3
4  nx1 = 5;
5  nx2 = 8;
6  ny1 = 6;
7  ny2 = 13;
8
9  ER = ones(Nx,Ny);
10 ER(nx1:nx2,ny1:ny2) = 6;
```

In practice, the array indices will not be hard-coded like in this example. Instead, they will be calculated from parameters defined in the dashboard. This practice will be demonstrated in later chapters when FDFD implementation is discussed.

1.4.2 The Centering Algorithm

Many times it is desired to center a device on a grid. For this example, the same rectangle depicted in Figure 1.11 will be centered on the grid. This begins by calculating how many cells wide nx and how many cells tall ny the rectangle is. Given these variables, the start and stop array indices are calculated using the MATLAB code below.

```

1  nx = 4;
2  ny = 8;
3  nx1 = 1 + floor((Nx - nx)/2);
4  nx2 = nx1 + nx - 1;
5  ny1 = 1 + floor((Ny - ny)/2);
6  ny2 = ny1 + ny - 1;
```

The concept of this technique is as follows. The array index corresponding to the center of the grid in the x -direction is $Nx/2$, temporarily ignoring that this quantity must be rounded to an integer. This means the array index $nx1$ where the rectangle starts should be $Nx/2$ minus half of the number of cells the rectangle is wide $nx/2$. Putting these together means the starting array index is $Nx/2 - nx/2$, which is simplified to $(Nx - nx)/2$. To get an integer array index, this is rounded down to the nearest integer using MATLAB's `floor()` function. To prevent getting an array index of 0, the value of 1 is added to the rounded integer. The final equation to calculate $nx1$ is on line 3 of the MATLAB code immediately above. Calculating the stop index $nx2$ is much easier after the start index $nx1$ is known. The stop index $nx2$ is $nx2 = nx1 + nx - 1$. The final equation to calculate $nx2$ is on line 4 of the MATLAB code

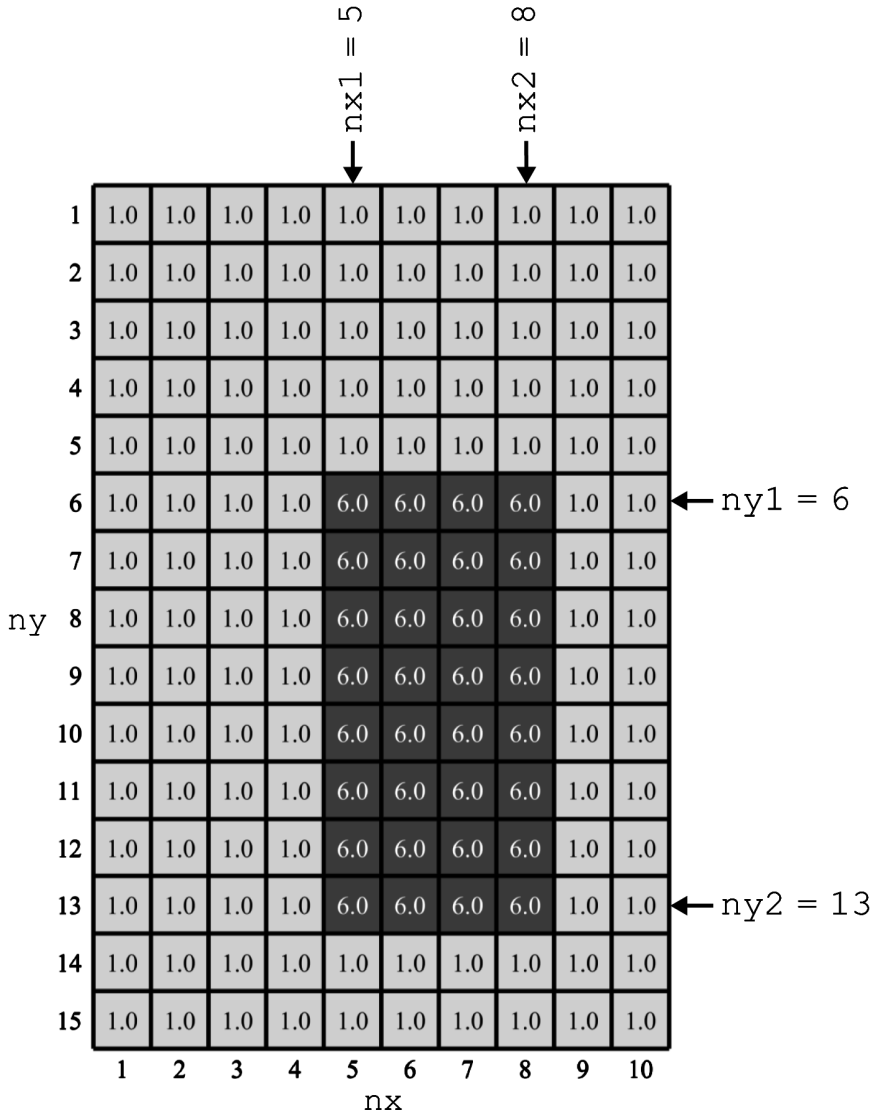


Figure 1.11 Building a rectangle onto a grid.

immediately above. The reason for the -1 is so the rectangle occupies the correct number of cells on the grid to accurately represent its size. In this example, $n_x=4$ and $n_{x1}=4$. If the -1 were not used, the stop index n_{x2} would be calculated to be 8, causing five cells to be filled in on the grid (4, 5, 6, 7, and 8). Doing this would make the rectangle one cell larger than it should be. While this may seem minimal, it can lead to very slow convergence requiring very high grid resolution to make the one-cell error insignificant. It is best practice to take great care to always use the most accurate number of cells. The final centered rectangle is shown in Figure 1.12. Observe that the rectangle is not perfectly centered in the y-direction. This is because the height of the rectangle is an even number of cells and the height of the grid is an odd number of cells. Often, the position of the device on the grid is much

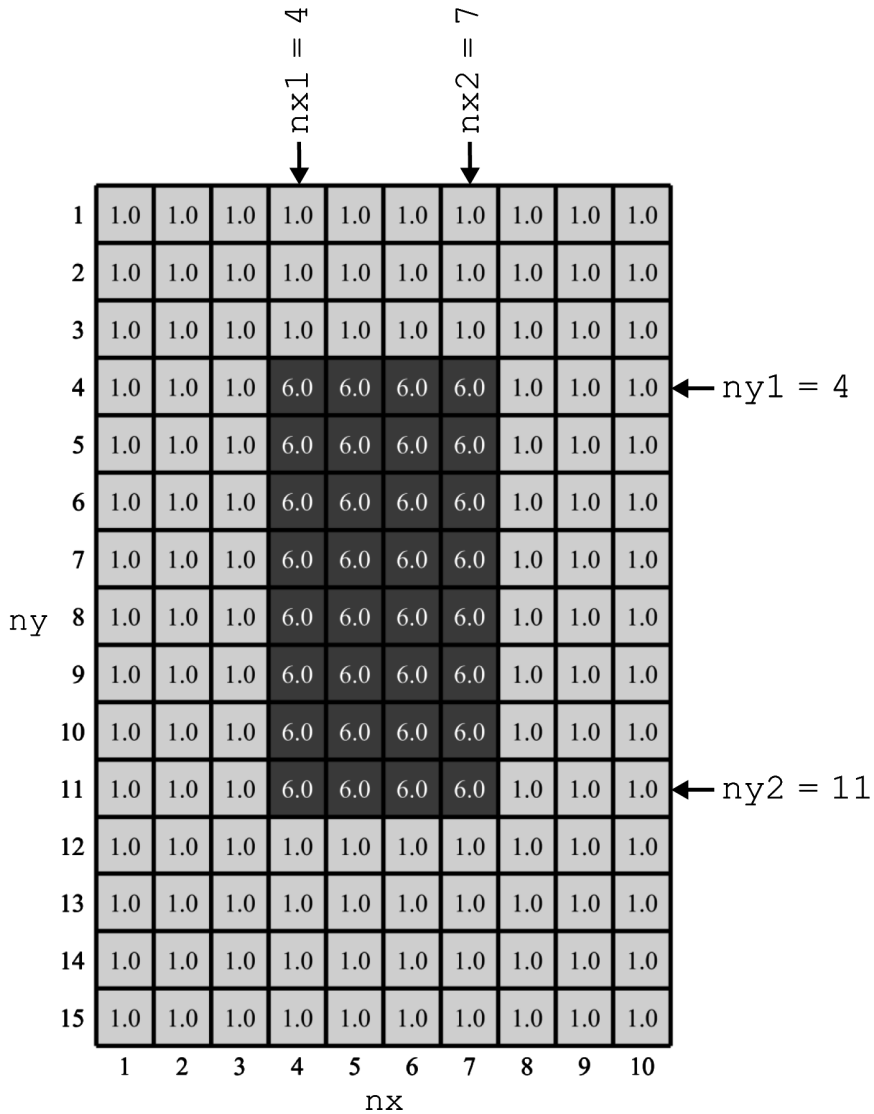


Figure 1.12 Centering a rectangle onto a grid.

less important than getting its dimensions correct. For this reason, there is more room to be lazy calculating $nx1$ than there is when calculating $nx2$ relative to $nx1$.

1.4.3 The Meshgrid

While rectangles are common shapes and easy to build onto a grid, it is cumbersome to use them for building shapes like circles and ellipses. The meshgrid technique in MATLAB makes the code to build a wide variety of shapes very simple and easy. The concept of the meshgrid and the variables related to it are illustrated in Figure 1.13. It starts with the array ER or UR that will be used to store the relative permittivity or relative permeability, respectively. The parameters that describe the grid

associated with these arrays were described previously. Two axis arrays, x_a and y_a , must be defined before the meshgrid parameters X and Y can be calculated. The array x_a contains the center position of all the cells in the x -direction, while the array y_a contains the center position of all the cells in the y -direction. These are illustrated below and to the left of the ER (or UR) array shown in Figure 1.13. The color in the cells of x_a and y_a represents the number values that are stored in those arrays.

Given the axis arrays x_a and y_a , the meshgrid parameters X and Y are calculated using MATLAB's function `meshgrid()`. Since X and Y are associated with arrays and not matrices, the order of x and y is swapped when calling the `meshgrid()` function.

```
[Y,X] = meshgrid(ya,xa)
```

Both X and Y are arrays the same size as the grid for ER (or UR). The number values in the array X contain the position of those number values in the x -direction. For this reason, the numbers in any vertical column of X are the same. The number values in the array Y contain the position of those number values in the y -direction. For this reason, the numbers in any horizontal row of Y are the same. This redundant information may seem like a waste of computer memory, but the utility of the meshgrid arrays will be demonstrated shortly.

1.4.4 Adding Circles and Ellipses to a Grid

The meshgrid arrays X and Y make it very easy to build circles and ellipses onto grids. The general equation for an ellipse with center position (x_0, y_0) , radius r_x in the x -direction, and radius r_y in the y -direction is

$$\left(\frac{x - x_0}{r_x} \right)^2 + \left(\frac{y - y_0}{r_y} \right)^2 = 1 \quad (1.22)$$

Using the meshgrid parameters X and Y , this ellipse can be built into the array ER simply by typing (1.22) directly into MATLAB.

```
ER = ((X - x0)/rx).^2 + ((Y - y0)/ry).^2 <= 1
```

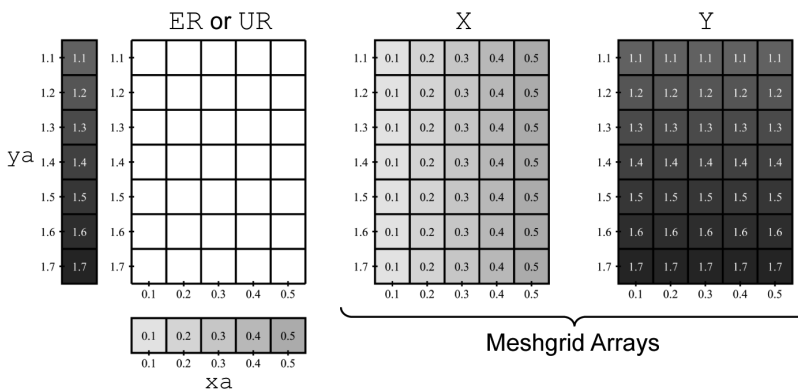


Figure 1.13 Illustration of the meshgrid arrays X and Y .

The meshgrid makes it incredibly easy to build an ellipse and the code to do it is very simple and easy to read. Figure 1.14 overlays the ellipse onto the final array `ER` where the ellipse has been incorporated using `meshgrid`. Observe how the dimensions of the ellipse are conveyed in the `ER` array.

The MATLAB code to build the ellipse into the array `ER` is provided below. The code begins with a short dashboard that defines the physical size of the grid with `Sx` and `Sy`, the number of points on the grid with `Nx` and `Ny`, the center of the ellipse with `x0` and `y0`, and the radii of the ellipse in the x - and y -directions with the variables `rx` and `ry`, respectively. The next parameters calculated are the grid resolution parameters `dx` and `dy`, axis arrays `xa` and `ya`, and finally the meshgrid arrays `X` and `Y`. The very last line uses the meshgrid arrays to build the ellipse in a single and simple line of code that comes directly from (1.22).

```
% DASHBOARD
Sx = 1;
Sy = 1;
Nx = 20;
Ny = 20;

x0 = 0.3;
y0 = 0.6;
rx = 0.2;
ry = 0.35;

% CALCULATE GRID
dx = Sx/Nx;
```

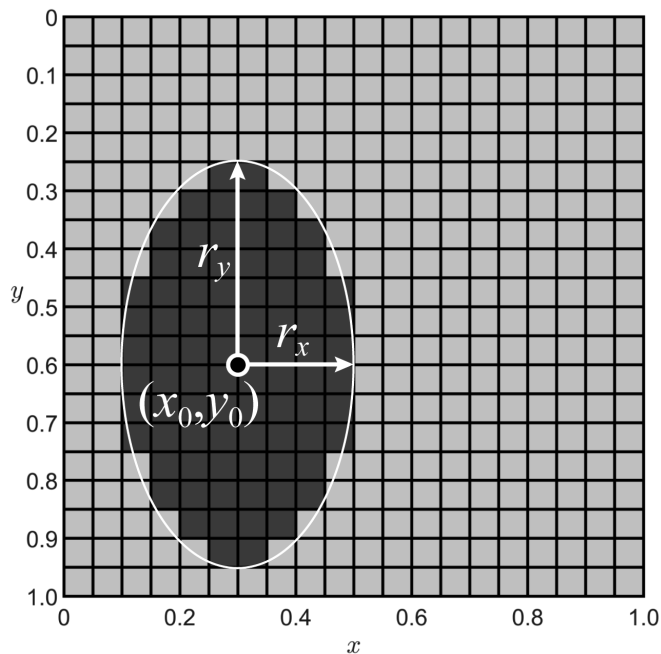


Figure 1.14 An ellipse is built into the array `ER`.

```

dy = Sy/Ny;
xa = [0.5:Nx-0.5]*dx;
ya = [0.5:Ny-0.5]*dy;
[Y,X] = meshgrid(ya,xa);

% BUILD ELLIPSE
ER = ((X - x0)/rx).^2 + ((Y - y0)/ry).^2 <= 1;

```

While this code builds an ellipse, it is easily modified to build a circle. One way to do this is to simply ensure that r_x and r_y are equal. Alternatively, and more simply, the last line above could be replaced with

```
ER = (X - x0).^2 + (Y - y0).^2 <= r^2;
```

Observe that neither the equation for the ellipse nor the circle calculate the square root of the meshgrid arrays. Instead, the radius quantities are squared so that only a single number has to be squared instead of calculating the square root of every number in the meshgrid. This is done for speed and efficiency.

1.4.5 Grid Rotation

Sometimes it is desired to build a shape onto a grid that is tilted by some angle. Later, this technique will be applied to calculate Gaussian beams at oblique angles. Both of these tasks are easily accomplished by rotating the meshgrid parameters. Suppose it is desired to build an ellipse as shown in Figure 1.14, but rotated counter-clockwise by 30° . The final rotated ellipse is depicted in Figure 1.15.

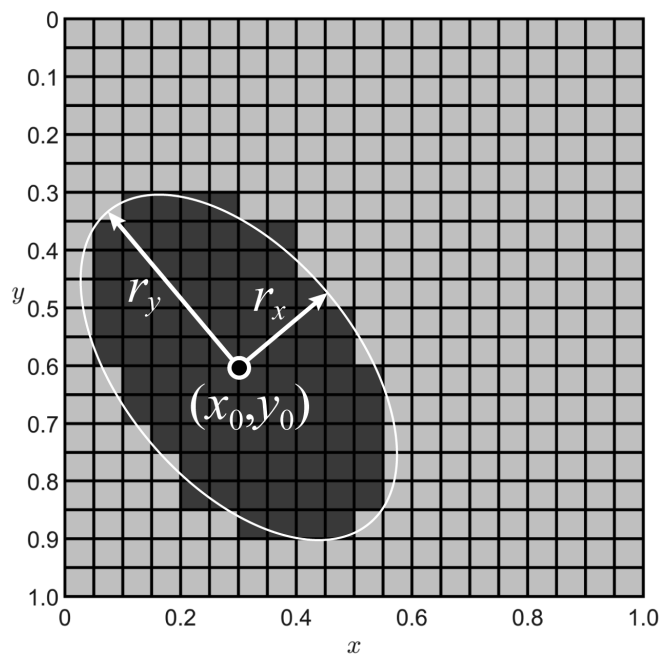


Figure 1.15 Building a rotated ellipse into the array ER.

The MATLAB code to build this rotated ellipse is provided below. The first step is to convert the Cartesian meshgrid parameters X and Y into polar coordinates TH and R using MATLAB's `cart2pol()` function. The meshgrid parameters passed to `cart2pol()` are made to equal zero at the point (x_0, y_0) by subtracting these values from X and Y . This makes the new meshgrid rotate about the center of the ellipse being built onto the grid. Next, the polar coordinates are immediately converted back to Cartesian coordinates to get a second set of meshgrid arrays XR and YR . This is the step where some angle θ is added to TH , making XR and YR the rotated meshgrid arrays. Now the ellipse is calculated using the rotated meshgrid terms in the exact same way it would be calculated using the standard meshgrid terms.

```
% UNITS
degrees = pi/180;

% DASHBOARD
Sx = 1;
Sy = 1;
Nx = 20;
Ny = 20;
x0 = 0.3;
y0 = 0.6;
rx = 0.2;
ry = 0.35;
theta = 30*degrees;

% CALCULATE GRID
dx = Sx/Nx;
dy = Sy/Ny;
xa = [0.5:Nx-0.5]*dx;
ya = [0.5:Ny-0.5]*dy;
[Y,X] = meshgrid(ya,xa);
[TH,R] = cart2pol(X-x0,Y-y0);
[XR,YR] = pol2cart(TH+theta,R);

% BUILD ELLIPSE
ER = (XR/rx).^2 + (YR/ry).^2 <= 1;
```

1.4.6 Boolean Operations

To build more complicated geometries onto FDFD grids, Boolean operations are very useful. Many complicated shapes can be envisioned as a combination of simpler shapes. Boolean operations can be used to combine multiple shapes, subtract one shape from another, and more. Figure 1.16 shows various types of Boolean operations that can be performed easily in MATLAB. The top two arrays labeled A and B are two squares offset from each other. The other arrays show how A and B can be combined in multiple ways to form a wide array of new shapes.

Observe that the original arrays A and B contain only zeros and ones instead of relative permittivity or relative permeability. When Boolean operations are going to

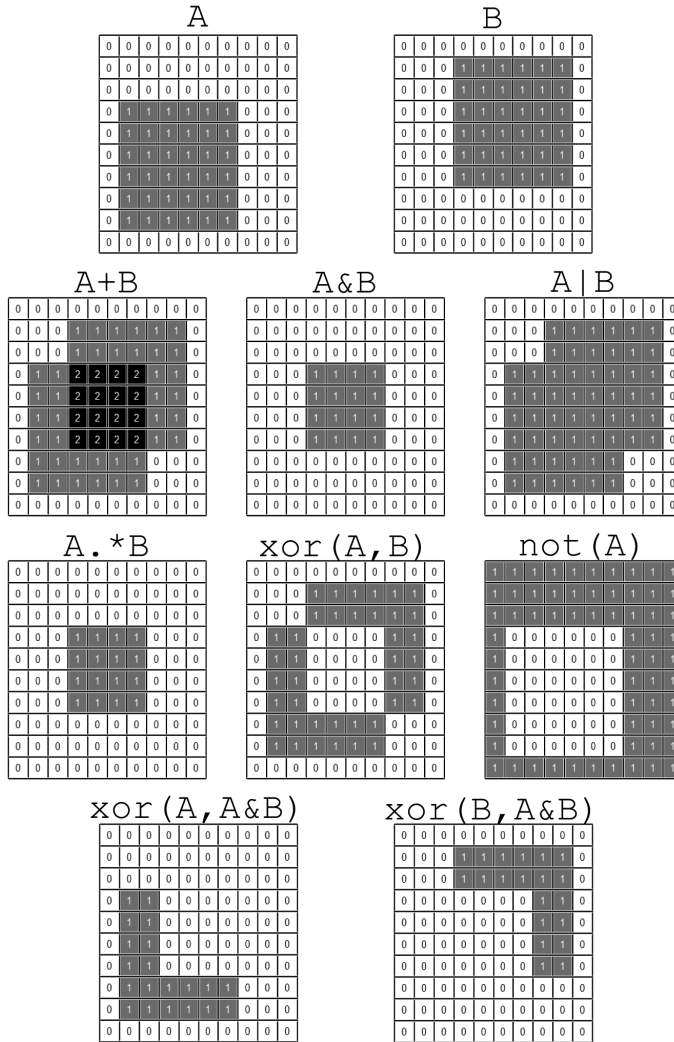


Figure 1.16 Examples of various Boolean operations in MATLAB.

be used, it is very useful to build the simple shapes from just zeros and ones so that the Boolean operations can be performed more naturally. When the final geometry resides on the grid, the zeros and ones are easily converted to physical values of relative permittivity and/or relative permeability. The concept of this calculation is illustrated in Figure 1.17. At left is a grid containing a square with a circle subtracted from it using a Boolean operation. This might be a unit cell of a photonic crystal, for example. The array at this point contains only zeros and ones to facilitate the Boolean operation.

If the structure depicted in Figure 1.17 is made of a material with relative permittivity $\epsilon_{r2}=9$ and is embedded in a medium with relative permittivity of $\epsilon_{r1}=2$, the zeros must be replaced with the value of ϵ_{r1} and the ones must be replaced with the value of ϵ_{r2} . One simple way to do this is using (1.23).

$$ER = \epsilon_{r1} + (\epsilon_{r2}-\epsilon_{r1}) * ER \quad (1.23)$$

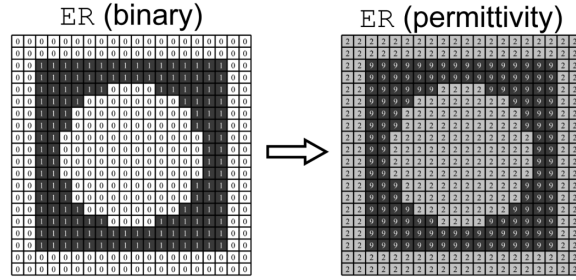


Figure 1.17 Converting binary arrays to arrays containing real values of relative permittivity.

The first occurrence of `er1` in (1.23) places that value throughout the entire array `ER`. The second term $(er2 - er1) * ER$ adds the difference $er2 - er1$ to the cells on the grid where $er2$ is to be placed. Since $er1$ is the current value in those cells, $er2 - er1$ is added to them to get a final value of $er2$. With some practice, this will become very intuitive and techniques for assigning more than two material values to the grid are straightforward to implement.

1.5 Three-Dimensional Grids

Everything discussed so far is easily generalized to three dimensions for use in Chapter 10. All of the same techniques are used to calculate the grid parameters and build geometries onto the grid. To demonstrate, suppose it is desired to build an ellipsoid onto a three-dimensional grid with radii $r_x = 0.4$, $r_y = 0.3$, and $r_z = 1.0$. The MATLAB code to do this is provided below.

```

1    % demo_3Dgrid.m
2
3    % INITIALIZE MATLAB
4    close all;
5    clc;
6    clear all;
7
8    % DASHBOARD
9    rx = 0.4;
10   ry = 0.3;
11   rz = 1.0;
12
13   er1 = 1.0;
14   er2 = 9.0;
15
16   Sx = 1;
17   Sy = 1;
18   Sz = 3;

```

```

19  Nx = 20;
20  Ny = 20;
21  Nz = 60;
22
23  % CALCULATE 3D MESHGRID
24  dx = Sx/Nx;
25  xa = [1:Nx]*dx;
26  xa = xa - mean(xa);
27
28  dy = Sy/Ny;
29  ya = [1:Ny]*dy;
30  ya = ya - mean(ya);
31
32  dz = Sz/Nz;
33  za = [1:Nz]*dz;
34  za = za - mean(za);
35
36  [Y,X,Z] = meshgrid(ya,xa,za);
37
38  % BUILD ELLIPSE
39  ER = (X/rx).^2 + (Y/ry).^2 + (Z/rz).^2 < 1;
40  ER = er1 + (er2 - er1)*ER;

```

Line 1 is a comment with the name of the program. Lines 3 to 6 initialize MATLAB. Lines 8 to 21 comprise the dashboard where everything about the program is controlled. The radii of the ellipsoid are defined on lines 9 to 11 as r_x , r_y , and r_z . Lines 13 and 14 define the relative permittivity values to be assigned to this grid. $er1$ is the relative permittivity outside of the ellipsoid and $er2$ is the relative permittivity of the ellipsoid. Lines 16 to 21 define a simple three-dimensional grid in terms of the physical size S_x , S_y , and S_z and the number of points along each dimension N_x , N_y , and N_z . The three-dimensional meshgrid is calculated from lines 24 to 36 in four groups of code. The first three groups of code calculate the grid resolution parameters dx , dy , and dz as well as the axis arrays x_a , y_a , and z_a . Line 36 calculates the meshgrid parameters X , Y , and Z . The meshgrid parameters are all three-dimensional arrays of size N_x -by- N_y -by- N_z . Last, lines 38 to 40 build the ellipsoid onto the three-dimensional grid. Line 39 actually builds the ellipsoid into the array ER where 0's are placed outside of the ellipsoid and 1's are placed inside of the ellipsoid. This makes the ellipsoid suitable for Boolean operations with other structures if that is desired. Line 40 converts the 0's and 1's to the actual values of relative permittivity. This line works by assigning the value of $er1$ to all points in the array ER and then adding $(er2 - er1)$ at the points in ER that contain 1's. After this line, all 0's in ER are replaced with $er1$ and all 1's in ER are replaced with $er2$.

The output of the program described above is the three-dimensional array ER containing the relative permittivity throughout the grid. Figure 1.18 shows the array ER that is constructed by the above MATLAB program. The axes are labeled with the array indices instead of the physical position along the grid for illustration purposes. For real simulation work, it is usually more meaningful to label the axes with physical dimensions.

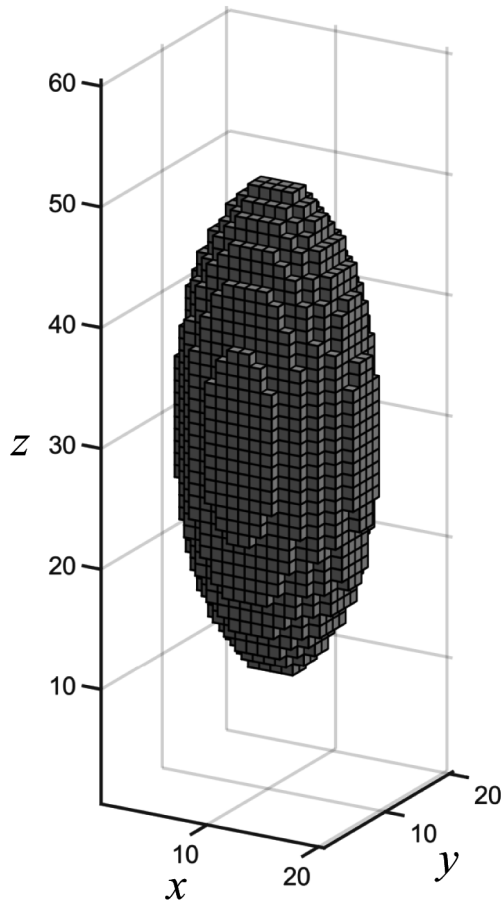


Figure 1.18 Ellipsoid built onto a three-dimensional grid.

1.6 Visualization Techniques

1.6.1 Visualizing Data on Grids

The two most common functions in MATLAB to visualize data arrays arising in FDFD are `imagesc()` and `pcolor()`. The key differences between these functions can be observed in Figure 1.19. The top two images show the same 5×5 array visualized with both functions. Notice that `imagesc()` clearly shows that there are 5×5 discrete numbers in the array while `pcolor()` seems to imply there are only 4×4 numbers. The function `imagesc()` is what the author uses for his day-to-day simulation tasks because it more precisely and intuitively conveys discrete functions. The function `pcolor()` is more intended to visualize smooth functions and is performing interpolations to smooth the data. It assigns the values from the array to the axis lines. The discrete shades shown in the top right image are essentially interpolated from the data values at the edges, thus the 5×5 set of data appears to be 4×4 . While this is not good for inspecting digital data, it is an excellent way to produce a very nice-looking final image of the fields calculated from a simulation.

Even low-resolution data can be made to look smooth and continuous. Also notice the orientation of the vertical axis. The function `imagesc()` inverts the y -axis so the numbers increase going downward. In contrast, `pcolor()` orients the vertical axis so that numbers increase going upward. This was discussed previously and the preferred orientation is to have numbers increase going downward like that produced by `imagesc()`. Last, observe that `pcolor()` adds coordinate lines by default to the image. When arrays contain many points, the black lines become so dense the entire image becomes black. In FDFD, it will be a standard practice to turn off those lines and to reverse the y -direction when using `pcolor()`. The bottom row of Figure 1.19 visualizes arrays containing 15×15 points. The image generated by `pcolor()` has the black lines removed and the y -axis reversed to match `imagesc()`. The application of `pcolor()` to visualize smooth data is apparent.

A key issue about `imagesc()` and `pcolor()` is that both assume the array being visualized is a matrix. Both graphics commands place the first dimension of the array along the vertical axis and the second dimension of the array along the horizontal axis. This is the exact opposite of how arrays will be handled throughout this book. To correct this, the arrays are simply transposed to put them in the correct orientation for visualization.

The MATLAB code to visualize an array R is provided below. Observe that the array R is transposed in the calls to `imagesc()` and `pcolor()` on lines 29 and 34, respectively. The axis lines produced by default for `pcolor()` are removed using shading `interp` on line 35. The direction of the y -axis for `pcolor()` is reversed

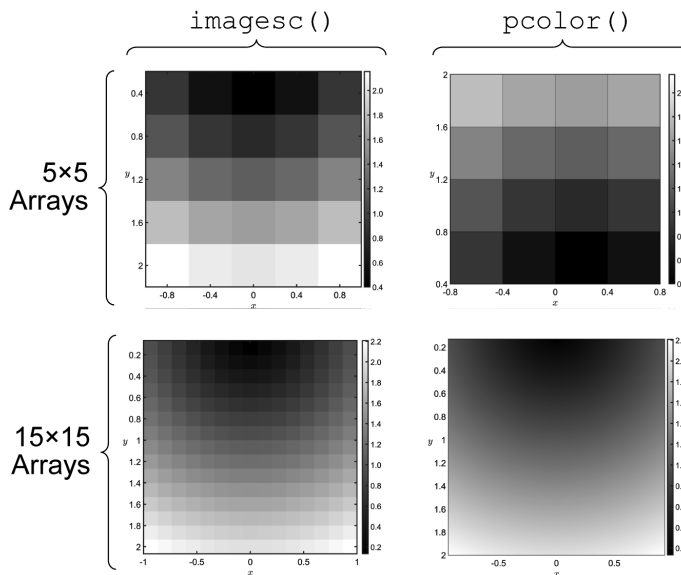


Figure 1.19 Visualizing data in arrays with `imagesc()` and `pcolor()`.

by setting the property 'YDir' to 'reverse' on line 37. Otherwise, the code uses standard graphics procedures.

```
1  % imagescpcolor.m
2
3  % INITIALIZE MATLAB
4  close all;
5  clc;
6  clear all;
7
8  % DASHBOARD GRID
9  Sx = 2;
10 Sy = 2;
11 Nx = 15;
12 Ny = 15;
13
14 % CALCULATE MESHGRID
15 dx = Sx/Nx;
16 xa = [1:Nx]*dx;
17 xa = xa - mean(xa);
18
19 dy = Sy/Ny;
20 ya = [1:Ny]*dy;
21
22 [Y,X] = meshgrid(ya,xa);
23
24 % CALCULATE DATA TO VISUALIZE
25 R = sqrt(X.^2 + Y.^2);
26
27 % VISUALIZE USING IMAGESC AND PCOLOR
28 subplot(121);
29 imagesc(xa,ya,R.');
```

1.6.2 Visualizing Three-Dimensional Data

Visualizing three-dimensional data is more difficult but can be done several different ways. By far the simplest in MATLAB is to use the built-in `slice()` function. The `slice()` function visualizes planes cut through a three-dimensional set of data, essentially using `pcolor()` to visualize each slice. The most common way to use `slice()` in FDFD is to visualize slices cut through the middle of the grid along each

plane. This can be done to visualize three-dimensional objects built onto a grid, as shown in Figure 1.20(a), or to visualize electromagnetic fields in three dimensions, as shown in Figure 1.20(b).

The MATLAB code to produce the visualizations in Figure 1.20 is provided below. The `slice` function takes three-dimensional meshgrid parameters X , Y , and Z as the first three input arguments. The fourth input argument is the three-dimensional array to be visualized. The fifth input argument is an array containing all of the positions along the x -axis to image slices. In this case, a single value of 0 is given to draw a single slice at the plane defined by $x = 0$. The sixth and seventh input arguments are similar arrays for the positions of the slices along the y - and z -axes. See the documentation for MATLAB for additional options for the `slice()` function.

```
slice(Y,X,Z,A,0,0,0);
shading flat;
grid on;
axis equal tight;
colorbar('LineWidth',3);
view(120,20);
camlight headlight;
```

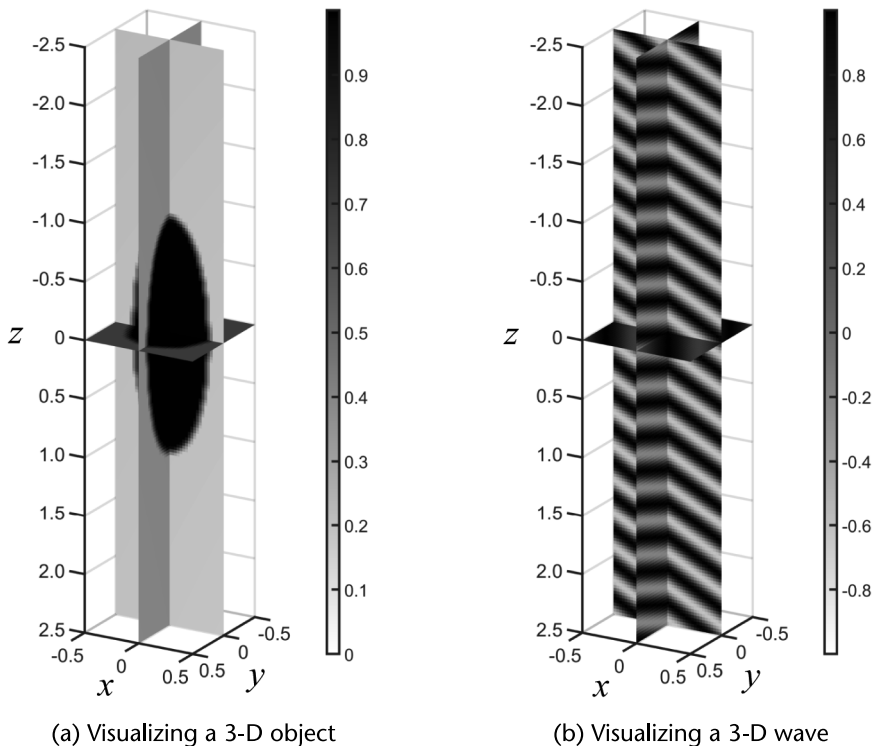


Figure 1.20 (a) Visualization of a three-dimensional ellipsoid object using `slice()`. (b) Visualizing a three-dimensional wave using `slice()`.

1.6.3 Visualizing Complex Data

The FDFD method is a frequency-domain simulation technique. This means it calculates the field f as an array of complex numbers to convey both amplitude and phase. A good question to ask is how an array of complex numbers should be visualized. Most of the time, visualizing just the real part of the array f will produce the most intuitive representation of the field from a simulation, but this can sometimes hide aspects of the field. Figure 1.21 shows the four primary ways of visualizing a complex field. In this example, a diverging Gaussian beam from an FDFD simulation is being visualized. Observe that the images produced by Figure 1.21(a, b) are essentially the same, but the phase appears different when comparing the real and imaginary parts. While both convey the same information, sometimes one may produce a better visualization of the field than the other. Figure 1.21(c) shows the absolute value of the field. This is excellent for visualizing where power is distributed, but is missing the wave nature of the field. In this case, the curvature of the wave fronts would be completely missed. Last, Figure 1.21(d) shows the phase part of the field. Due to wrapping of the phase and absence of conveying where power is distributed, this visualization is rarely used. There is no single best way to visualize complex data, and sometimes, all four visualizations are needed in order

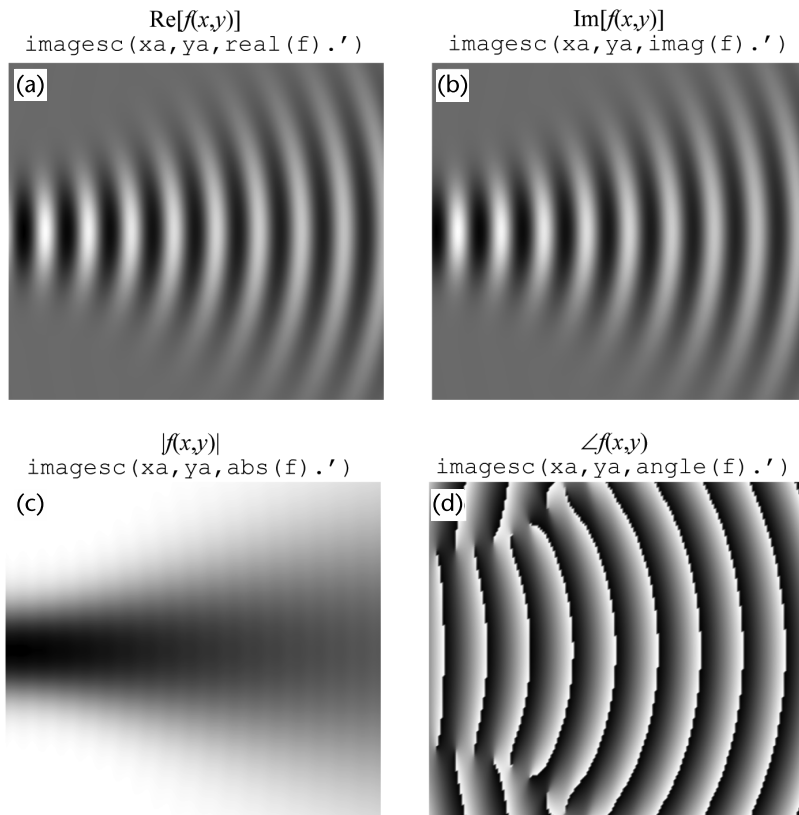


Figure 1.21 Four ways to visualize data in complex arrays: (a) real part, (b) imaginary part, (c) absolute value, and (d) phase.

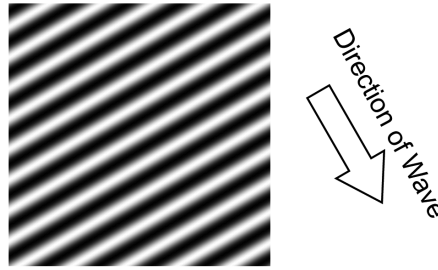


Figure 1.22 A single frame of the GIF animation of a plane wave.

to completely understand the field. For most work, the author prefers to visualize the real part of the field.

1.6.4 Animating the Fields Calculated by FDFD

The FDFD method is a frequency-domain method, so it obtains a snapshot of the fields at steady-state at a single frequency. Based on this, it would seem that it is not possible to create animations of the simulated fields propagating and scattering on the grid. This is false! It is absolutely possible to create stunning animations and these will help you to learn about your devices and to showcase your results. There are times where it can really help your career to show off! Consider making it standard practice for you to show an animated GIF of your simulation in a presentation instead of a boring static image like everybody else.

The rest of this book will teach you how to simulate a wide variety of devices using the FDFD method. In the end, you will have an array `f` containing the frequency-domain solution to the simulation. `f` is the array from which the animation will be generated. The MATLAB code to create an animated GIF from the complex field `f` can be downloaded from <https://empossible.net/fdfdbook/>. It is called `Chapter1_GIFanimation.m` and is described below. The code essentially captures frames over one wave cycle in a way that if repeated will produce the illusion that the wave keeps propagating. It is typical to use around 40 frames in a GIF animation. The program saves the GIF with the name “FDFD_animation.gif.” To replace a real FDFD simulation, this code calculates a plane wave propagating vertically at an angle of 30° in the array `f`. Lines 37 to 64 create the animation from the array `f` and these lines can be copied and pasted to the end of any FDFD program to animate the results. A single frame from the animation created from the code above is shown in Figure 1.22. The direction of the wave is downward and slightly to the right consistent with the angle `theta`. This is a common source used in FDFD analysis.

Reference

- [1] Gentle, J. E., “Matrix Algebra Theory, Computations and Applications in Statistics” *Springer Texts in Statistics*, New York: Springer, Vol. 10, 2007.

Electromagnetic Preliminaries

This chapter will review the key electromagnetic concepts and equations needed to understand and implement the finite-difference frequency-domain (FDFD) method. Maxwell's equations and the constitutive relations are introduced and expanded into Cartesian coordinates that will be used in the FDFD method. From here, the wave equation is derived and some properties of waves are discussed including polarization and the dispersion relation. The scattering of waves at an interface and how polarization is defined relative to the interface are reviewed. The conditions are explained where Maxwell's equations can be reduced to analyze devices in just two dimensions. To set up how reflection and transmission will be calculated in FDFD for periodic structures, diffraction gratings are reviewed as well as diffraction efficiency of the diffraction orders. Waveguides and transmission lines will be briefly discussed to understand the terminology, parameters, and mathematical form of the modes associated with the devices. Last, scalability in electromagnetics is discussed along with its implications and applications in FDFD.

2.1 Maxwell's Equations

Maxwell's equations are an incomplete set of equations that describe classical electromagnetics [1–5]. The most general form of Maxwell's equations is the time-domain integral form.

$$\oiint_S \vec{D}(t) \cdot d\vec{s} = \iiint_V \rho_v(t) dv \quad \text{Gauss' Law for Electric Fields} \quad (2.1)$$

$$\oiint_S \vec{B}(t) \cdot d\vec{s} = 0 \quad \text{Gauss' Law for Magnetic Fields} \quad (2.2)$$

$$\oint_L \vec{E}(t) \cdot d\vec{\ell} = -\iint_S \left[\frac{\partial \vec{B}(t)}{\partial t} \right] \cdot d\vec{s} \quad \text{Faraday's Law} \quad (2.3)$$

$$\oint_L \vec{H}(t) \cdot d\vec{\ell} = \iint_S \left[\vec{J}(t) + \frac{\partial \vec{D}(t)}{\partial t} \right] \cdot d\vec{s} \quad \text{Ampere's Circuit Law} \quad (2.4)$$

There are two ways that electric fields and static electric charges can store energy. First, energy can be stored in the electric field itself because energy can be propagated through the vacuum of space in electromagnetic waves. This is the *electric*

field intensity $\vec{E}(t)$ and it has units of volts per meter (V/m). The electric field intensity is most closely associated with voltage and force. Second, energy can be stored in the matter as displaced charge, like in a battery or the dielectric in a capacitor. The *electric flux density* $\vec{D}(t)$ includes both forms of electric energy and has units of Coulombs per square meter (C/m²). It is most closely associated with charge.

Similarly, there are two ways that magnetic fields and moving electric charges can store energy. First, energy can be stored in the magnetic field itself because energy can be propagated through the vacuum of space in electromagnetic waves. This is the *magnetic field intensity* $\vec{H}(t)$ and it has units of Amperes per meter (A/m). The magnetic field intensity is most closely associated with electric current. Second, magnetic energy can be stored in matter as rotated magnetic dipoles. The *magnetic flux density* $\vec{B}(t)$ includes both forms of magnetic energy and has units of Webers per square meter (Wb/m²) or Tesla (T). Magnetic flux is most closely associated with force. The needle of a compass aligns with the magnetic flux.

Gauss' law for electric fields equates two different ways of calculating the total charge enclosed in a volume. The total charge can be obtained either by integrating the electric flux $\vec{D}(t)$ through a closed surface or by integrating the *electric charge density* ρ_v (C/m³) throughout the volume enclosed by the same surface. Since there is no such thing as a magnetic charge, Gauss' law for magnetic fields equals zero and only contains the term integrating the magnetic flux. Faraday's law equates two different ways of calculating the *electromotive force* (EMF). The EMF can be calculated from a line integral of the electric field intensity around a closed path or by integrating the rate of change of the magnetic flux through the surface enclosed by the path. A negative sign is incorporated to enforce the negative sign convention. Ampere's circuit law equates two different ways of calculating the total electric current passing through an area. The total current can be calculated from a line integral of the magnetic field intensity around a path enclosing the area or by integrating the *electric current density* $\vec{J}(t)$ (A/m²) plus the rate of change of the electric flux density passing through that area.

Observe that Maxwell's equations do not contain any material parameters. For this reason, Maxwell's equations only describe the manner in which electric and magnetic fields are created and interact with each other. They do not directly describe how the fields interact with matter. This information comes from the *constitutive relations* [2].

$$\vec{D}(t) = \epsilon(t) * \vec{E}(t) \quad \text{Electric Response of Matter} \quad (2.5)$$

$$\vec{B}(t) = \mu(t) * \vec{H}(t) \quad \text{Magnetic Response of Matter} \quad (2.6)$$

In these equations, $\epsilon(t)$ is the electric permittivity and has units of Farads per meter (F/m). The *electric permittivity* is a measure of how well a medium stores electric energy. It accounts for the ability of both matter and the field itself to store electric energy. Similarly, the *magnetic permeability* $\mu(t)$ has units of Henries per meter (H/m) and is a measure of how well a medium stores magnetic energy. The permittivity and permeability change with frequency. This is called *material dispersion*. This leads to the convolution operation observed in the time-domain constitutive relations.

One final equation is needed in order to describe all of classical electromagnetics. This is the *Lorentz force law* that describes the force electric and magnetic fields put on a charge q [1]. The Lorentz force law will not be used in this book.

$$\vec{F}(t) = q\vec{E}(t) + q[\vec{v}(t) \times \vec{B}(t)] \quad \text{Lorentz Force Law} \quad (2.7)$$

For FDFD, Maxwell's equations will be handled almost exclusively in the frequency-domain differential form. These are derived by Fourier transforming Maxwell's equations and applying Stoke's theorem and the divergence theorem [1].

$$\nabla \cdot \vec{D} = \rho_v \quad \text{Gauss' Law for Electric Fields} \quad (2.8)$$

$$\nabla \cdot \vec{B} = 0 \quad \text{Gauss' Law for Magnetic Fields} \quad (2.9)$$

$$\nabla \times \vec{E} = -j\omega\vec{B} \quad \text{Faraday's Law} \quad (2.10)$$

$$\nabla \times \vec{H} = \vec{J} + j\omega\vec{D} \quad \text{Ampere's Circuit Law} \quad (2.11)$$

In differential form, Gauss' law for electric fields states that electric flux will diverge from positive charge and converge on negative charge. In the absence of charge, the electric flux cannot diverge or converge so it cannot have a beginning or an end. This means the electric flux can only form loops when no charge is present. There is no magnetic charge so Gauss' law for magnetic fields states that magnetic flux cannot have divergence. That is, the magnetic flux cannot have a beginning or an end so it can only form loops. Faraday's law states that oscillating magnetic flux will have an electric field circulating around it. Ampere's circuit law states that a circulating magnetic field will exist around either an oscillating electric flux or an electrical current. In FDFD, the two divergence equations play the most critical role in how Maxwell's equations are made discrete, and the two curl equations play the most critical role in equations used for FDFD.

Last, the constitutive relations in the frequency-domain simplify considerably since they no longer contain convolutions.

$$\vec{D} = \epsilon\vec{E} \quad \text{Electric Response of Matter} \quad (2.12)$$

$$\vec{B} = \mu\vec{H} \quad \text{Magnetic Response of Matter} \quad (2.13)$$

The volume charge density ρ_v is not often used in electrodynamics (i.e., waves) problems because the waves being analyzed are away from free charges. For this reason, ρ_v will be set equal to zero and dropped from Gauss' law. It is common to substitute the constitutive relations into Maxwell's equations to eliminate the electric flux density \vec{D} and magnetic flux density \vec{B} terms. This gives

$$\nabla \cdot (\epsilon\vec{E}) = 0 \quad (2.14)$$

$$\nabla \cdot (\mu\vec{H}) = 0 \quad (2.15)$$

$$\nabla \times \vec{E} = -j\omega\mu\vec{H} \quad (2.16)$$

$$\nabla \times \vec{H} = \vec{J} + j\omega\epsilon\vec{E} \quad (2.17)$$

The electric current density \vec{J} in (2.17) is related to the electric field intensity through a form of Ohm's law in electromagnetics [2] where σ is the *electrical conductivity* of the medium.

$$\vec{J} = \sigma \vec{E} \quad (2.18)$$

Electrical conductivity σ has units of $\Omega \cdot \text{m}$. Substituting (2.18) into (2.17) gives

$$\nabla \times \vec{H} = \sigma \vec{E} + j\omega \epsilon \vec{E} \quad (2.19)$$

On the right-hand side of this equation, $j\omega$ and \vec{E} are factored out of the expression to obtain

$$\nabla \times \vec{H} = j\omega \left(\frac{\sigma}{j\omega} + \epsilon \right) \vec{E} \quad (2.20)$$

The expression in parentheses can be interpreted as a complex permittivity $\tilde{\epsilon}$ that accounts for both the permittivity ϵ and electrical conductivity σ at the same time. Using the complex permittivity $\tilde{\epsilon}$, (2.20) becomes

$$\nabla \times \vec{H} = j\omega \tilde{\epsilon} \vec{E} \quad (2.21)$$

where

$$\tilde{\epsilon} = \epsilon + \frac{\sigma}{j\omega} \quad (2.22)$$

It is so common to account for loss with a complex permittivity that the tilde notation is rarely used. In fact, the complex permittivity throughout this book will be written without the tilde even though it is a complex quantity. It is more common to communicate about the properties of a material through the complex relative permittivity $\tilde{\epsilon}_r$ than it is the complex permittivity $\tilde{\epsilon}$. Given the free space permittivity ϵ_0 , these parameters are related as follows.

$$\tilde{\epsilon} = \epsilon_0 \tilde{\epsilon}_r \quad (2.23)$$

In FDFD, material loss is handled simply by making the relative permittivity a complex number $\tilde{\epsilon}$. It is important to be cautious of sign convention when expressing the complex permittivity $\tilde{\epsilon}$. This book, and most of the engineering, adopts the negative sign convention where a wave propagating in the $+z$ -direction is written as $\exp(-jkz)$. The positive sign convention would express the same wave as $\exp(jkz)$. For the negative sign convention, the imaginary part of the complex permittivity is negative when a material has loss and waves decay as they propagate. When simulating active materials that have gain, the imaginary part of the complex permittivity is positive and waves grow as they propagate. This is consistent with (2.22) where the imaginary part has j in the denominator. When brought to the numerator, the imaginary part becomes negative. It is very easy to mistakenly use the wrong sign for the imaginary part of the complex permittivity.

2.2 The Constitutive Parameters

In a vacuum, the permittivity becomes the *free space permittivity*, or *vacuum permittivity*, and is

$$\epsilon_0 = 8.8541878176 \times 10^{-12} \text{ F/m} \quad (2.24)$$

The permittivity ϵ of any other material can be written as the product of the vacuum permittivity ϵ_0 and the *relative permittivity* ϵ_r , also called the *dielectric constant*.

$$\epsilon = \epsilon_0 \epsilon_r \quad (2.25)$$

The relative permittivity ϵ_r is a much more convenient quantity to describe materials than the permittivity ϵ because the relative permittivity has no units and is typically between the values of 1 and 12. When loss is considered, the relative permittivity becomes complex $\tilde{\epsilon}$. Metals are commonly characterized by a real-valued relative permittivity ϵ_r and a conductivity σ through (2.26). Many times only the conductivity σ is specified for metals and the relative permittivity is assumed to be $\epsilon_r \approx 1$.

$$\tilde{\epsilon}_r = \epsilon_r + \frac{\sigma}{j\omega\epsilon_0} \quad (2.26)$$

At radio frequencies, it is common to specify dielectrics through a real-valued relative permittivity ϵ_r and a *loss tangent* $\tan\delta$. The loss tangent does not have any units. The complex relative permittivity $\tilde{\epsilon}$ is calculated from these parameters according to

$$\tilde{\epsilon}_r \equiv \epsilon_r (1 - j \tan\delta) \quad (2.27)$$

Similarly, the permeability μ can be written as the *free space permeability*, or *vacuum permeability*, μ_0 times the relative permeability μ_r .

$$\mu = \mu_0 \mu_r \quad (2.28)$$

$$\mu_0 = 1.2566370614 \times 10^{-6} \text{ H/m} \quad (2.29)$$

Following a similar line of reasoning, the relative permeability can also be a complex number $\tilde{\mu}$. When simulating physical materials, it is rare to have a complex permeability. However, complex permeability arises frequently with metamaterials [6]. This will be discussed in detail in Chapter 10.

2.2.1 Anisotropy, Tensors, and Rotation Matrices

Isotropic materials exhibit the same electromagnetic properties regardless of the direction of the fields. The permittivity and permeability of isotropic mediums are scalar quantities. Some materials, however, can exhibit different electromagnetic properties depending on the direction of the fields. Such materials are called *anisotropic* [2, 7, 8]. Anisotropy arises in crystalline structures when atomic-scale charges are more easily displaced in some directions than others. For anisotropic

materials, the constitutive parameters are tensor quantities and the constitutive relations are written as

$$\vec{D} = [\epsilon] \vec{E} \quad \text{or} \quad \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \quad (2.30)$$

$$\vec{B} = [\mu] \vec{H} \quad \text{or} \quad \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} \mu_{xx} & \mu_{xy} & \mu_{xz} \\ \mu_{yx} & \mu_{yy} & \mu_{yz} \\ \mu_{zx} & \mu_{zy} & \mu_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \quad (2.31)$$

Handling full nine-element tensors in FDFD is more complicated and will be described in Chapter 10. For all other chapters in this book, anisotropy will be restricted to *diagonally anisotropic* media where all of the off-diagonal terms in the tensors are zero and FDFD simplifies considerably.

$$\begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} \epsilon_{xx} & 0 & 0 \\ 0 & \epsilon_{yy} & 0 \\ 0 & 0 & \epsilon_{zz} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \quad (2.32)$$

$$\begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} \mu_{xx} & 0 & 0 \\ 0 & \mu_{yy} & 0 \\ 0 & 0 & \mu_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \quad (2.33)$$

We live in a three-dimensional world so there are only three dimensions that an electric or magnetic field can be directed toward. For this reason, electric fields can only experience a combination of three different permittivity values, and magnetic fields can only experience only a combination of three different permeability values. These are called the *principal values* and these occur in the directions of the *principal axes* of the anisotropic medium. When the anisotropic medium is analyzed in a coordinate system that matches perfectly to the principal axes, the permittivity and permeability tensors are diagonal and contain the principal values along the center diagonal.

$$[\epsilon_r] = \begin{bmatrix} \epsilon_a & 0 & 0 \\ 0 & \epsilon_b & 0 \\ 0 & 0 & \epsilon_c \end{bmatrix} \quad (2.34)$$

$$[\mu_r] = \begin{bmatrix} \mu_a & 0 & 0 \\ 0 & \mu_b & 0 \\ 0 & 0 & \mu_c \end{bmatrix} \quad (2.35)$$

Isotropic materials have all of the principal values equal ($\epsilon_a = \epsilon_b = \epsilon_c$) and the tensor reduces to a single scalar quantity. Materials where only two of the principal values are equal ($\epsilon_a = \epsilon_b \neq \epsilon_c$) are called *uniaxial* [8]. Materials where all three of the principal values are different ($\epsilon_a \neq \epsilon_b \neq \epsilon_c$) are called *biaxial* [8]. The subscripts of the principal values in (2.34) and (2.35) are written as a , b , and c to indicate that the principal axes can be in directions other than x , y , and z . The principal axes of an anisotropic medium do not even have to be perpendicular to each other. When the principal axes align perfectly with the Cartesian axes, the tensors are diagonal, and (2.34) and (2.35) are written in terms of the Cartesian axes as

$$[\epsilon_r] = \begin{bmatrix} \epsilon_{xx} & 0 & 0 \\ 0 & \epsilon_{yy} & 0 \\ 0 & 0 & \epsilon_{zz} \end{bmatrix} \quad (2.36)$$

$$[\mu_r] = \begin{bmatrix} \mu_{xx} & 0 & 0 \\ 0 & \mu_{yy} & 0 \\ 0 & 0 & \mu_{zz} \end{bmatrix} \quad (2.37)$$

When the principal axes of the anisotropic medium are not aligned with the Cartesian axes, off-diagonal terms arise in the tensors. The diagonal element ϵ_{mn} in the tensor $[\epsilon_r]$ is interpreted as how much of electric field intensity component E_n contributes to electric flux density component D_m . The diagonal element μ_{mn} in the tensor $[\mu_r]$ is interpreted as how much of magnetic field intensity component H_n contributes to magnetic flux density component B_m .

$$[\epsilon_r] = \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \quad (2.38)$$

$$[\mu_r] = \begin{bmatrix} \mu_{xx} & \mu_{xy} & \mu_{xz} \\ \mu_{yx} & \mu_{yy} & \mu_{yz} \\ \mu_{zx} & \mu_{zy} & \mu_{zz} \end{bmatrix} \quad (2.39)$$

2.2.2 Rotation Matrices and Tensor Rotation

It is very useful to know how to calculate full nine-element tensors from the principal values when the principal axes of the anisotropic medium are rotated relative to the Cartesian axes. One easy way to do this is to use *rotation matrices*. First, rotation matrices will be introduced as a way to rotate a vector quantity. From there, it will be shown how to rotate tensors. Rotating tensors is different than rotating vectors, but involves the same rotation matrices.

Suppose there exists a vector \vec{v}_1 (not necessarily a principal axis) which is to be rotated about the x -axis by an angle θ to obtain a vector \vec{v}_2 . This can be done by

calculating the rotation matrix $[R_x(\theta)]$ and performing the rotation according to $\vec{v}_2 = [R_x(\theta)]\vec{v}_1$. In Cartesian coordinates, this expands to

$$\begin{bmatrix} v_{x,2} \\ v_{y,2} \\ v_{z,2} \end{bmatrix} = [R_x(\theta)] \begin{bmatrix} v_{x,1} \\ v_{y,1} \\ v_{z,1} \end{bmatrix} \quad (2.40)$$

where

$$[R_x(\theta)] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \quad (2.41)$$

Similar rotation matrices can be constructed to rotate about the y - or z -axes. The rotation matrices for rotation about the Cartesian axes and illustrations of the rotations they perform are provided in Figure 2.1.

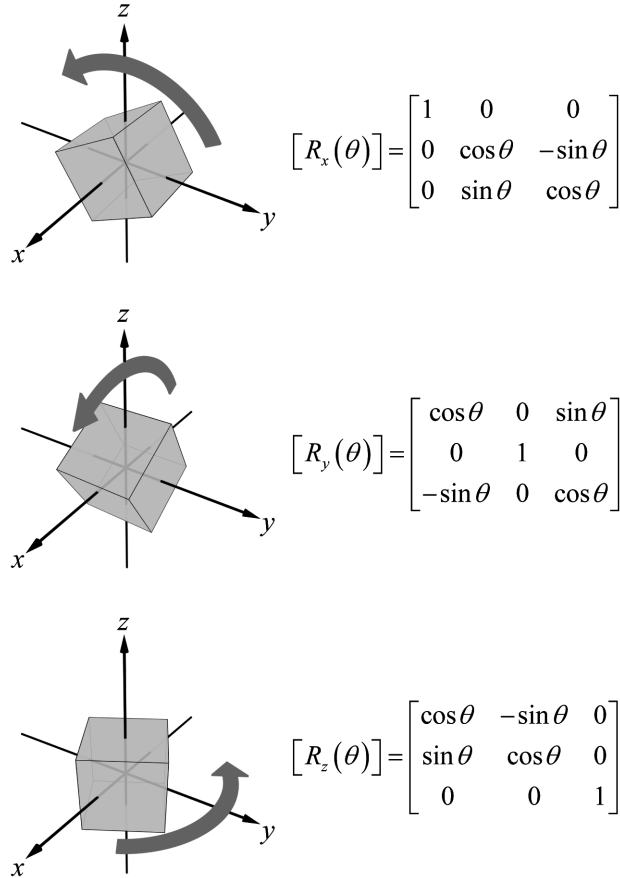


Figure 2.1 Rotation matrices that rotate about the Cartesian axes by angle θ .

It is possible to perform multiple rotations. Suppose it is desired to first rotate \vec{v}_1 about the y -axis by angle θ_1 and second rotate about the z -axis by angle θ_2 . This is accomplished using two rotation matrices as follows.

$$\begin{bmatrix} v_{x,2} \\ v_{y,2} \\ v_{z,2} \end{bmatrix} = [R_z(\theta_2)][R_y(\theta_1)] \begin{bmatrix} v_{x,1} \\ v_{y,1} \\ v_{z,1} \end{bmatrix} \quad (2.42)$$

Observe the order of the rotation matrices in (2.42) that may appear at first glance to be backward. However, it is $[R_y(\theta_1)]$ that operates on \vec{v}_1 first and then $[R_z(\theta_2)]$ operates on the result second. It is further possible to calculate a *composite rotation matrix* $[R]$ that performs both rotations in the specified order with a single matrix according to $\vec{v}_2 = [R]\vec{v}_1$. This expands to

$$\begin{bmatrix} v_{x,2} \\ v_{y,2} \\ v_{z,2} \end{bmatrix} = [R] \begin{bmatrix} v_{x,1} \\ v_{y,1} \\ v_{z,1} \end{bmatrix} \quad (2.43)$$

$$[R] = [R_z(\theta_2)][R_y(\theta_1)] \quad (2.44)$$

Now that rotating vectors using rotation matrices is understood, rotating tensors can be discussed. Both operations utilize the same rotation matrices and even the same composite rotation matrices. The tensor $[\epsilon_r]$ is rotated about the x -axis by angle θ according to

$$[\epsilon'_r] = [R_x(\theta)][\epsilon_r][R_x(\theta)]^{-1} \quad (2.45)$$

Rotating tensors is mathematically different than rotating vectors because it is actually a transform being performed instead of a simple rotation [9]. This is the reason for the appearance of the inverse in (2.45).

Now suppose the tensor is to be rotated first about the y -axis by angle θ_1 and rotated second about the z -axis by angle θ_2 . This is accomplished according to

$$[\epsilon'_r] = [R_z(\theta_2)][R_y(\theta_1)][\epsilon_r][R_y(\theta_1)]^{-1}[R_z(\theta_2)]^{-1} \quad (2.46)$$

Using the composite rotation matrix in (2.44), the tensor is rotated according to

$$[\epsilon'_r] = [R][\epsilon_r][R]^{-1} \quad (2.47)$$

It is often convenient in FDFD to define the principal values of tensors and the angles that are to be rotated in the dashboard. When the device is built onto the grid, the full tensors can be calculated by rotation during the build process.

2.3 Expansion of Maxwell's Curl Equations in Cartesian Coordinates

Maxwell's curl equations, as presented in (2.16) and (2.21), are vector equations and independent of the coordinate system. It is possible to expand the vector terms in these equations into their Cartesian components. While other coordinate systems for FDFD are possible and very useful in special cases [10–13], Cartesian coordinates will be used throughout this book. Using Cartesian coordinates, the vectors in the curl equations expand to

$$\nabla \times (E_x \hat{a}_x + E_y \hat{a}_y + E_z \hat{a}_z) = -j\omega[\mu](H_x \hat{a}_x + H_y \hat{a}_y + H_z \hat{a}_z) \quad (2.48)$$

$$\nabla \times (H_x \hat{a}_x + H_y \hat{a}_y + H_z \hat{a}_z) = j\omega[\epsilon](E_x \hat{a}_x + E_y \hat{a}_y + E_z \hat{a}_z) \quad (2.49)$$

Next, on the left-hand side of the equations the curl is calculated, and on the right-hand side the tensor quantities $[\mu]$ and $[\epsilon]$ are assumed to be diagonally anisotropic. This converts the curl equations to

$$\begin{aligned} & \left(\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \right) \hat{a}_x + \left(\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} \right) \hat{a}_y + \left(\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \right) \hat{a}_z \\ &= -j\omega\mu_{xx} H_x \hat{a}_x - j\omega\mu_{yy} H_y \hat{a}_y - j\omega\mu_{zz} H_z \hat{a}_z \end{aligned} \quad (2.50)$$

$$\begin{aligned} & \left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \right) \hat{a}_x + \left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \right) \hat{a}_y + \left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \right) \hat{a}_z \\ &= j\omega\epsilon_{xx} E_x \hat{a}_x + j\omega\epsilon_{yy} E_y \hat{a}_y + j\omega\epsilon_{zz} E_z \hat{a}_z \end{aligned} \quad (2.51)$$

Equation (2.50) can be expanded into a set of three coupled partial differential equations by setting the individual vector components equal on either side of the equation.

$$\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} = -j\omega\mu_{xx} H_x \quad (2.52)$$

$$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = -j\omega\mu_{yy} H_y \quad (2.53)$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -j\omega\mu_{zz} H_z \quad (2.54)$$

Similarly, (2.51) can be expanded into a set of three coupled partial differential equations by setting the individual vector components equal on either side of this equation.

$$\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} = j\omega\epsilon_{xx} E_x \quad (2.55)$$

$$\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} = j\omega\epsilon_{yy} E_y \quad (2.56)$$

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = j\omega\epsilon_{zz}E_z \quad (2.57)$$

Equations (2.52)–(2.57) are the key equations that will be solved numerically in FDFD.

2.4 The Electromagnetic Wave Equation

Two different electromagnetic wave equations can be derived from Maxwell's curl equations, one in terms of just the electric field \vec{E} and the other just in terms of the magnetic field \vec{H} . To derive a wave equation solely in terms of the electric field, (2.16) is solved for \vec{H} and that expression is substituted into (2.21) to eliminate the magnetic field term \vec{H} . Last, all terms are brought to the left side of the equation. The tilde notation on permittivity $\tilde{\epsilon}$ was dropped even though the permittivity may be complex.

$$\nabla \times \left(\frac{1}{\mu} \nabla \times \vec{E} \right) - \omega^2 \epsilon \vec{E} = 0 \quad (2.58)$$

This equation cannot be simplified further because the permeability μ may be inhomogeneous. That is, μ may vary with position making it so the term cannot be brought outside of the curl operation that contains spatial derivatives. If the medium is homogeneous, both μ and ϵ become constants and μ can be moved outside of the curl operation and (2.58) becomes

$$\nabla \times (\nabla \times \vec{E}) - \omega^2 \mu \epsilon \vec{E} = 0 \quad (2.59)$$

The double-curl operation is rewritten using the general vector identity $\nabla \times (\nabla \times \vec{A}) = \nabla(\nabla \cdot \vec{A}) - \nabla^2 \vec{A}$. This lets (2.59) be written as

$$\nabla(\nabla \cdot \vec{E}) - \nabla^2 \vec{E} - \omega^2 \mu \epsilon \vec{E} = 0 \quad (2.60)$$

From (2.14), the divergence of \vec{E} is zero in a homogeneous medium so the divergence term in (2.60) drops from the equation. This reduces (2.60) to

$$\nabla^2 \vec{E} + \omega^2 \mu \epsilon \vec{E} = 0 \quad (2.61)$$

From here, it is useful to define the *wavenumber* as $k = \omega\sqrt{\mu\epsilon}$. The wavenumber conveys wavelength λ through

$$k = \frac{2\pi}{\lambda} = \omega\sqrt{\mu\epsilon} \quad (2.62)$$

In a vacuum, the wavenumber becomes the free space wavenumber k_0 . This conveys the free space wavelength λ_0 through

$$k_0 = \frac{2\pi}{\lambda_0} = \frac{\omega}{c_0} \quad (2.63)$$

Substituting (2.62) into (2.61) gives the wave equation for waves inside of a homogeneous medium.

$$\nabla^2 \vec{E} + k^2 \vec{E} = 0 \quad (2.64)$$

Equation (2.64) can be compared to the historical wave equation that has been known since at least the 1700s [14]. Given the angular frequency ω and the velocity of the wave v , the historical wave equation is

$$\nabla^2 \psi + \left(\frac{\omega}{v} \right)^2 \psi = 0 \quad (2.65)$$

In (2.65), ψ is the wave disturbance and can be many things including height of a vibrating string, pressure in a fluid, an electric field, and more. Comparing the terms in parentheses in the two wave equations gives some additional physical meaning to the terms. Seeing that $\omega\sqrt{\mu\epsilon} = \omega/v$, the velocity of an electromagnetic wave must be

$$v = \frac{1}{\sqrt{\mu\epsilon}} \quad (2.66)$$

In a vacuum, $\mu = \mu_0$ and $\epsilon = \epsilon_0$ and an equation for the speed of light c_0 can be written from (2.66).

$$c_0 = \frac{1}{\sqrt{\mu_0\epsilon_0}} = 299,792,458 \text{ m/s} \quad (2.67)$$

Next, the term inside of the square root in (2.66) can be expanded into $\sqrt{\mu\epsilon} = \sqrt{\mu_0\epsilon_0} \sqrt{\mu_r\epsilon_r}$. After letting $n = \sqrt{\mu_r\epsilon_r}$, (2.66) can be written as

$$v = \frac{c_0}{n} \quad (2.68)$$

$$n = \sqrt{\mu_r\epsilon_r} \quad (2.69)$$

The parameter n is called the *refractive index*. From (2.68), it is interpreted as the factor by which a wave slows down inside of a medium relative to the speed of light in a vacuum c_0 . Since both μ_r and ϵ_r can be complex quantities, the refractive index can also be complex. Using the negative sign convention, it is usually written as

$$n = n_o - j\kappa \quad (2.70)$$

The real part of the refractive index n_o is called the *ordinary refractive index* and quantifies the speed of the wave according to $v = c_0/n_o$. The imaginary part of refractive index κ is called the *extinction coefficient* and it quantifies the attenuation

of a wave. Waves decay with distance z according to $\exp(-jk_0\kappa z)$. The product $k_0\kappa$ is called the *attenuation coefficient* α of a wave.

2.5 Electromagnetic Waves in LHI Media

The general solution to (2.64) in a linear, homogenous, and isotropic (LHI) medium is

$$\vec{E} = \vec{E}_0^+ e^{-j\vec{k}\cdot\vec{r}} + \vec{E}_0^- e^{j\vec{k}\cdot\vec{r}} \quad (2.71)$$

where \vec{E}_0^+ is the complex vector amplitude of the wave traveling in the positive \vec{k} direction and \vec{E}_0^- is the complex vector amplitude of the wave traveling in the negative \vec{k} direction. The amplitudes are complex quantities because waves have both a magnitude and a phase. In this equation, the vector position is $\vec{r} = x\hat{a}_x + y\hat{a}_y + z\hat{a}_z$. It will be useful to focus attention only on a single term in (2.71). For convenience, the forward wave will be chosen.

$$\vec{E} = \vec{P} e^{-j\vec{k}\cdot\vec{r}} \quad (2.72)$$

The vector amplitude \vec{E}_0^+ is replaced with \vec{P} to convey that this term describes the polarization and complex amplitude of the wave. The wave vector \vec{k} conveys two pieces of information at the same time. First, the direction of \vec{k} is the direction that the phase of the wave advances. That is, \vec{k} is the direction that the ripples of the wave will move as illustrated in Figure 2.2. Second, the magnitude of \vec{k} conveys the wavelength λ of the wave through

$$|\vec{k}| = \frac{2\pi}{\lambda} \quad (2.73)$$

When the frequency f is known, the free space wavelength λ_0 is also known because they are related through $c_0 = f\lambda_0$. In this case, the magnitude of \vec{k} conveys refractive index n through

$$|\vec{k}| = k_0 n \quad (2.74)$$

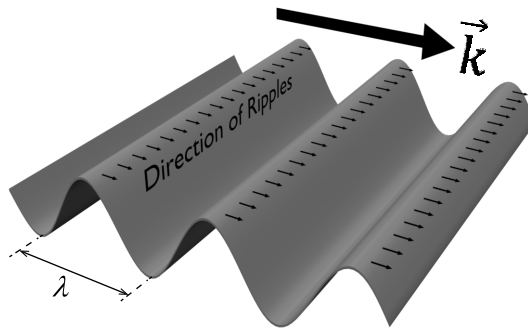


Figure 2.2 Relation between the wave vector \vec{k} , wavelength λ , and the direction that the ripples of the wave move.

An expression for the magnetic field component of the wave is found by substituting (2.72) into (2.16) and solving for \vec{H} .

$$\vec{H} = \frac{\vec{k} \times \vec{P}}{\omega\mu} e^{-j\vec{k} \cdot \vec{r}} \quad (2.75)$$

The expression for \vec{H} can be written in a more meaningful way by recognizing that the wave vector can be written as $\vec{k} = \omega\sqrt{\mu\epsilon}\hat{k}$, where \hat{k} is a unit vector in the direction of \vec{k} , and the impedance of the medium is $\eta = \sqrt{\mu/\epsilon}$.

$$\vec{H} = \frac{\hat{k} \times \vec{P}}{\eta} e^{-j\vec{k} \cdot \vec{r}} \quad (2.76)$$

The *impedance* of the medium η relates the complex amplitude of the electric field E_0 and the complex amplitude of the magnetic field H_0 of the wave. The impedance can be a complex number because both the amplitude and phase of E_0 and H_0 can be different.

$$\eta = \frac{E_0}{H_0} \quad (2.77)$$

In vacuum, the free space impedance is $\eta_0 = 376.730313668 \, \Omega$, meaning that the electric and magnetic field components are numerically around three orders of magnitude different. This will have implications in how Maxwell's equations are solved so that numerical error is minimized. The cross product in (2.76) shows that the magnetic field will be perpendicular to both the electric field and the direction of the wave. In fact, \vec{E} , \vec{H} , and \vec{k} are all perpendicular to each other and follow a *right-hand rule* such that the cross product $\vec{E} \times \vec{H}$ is in the direction of \vec{k} . When the medium is not linear, not homogeneous or not isotropic, these vector quantities are not necessarily perpendicular and very interesting behavior can happen!

All of the above could have been performed in terms of the magnetic field from which the electric field component can be calculated. As mentioned above, these equations are only valid for waves in LHI media. Let \vec{M} be the amplitude and direction of the magnetic field, and the above analysis performed in terms of the magnetic field gives

$$\nabla^2 \vec{H} + k^2 \vec{H} = 0 \quad (2.78)$$

$$\vec{H} = \vec{M} e^{-j\vec{k} \cdot \vec{r}} \quad (2.79)$$

$$\vec{E} = \eta (\vec{M} \times \hat{k}) e^{-j\vec{k} \cdot \vec{r}} \quad (2.80)$$

2.5.1 Wave Polarization

The *polarization* of a wave is defined as the time-varying direction and amplitude of the electric field component of an electromagnetic wave [2, 15]. It is not necessary to consider the magnetic field component when determining polarization because it would lead to the same conclusion. There is no extra information contained in

the magnetic field that is not conveyed through the electric field. Polarization first appeared in (2.72) as the vector \vec{P} . Accounting for polarization is very important in electromagnetic simulations because waves interact with structures differently depending on the orientation of the fields. This is particularly important when the structure being simulated is on the same physical scale as the wavelength.

An electromagnetic wave is said to have *linear polarization* (LP) when the electric field oscillates in a single plane. An LP wave is illustrated in Figure 2.3(a). A wave propagating in the $+z$ -direction that is linearly polarized in the x -direction is written as

$$\vec{E}(z) = (E_0 \hat{a}_x) \exp(-jkz) \quad (2.81)$$

The expression in parentheses is the polarization vector $\vec{P} = E_0 \hat{a}_x$. The term E_0 is the complex amplitude of the wave. It is a complex number because it affects both the magnitude and phase of the wave. A wave propagating in the $+z$ -direction that is linearly polarized in the y -direction is written as

$$\vec{E}(z) = (E_0 \hat{a}_y) \exp(-jkz) \quad (2.82)$$

The electric field for an LP wave is not restricted to being just in the x - or y -directions. The electric field can oscillate at any angle θ in the xy plane and still be linearly polarized. The expression for such a wave is

$$\vec{E}(z) = E_0 (\cos\theta \hat{a}_x + \sin\theta \hat{a}_y) \exp(-jkz) \quad (2.83)$$

An electromagnetic wave is said to have *circular polarization* (CP) when the direction of the electric field rotates with constant magnitude in a plane perpendicular to the direction of the wave. A wave propagating in the $+z$ -direction that is circularly polarized can be written as

$$\vec{E}(z) = E_0 (\hat{a}_x \pm j \hat{a}_y) \exp(-jkz) \quad (2.84)$$

Observe that a CP wave is nothing more than two LP waves propagating in the same direction which are 90° out of phase. When the sign of the imaginary component in (2.84) is negative $-$, the wave is said to have *left circular polarization* (LCP) because the electric field rotates counterclockwise when viewed from behind. An LCP wave is illustrated in Figure 2.3(b). When the sign in (2.84) is positive $+$, the wave is said to have *right circular polarization* (RCP) because the electric field rotates clockwise when viewed from behind. An RCP wave is illustrated in Figure 2.3(c). Any polarization that is not LP or CP is said to have *elliptical polarization* (EP) because the electric field will rotate in a way that traces an ellipse when viewed from behind. An EP wave is illustrated in Figure 2.3(d).

From the above discussion, a general equation can be written that easily classifies all polarization types for an arbitrary wave propagating in the direction of \vec{k} . This is

$$\vec{E}(z) = (E_1 \hat{a}_1 + E_2 e^{j\delta} \hat{a}_2) e^{j\theta} \exp(-j\vec{k} \cdot \vec{r}) \quad (2.85)$$

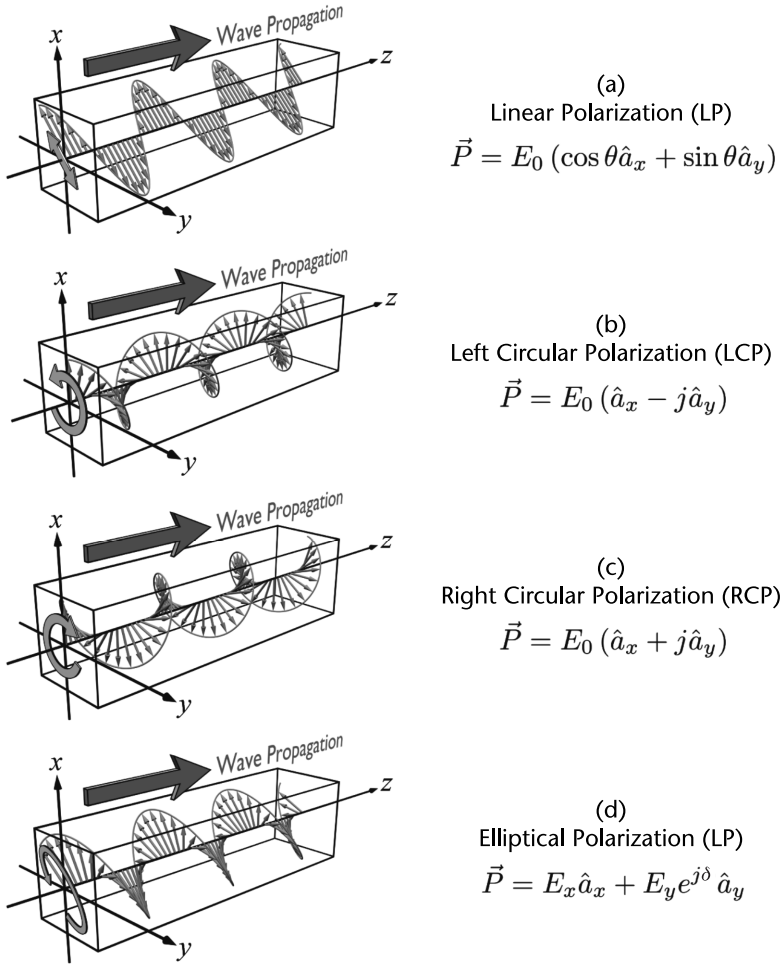


Figure 2.3 Summary of electromagnetic wave polarizations. (a) Linear polarization where the electric field oscillates within a plane. (b) Left circular polarization where electric field rotates counterclockwise from behind. (c) Right circular polarization where electric field rotates clockwise from behind. (d) Elliptical polarization where electric field rotation traces an ellipse.

\hat{a}_1 and \hat{a}_2 are unit vectors perpendicular to \vec{k} and satisfy the handedness defined by $\hat{a}_1 \times \hat{a}_2 = \vec{k}/|\vec{k}|$. Otherwise, the direction of \hat{a}_1 and \hat{a}_2 do not matter when specifying polarization until a device or interface is involved. E_1 is the real-valued amplitude of the electric field component in the direction of \hat{a}_1 . E_2 is the real-valued amplitude of the electric field component in the direction of \hat{a}_2 . θ is the phase common to both of these components and does not need to be considered when trying to determine the polarization. δ is the phase difference between the two components of the electric field. If $\delta = 0^\circ$, the polarization is linear no matter the values of E_1 and E_2 . If $\delta = \pm 90^\circ$ and $E_1 = E_2$, the polarization is circular. A phase of $\delta = +90^\circ$ indicates RCP and a phase of $\delta = -90^\circ$ indicates LCP. Anything else is considered elliptical polarization. In some sense, all polarizations are elliptical. Linear and circular polarizations are just special cases of elliptical polarization.

2.6 The Dispersion Relation for LHI Media

Only a limited set of choices for the wave vector \vec{k} are possible given the frequency ω of the wave and the electromagnetic properties of the medium. The rule that the wave vector must follow is called the *dispersion relation*. It is derived by substituting the expression for a plane wave into the wave equation and simplifying to get

$$k_x^2 + k_y^2 + k_z^2 = k^2 \quad (2.86)$$

The wavenumber k can be expanded two different ways. Both are given on the right side of (2.87) below.

$$k_x^2 + k_y^2 + k_z^2 = (k_0 n)^2 = \left(\frac{\omega n}{c_0} \right)^2 \quad (2.87)$$

Equation (2.87) is the dispersion relation for an LHI medium. It states that the magnitude of the wave vector in an LHI medium is constant regardless of the direction of the wave. The dispersion relation is used in FDFD to calculate a missing wave vector component when the others are known.

2.7 Scattering at an Interface

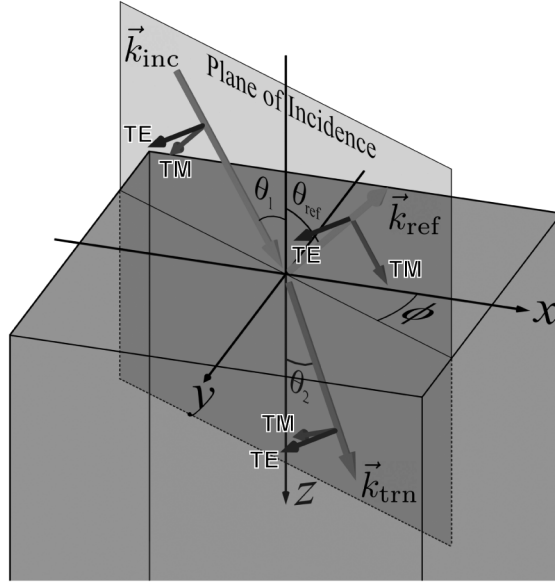
It is useful to understand what happens when a wave propagating in one medium encounters a second medium. This is not only needed to understand FDFD, but it provides ways to test FDFD to ensure it is working correctly. It will be assumed that the interface between the two mediums is perfectly flat and of an infinite extent. Figure 2.4 illustrates the geometry for two different configurations where \vec{k}_{inc} is the wave vector of the incident wave, \vec{k}_{ref} is the wave vector of the reflected wave, and \vec{k}_{trn} is the wave vector of the transmitted wave. Given the angles θ_1 , θ_2 , and ϕ , the wave vectors in the figure are written as

$$\begin{aligned} \vec{k}_{\text{inc}} &= k_0 n_1 (\sin \theta_1 \cos \phi \hat{a}_x + \sin \theta_1 \sin \phi \hat{a}_y + \cos \theta_1 \hat{a}_z) \\ \vec{k}_{\text{ref}} &= k_0 n_1 (\sin \theta_1 \cos \phi \hat{a}_x + \sin \theta_1 \sin \phi \hat{a}_y - \cos \theta_1 \hat{a}_z) \\ \vec{k}_{\text{trn}} &= k_0 n_2 (\sin \theta_2 \cos \phi \hat{a}_x + \sin \theta_2 \sin \phi \hat{a}_y + \cos \theta_2 \hat{a}_z) \end{aligned} \quad (2.88)$$

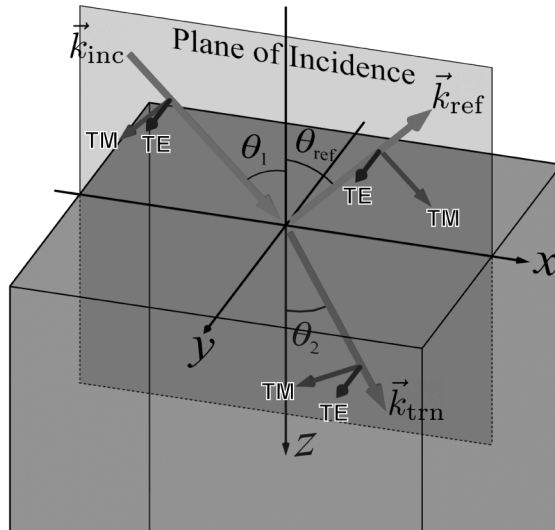
The plane defined by the incident wave vector and the z -axis is called the *plane of incidence* (POI). The wave vectors for the incident wave, reflected wave, and transmitted wave all lie in the POI. Some waves will have the electric field completely perpendicular to the POI. This is called the *transverse electric* (TE) polarization because the electric field is transverse to the POI. The TE polarization will have the magnetic field parallel to the POI. Other waves will have the electric field completely parallel to the POI. This is called the *transverse magnetic* (TM) polarization because the magnetic field will be completely transverse to the POI. The angle ϕ can be set to 0° to put the POI in the xz plane without affecting the angles, reflection

or transmission, as illustrated in Figure 2.4(b). For this reason, scattering at an interface, and later diffraction, will be analyzed for waves parallel to the xz plane.

Given the surface normal \hat{a}_z and the incident wave vector \vec{k}_{inc} , unit vectors in the direction of the TE and TM polarizations can be calculated. This is needed when calculating sources for three-dimensional simulations. The direction of the TE polarization \hat{a}_{TE} is perpendicular to the POI that is defined by the vectors \hat{a}_z and \vec{k}_{inc} .



(a) Scattering at an interface with $\phi \neq 0^\circ$



(b) Scattering at an interface with $\phi = 0^\circ$

Figure 2.4 (a) Diagram of scattering at an interface with azimuthal angle $\phi \neq 0^\circ$. (b) Diagram of scattering at an interface with azimuthal angle $\phi = 0^\circ$. The angles and amplitudes remain constant regardless of azimuthal angle ϕ .

Therefore, the direction \hat{a}_{TE} must be in the same direction as the cross product $\hat{a}_z \times \vec{k}_{\text{inc}}$. However, this cross product has no meaning at normal incidence when \hat{a}_z and \vec{k}_{inc} are parallel. This special case must be handled when calculating \hat{a}_{TE} . At normal incidence, the direction of the TE polarization will be chosen to be in the y -direction. Putting all of this together, the direction of the TE polarization is calculated according to

$$\hat{a}_{\text{TE}} = \begin{cases} \frac{\hat{a}_z \times \vec{k}_{\text{inc}}}{|\hat{a}_z \times \vec{k}_{\text{inc}}|} & \theta_1 \neq 0^\circ \\ \hat{a}_y & \theta_1 = 0^\circ \end{cases} \quad (2.89)$$

After the direction of the TE polarization is known, the direction of the TM polarization is easily calculated from it. The TM polarization must be perpendicular to both the TE polarization and \vec{k}_{inc} so it is in the direction of the cross product $\vec{k}_{\text{inc}} \times \hat{a}_{\text{TE}}$.

$$\hat{a}_{\text{TM}} = \vec{k}_{\text{inc}} \times \hat{a}_{\text{TE}} \quad (2.90)$$

Given unit vectors in the TE and TM polarization directions, the polarization vector \vec{P} for the electric field is easily written given the complex amplitude of the TE polarization P_{TE} and the complex amplitude of the TM polarization P_{TM} .

$$\vec{P} = p_{\text{TE}} \hat{a}_{\text{TE}} + p_{\text{TM}} \hat{a}_{\text{TM}} \quad (2.91)$$

This equation for polarization is a manifestation of (2.85), but here a device is involved so the directions of the unit vectors are important. Without a device or interface defined, it does not make sense to talk about TE and TM polarizations.

When the incident wave encounters the interface, some of the incident wave can reflect. The angle of the reflected wave θ_{ref} is equal to the angle of the incident wave θ_1 . This is called Snell's law of reflection [16].

$$\theta_{\text{ref}} = \theta_1 \quad (2.92)$$

In addition, some of the incident wave can transmit through the interface into the second medium. If the wave changes speed, the angle of the transmitted wave θ_2 may be different from the angle of the incident wave θ_1 in order to keep the field continuous across the interface. The angles θ_1 and θ_2 will be different when the wave is incident at some angle $\theta_1 \neq 0^\circ$ and when the first and second mediums have different refractive indices, $n_1 \neq n_2$. Snell's law of refraction [16] relates the angle of the incident wave θ_1 , angle of the transmitted wave θ_2 , and refractive indices on both sides of the interface n_1 and n_2 .

$$n_1 \sin \theta_1 = n_2 \sin \theta_2 \quad (2.93)$$

Snell's laws of reflection and refraction calculate the angles of the waves but not the amplitudes. To calculate how much of the incident wave is reflected and transmitted, the Fresnel equations are needed [2]. The *Fresnel equations* relate the amplitudes of the incident $E_{0,\text{inc}}$, reflected $E_{0,\text{ref}}$, and transmitted waves $E_{0,\text{trn}}$ given

the angles θ_1 and θ_2 and the impedances η_1 and η_2 of the two mediums. They are derived by enforcing the boundary conditions for both the electric and magnetic fields at the interface.

$$r_{\text{TE}} = \frac{E_{0,\text{ref}}^{\text{TE}}}{E_{0,\text{inc}}^{\text{TE}}} = \frac{\eta_2 \cos \theta_1 - \eta_1 \cos \theta_2}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2} \quad t_{\text{TE}} = \frac{E_{0,\text{trn}}^{\text{TE}}}{E_{0,\text{inc}}^{\text{TE}}} = \frac{2\eta_2 \cos \theta_1}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2} \quad (2.94)$$

$$r_{\text{TM}} = \frac{E_{0,\text{ref}}^{\text{TM}}}{E_{0,\text{inc}}^{\text{TM}}} = \frac{\eta_2 \cos \theta_2 - \eta_1 \cos \theta_1}{\eta_2 \cos \theta_2 + \eta_1 \cos \theta_1} \quad t_{\text{TM}} = \frac{E_{0,\text{trn}}^{\text{TM}}}{E_{0,\text{inc}}^{\text{TM}}} = \frac{2\eta_2 \cos \theta_1}{\eta_2 \cos \theta_2 + \eta_1 \cos \theta_1} \quad (2.95)$$

In these equations, the reflection and transmission properties are different for the TE and TM polarizations because each has to satisfy different boundary conditions at the interface.

2.7.1 Reflectance and Transmittance

Sometimes it is more meaningful to calculate the fraction of power that is scattered from an interface instead of the wave amplitudes. In electromagnetics, RMS power flow is quantified through the Poynting vector $\vec{\phi}$ as [1, 2]

$$\vec{\phi} = \frac{1}{2} \text{Re}[\vec{E} \times \vec{H}^*] \quad (2.96)$$

The $*$ superscript in the H^* term represents a complex conjugate operation. It is known that \vec{E} and \vec{H}^* are perpendicular to each other so the magnitude of their cross product $\vec{E} \times \vec{H}^*$ is $|\vec{E}||\vec{H}^*|$. It is also known that \vec{E} , \vec{H} , and \vec{k} form a right-handed system where the direction of $\vec{E} \times \vec{H}^*$ has to be in the same direction as \vec{k} . Therefore, the direction of the cross product is $\vec{k}/|\vec{k}|$. Combining the magnitude and direction of the cross product as described above lets the Poynting vector be written as

$$\vec{\phi} = \frac{1}{2} \text{Re} \left[|\vec{E}| |\vec{H}| \frac{\vec{k}}{|\vec{k}|} \right] \quad (2.97)$$

Next, the magnitudes $|\vec{E}|$ and $|\vec{H}|$ are related through the impedance η of the medium according to (2.77). This leads to $|\vec{H}| = |\vec{E}|/\eta$ and the Poynting vector can be written completely in terms of the electric field.

$$\vec{\phi} = \frac{1}{2} \text{Re} \left[|\vec{E}| \frac{|\vec{E}|}{\eta} \frac{\vec{k}}{|\vec{k}|} \right] \quad (2.98)$$

It is only the normal component of the Poynting vector that contributes to power flow to and from the interface. Components of the Poynting vector that are parallel to the interface can be ignored because they describe the power that stays confined at the interface. In the present analysis, z is the normal direction so it is only the z component of the Poynting vector that needs to be considered.

$$\wp_z = \frac{1}{2} \operatorname{Re} \left[\frac{|\vec{E}|^2}{\eta} \frac{k_z}{|\vec{k}|} \right] \quad (2.99)$$

Reflectance R is the fraction of power from the incident wave that is reflected from the interface. The transmittance T is the fraction of power from the incident wave that is transmitted through the interface. From this definition, reflectance and transmittance are calculated from the z components of the Poynting vectors of the incident, reflected, and transmitted waves as

$$R = \frac{\wp_{z,\text{ref}}}{\wp_{z,\text{inc}}} = \frac{\frac{1}{2} \operatorname{Re} \left[\frac{|\vec{E}_{0,\text{ref}}|^2}{\eta_1} \frac{k_{z,\text{ref}}}{|\vec{k}_{\text{ref}}|} \right]}{\frac{1}{2} \operatorname{Re} \left[\frac{|\vec{E}_{0,\text{inc}}|^2}{\eta_1} \frac{k_{z,\text{inc}}}{|\vec{k}_{\text{inc}}|} \right]} \quad (2.100)$$

$$T = \frac{\wp_{z,\text{trn}}}{\wp_{z,\text{inc}}} = \frac{\frac{1}{2} \operatorname{Re} \left[\frac{|\vec{E}_{0,\text{trn}}|^2}{\eta_2} \frac{k_{z,\text{trn}}}{|\vec{k}_{\text{trn}}|} \right]}{\frac{1}{2} \operatorname{Re} \left[\frac{|\vec{E}_{0,\text{inc}}|^2}{\eta_1} \frac{k_{z,\text{inc}}}{|\vec{k}_{\text{inc}}|} \right]} \quad (2.101)$$

Observe the negative sign in the numerator of (2.100). The reflected wave is propagating in the $-z$ -direction so $k_{z,\text{ref}}$ is negative. The negative sign is incorporated in the equation to get an overall positive number.

From (2.88), it follows that $|\vec{k}_{\text{inc}}| = |\vec{k}_{\text{ref}}| = k_0 n_1$, $|\vec{k}_{\text{trn}}| = k_0 n_2$, $k_{z,\text{inc}} = -k_{z,\text{ref}} = k_0 n_1 \cos \theta_1$, and $k_{z,\text{trn}} = k_0 n_2 \cos \theta_2$. Given these relations, (2.100) and (2.101) reduce to the following for lossless media.

$$R = \frac{|\vec{E}_{0,\text{ref}}|^2}{|\vec{E}_{0,\text{inc}}|^2} = |r|^2 \quad (2.102)$$

$$T = \frac{|\vec{E}_{0,\text{trn}}|^2}{|\vec{E}_{0,\text{inc}}|^2} \frac{\eta_1 \cos \theta_2}{\eta_2 \cos \theta_1} = |t|^2 \frac{\eta_1 \cos \theta_2}{\eta_2 \cos \theta_1} \quad (2.103)$$

Writing these equations for both the TE and TM polarizations gives

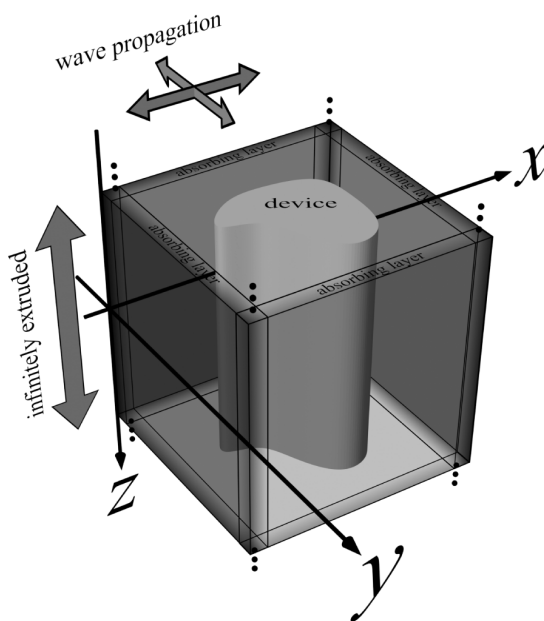
$$R_{\text{TE}} = |r_{\text{TE}}|^2 \quad T_{\text{TE}} = |t_{\text{TE}}|^2 \frac{\eta_1 \cos \theta_2}{\eta_2 \cos \theta_1} \quad (2.104)$$

$$R_{\text{TM}} = |r_{\text{TM}}|^2 \quad T_{\text{TM}} = |t_{\text{TM}}|^2 \frac{\eta_1 \cos \theta_2}{\eta_2 \cos \theta_1} \quad (2.105)$$

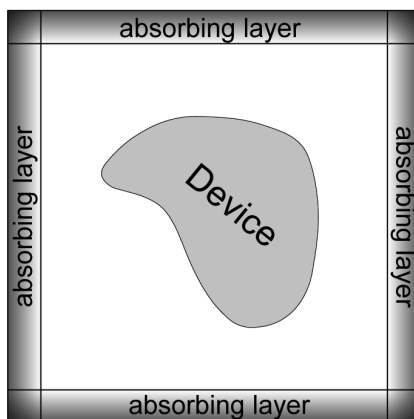
Conservation of power requires that these parameters are related through

$$R_{\text{TE}} + T_{\text{TE}} = 1 \quad (2.106)$$

$$R_{\text{TM}} + T_{\text{TM}} = 1 \quad (2.107)$$



(a) Conditions and interpretation of a two-dimensional simulation



(b) Representation as a two-dimensional simulation

Figure 2.5 (a) Illustration of the conditions for a three-dimensional simulation to numerically reduce to two dimensions. (b) Interpretation as a two-dimensional “al” simulation.

2.8 What is a Two-Dimensional Simulation?

Our world is three-dimensional, so all simulations are always three-dimensional. Sometimes, however, it is possible to reduce the math of a three-dimensional problem to just two dimensions. In electromagnetics, this happens when three conditions are met. First, the device being simulated is uniform and of infinite extent in one direction. For convenience, let the uniform direction be the z -direction. Second, wave propagation is restricted to be solely in the xy plane. Third, all materials in the simulation are isotropic, or at most diagonally anisotropic. These conditions, and the final two-dimensional interpretation, are illustrated in Figure 2.5.

When diagonally anisotropic materials are assumed, Maxwell's curl equations expand into (2.52)–(2.57). When the device is uniform in the z -direction and wave propagation is restricted to the xy plane, all of the partial derivatives taken with respect to z in these equations are zero because nothing changes in the z -direction. When the z derivatives are set to zero, (2.52)–(2.57) reduce to the following.

$$\frac{\partial E_z}{\partial y} = -j\omega\mu_{xx}H_x \quad (2.108)$$

$$-\frac{\partial E_z}{\partial x} = -j\omega\mu_{yy}H_y \quad (2.109)$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -j\omega\mu_{zz}H_z \quad (2.110)$$

$$\frac{\partial H_z}{\partial y} = j\omega\epsilon_{xx}E_x \quad (2.111)$$

$$-\frac{\partial H_z}{\partial x} = j\omega\epsilon_{yy}E_y \quad (2.112)$$

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = j\omega\epsilon_{zz}E_z \quad (2.113)$$

After inspecting these equations, it becomes apparent that they have separated into two independent sets of three coupled partial differential equations. No terms in (2.108), (2.109), and (2.113) exist in (2.110), (2.111), and (2.112). Likewise, no terms in (2.110), (2.111), and (2.112) exist in (2.108), (2.109), and (2.113). If the angle of incidence in Figure 2.4 is set to $\theta_1 = 90^\circ$, then (2.108), (2.109), and (2.113) correspond to the TM polarization while (2.110), (2.111), and (2.112) correspond to the TE polarization. For the TM polarization, it is possible to derive a single differential equation just in terms of E_z by solving (2.108) for H_x and (2.109) for H_y and then substituting these expressions into (2.113). For this reason, the TM polarization will also be called the E mode in this book. These equations are

$$H_x = -\frac{1}{j\omega\mu_{xx}}\frac{\partial E_z}{\partial y} \quad (2.114)$$

$$H_y = \frac{1}{j\omega\mu_{yy}} \frac{\partial E_z}{\partial x} \quad (2.115)$$

$$\frac{\partial}{\partial x} \left(\frac{1}{\mu_{yy}} \frac{\partial E_z}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\mu_{xx}} \frac{\partial E_z}{\partial y} \right) + \omega^2 \epsilon_{zz} E_z = 0 \quad (2.116)$$

For the TE polarization, it is possible to derive a single differential equation just in terms of H_z by solving (2.111) for E_x and (2.112) for E_y and then substituting these expressions into (2.110). For this reason, the TE polarization will also be called the H mode in this book. These equations are

$$E_x = \frac{1}{j\omega\epsilon_{xx}} \frac{\partial H_z}{\partial y} \quad (2.117)$$

$$E_y = -\frac{1}{j\omega\epsilon_{yy}} \frac{\partial H_z}{\partial x} \quad (2.118)$$

$$\frac{\partial}{\partial x} \left(\frac{1}{\epsilon_{yy}} \frac{\partial H_z}{\partial x} \right) + \frac{\partial}{\partial y} \left(\frac{1}{\epsilon_{xx}} \frac{\partial H_z}{\partial y} \right) + \omega^2 \mu_{zz} H_z = 0 \quad (2.119)$$

Solving a single differential equation instead of six coupled partial differential equations greatly reduces the computational complexity and improves efficiency. This allows simulations to be performed more quickly and larger problems to be solved. Similarly, if the problem is uniform along two dimensions, the simulation can be reduced to just one dimension for even greater computational efficiency. It is always a good practice to reduce the dimensionality of a simulation if at all possible. If a simulation is performed in something other than the xy plane, the sense of TE and TM may be different for the E and H modes.

2.9 Diffraction from Gratings

A *diffraction grating* is a planar periodic structure that splits and disperses electromagnetic waves [17]. Essentially, all periodic structures behave like diffraction gratings in terms of reflected and transmitted waves. For this reason, every periodic structure in FDFD will be analyzed in the framework of a diffraction grating even if the device is not a diffraction grating. Understanding diffraction gratings is very important for understanding how reflection and transmission from any periodic structure will be evaluated after FDFD calculates the fields. When a wave is incident onto a diffraction grating, the grating splits the reflected and transmitted waves into multiple discrete waves propagating away from the grating at different angles. These are called *diffraction orders* and the concept is illustrated in Figure 2.6. The reason for the splitting of the wave into discrete directions will be discussed in Section 2.9.1. The fraction of power coupled into a specific diffraction order from the incident wave is called the *diffraction efficiency*. The overall reflectance from a device is the sum of the diffraction efficiencies of all the reflected diffraction orders.

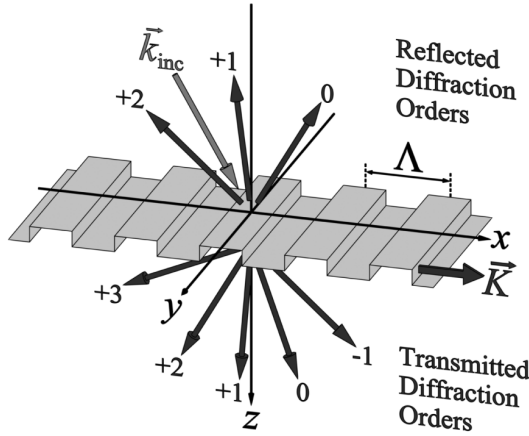


Figure 2.6 Geometry of diffraction from a grating. The grating is depicted as a paper-thin interface to generalize the configuration of the grating. The grating could be at the bottom of the top medium, on the surface of the bottom medium, or in some intermediate medium.

Similarly, the overall transmittance from a device is the sum of the diffraction efficiencies of all the transmitted diffraction orders.

2.9.1 The Grating Equation

Suppose a diffraction grating is made from a device with a periodic permittivity $\epsilon(\vec{r})$. The permittivity of the diffraction grating can be written in terms of the average permittivity ϵ_{avg} , permittivity contrast $\Delta\epsilon$, and the grating vector \vec{K} .

$$\epsilon(\vec{r}) = \epsilon_{\text{avg}} + \Delta\epsilon \cos(\vec{K} \cdot \vec{r}) \quad (2.120)$$

The permittivity contrast $\Delta\epsilon$ describes how much the permittivity function varies around the average value ϵ_{avg} . The grating vector \vec{K} is analogous to the wave vector \vec{k} and conveys two pieces of information at the same time. First, the direction of \vec{K} is perpendicular to the grooves of the diffraction grating like \vec{k} is perpendicular to the ripples of a wave. In this sense, the grooves are analogous to wave ripples, but the grooves of a grating do not move and do not oscillate like the ripples of a wave. Second, the magnitude $|\vec{K}|$ conveys the period Λ of the grating just like the magnitude $|\vec{k}|$ conveys the wavelength of a wave.

$$|\vec{K}| = \frac{2\pi}{\Lambda} \quad (2.121)$$

When a wave passes through a diffraction grating, its amplitude gets perturbed by the grating with the same basic pattern as the grating. If the applied wave is $\vec{E}_{\text{inc}}(\vec{r}) = \vec{E}_0 \exp(-j\vec{k}_{\text{inc}} \cdot \vec{r})$, the perturbed wave can be written as

$$\vec{E}(\vec{r}) \propto \vec{E}_0 [\epsilon_{\text{avg}} + \Delta\epsilon \cos(\vec{K} \cdot \vec{r})] \exp(-j\vec{k}_{\text{inc}} \cdot \vec{r}) \quad (2.122)$$

The proportionality sign \propto is used here because no conclusions are being made about the amplitude of the perturbed wave. Attention is focused solely on the

interaction between the grating vector \vec{K} and wave vector \vec{k}_{inc} . Applying trigonometric identities to this equation shows that (2.122) is actually three waves propagating in different directions. One wave propagates in the direction of \vec{k}_{inc} , another propagates in the direction of $\vec{k}_{\text{inc}} - \vec{K}$, and the last propagates in the direction of $\vec{k}_{\text{inc}} + \vec{K}$.

$$\begin{aligned} \vec{E}(\vec{r}) \propto & \vec{E}_0 \epsilon_{\text{avg}} \exp(-j\vec{k}_{\text{inc}} \cdot \vec{r}) \\ & + \vec{E}_0 \frac{\Delta\epsilon}{2} \exp[-j(\vec{k}_{\text{inc}} - \vec{K}) \cdot \vec{r}] + \vec{E}_0 \frac{\Delta\epsilon}{2} \exp[-j(\vec{k}_{\text{inc}} + \vec{K}) \cdot \vec{r}] \end{aligned} \quad (2.123)$$

Each of these three waves is also perturbed by the diffraction grating and splits into three more directions, giving a total of nine waves. Each of these nine waves is perturbed by the diffraction grating, split into three waves, and so on. This leads to an infinite sum of discrete waves.

$$\vec{E}(\vec{r}) \propto \sum_{m=-\infty}^{\infty} \vec{E}_m \exp[-j(\vec{k}_{\text{inc}} - m\vec{K}) \cdot \vec{r}] \quad (2.124)$$

To derive the *grating equation* that will calculate the angles of the diffraction orders, let the geometry of the grating be that shown in Figure 2.6. The incident wave vector is written as

$$\vec{k}_{\text{inc}} = k_0 n_{\text{inc}} \sin \theta_{\text{inc}} \hat{a}_x + k_0 n_{\text{inc}} \cos \theta_{\text{inc}} \hat{a}_y \quad (2.125)$$

After experiencing the grating, the wave vector expansion from (2.124) becomes $\vec{k}(m) = \vec{k}_{\text{inc}} - m\vec{K}$. Replacing \vec{k}_{inc} in this expression with (2.125) and letting $K = \left(\frac{2\pi}{\Lambda}\right)\hat{a}_x$ gives a new expression for $\vec{k}(m)$.

$$\vec{k}(m) = \left(k_0 n_{\text{inc}} \sin \theta_{\text{inc}} - m \frac{2\pi}{\Lambda}\right) \hat{a}_x + k_0 n_{\text{inc}} \cos \theta_{\text{inc}} \hat{a}_y \quad (2.126)$$

Boundary conditions require that the x -components of the wave vectors be the same for both the reflected and transmitted diffraction orders. For this reason, $k_x(m)$ taken from (2.126) is written without having to identify it as either reflected or transmitted.

$$k_x(m) = k_0 n_{\text{inc}} \sin \theta_{\text{inc}} - m \frac{2\pi}{\Lambda} \quad (2.127)$$

Equation (2.127) will be used in FDFD to calculate the tangential components of the wave vectors associated with the diffraction orders. The term $k_x(m)$ can be written in terms of the refractive index n_{obs} where the diffraction is being observed and the angle $\theta(m)$ of the diffraction order. Putting $k_x(m) = k_0 n_{\text{obs}} \sin[\theta(m)]$ into (2.127) gives

$$k_0 n_{\text{obs}} \sin[\theta(m)] = k_0 n_{\text{inc}} \sin \theta_{\text{inc}} - m \frac{2\pi}{\Lambda} \quad (2.128)$$

The grating equation is derived by dividing this equation by the free space wave-number $k_0 = 2\pi/\lambda_0$.

$$n_{\text{obs}} \sin[\theta(m)] = n_{\text{inc}} \sin \theta_{\text{inc}} - m \frac{\lambda_0}{\Lambda} \quad (2.129)$$

The grating equation can be thought of as a generalization to Snell's law of refraction that accounts for multiple diffraction orders. In this equation, m is the diffraction order number. The zero-order ($m = 0$) term is the only wave that would be present if no diffraction occurred and the grating equation reduces exactly to Snell's law of refraction. Observe from (2.129) that it is possible for diffraction order angles $\theta(m)$ to become imaginary due to the subtraction on the righthand side of the equation. Imaginary diffraction angles indicate that the diffraction order is cutoff and is not a propagating wave. The grating equation is useful for calculating the direction $\theta(m)$ of diffraction orders and what diffraction orders may be cutoff. Sometimes it is desired to control which diffraction orders are present in a grating. For simulations, it is useful to know when there are diffraction orders near cutoff because these cause numerical problems at the boundaries that are difficult to impossible to handle accurately. Longer grating periods produce more diffraction orders. As the grating period is increased, diffraction orders first appear propagating near parallel to the surface of the grating. As the period is increased further, the angles of the diffraction orders decrease and new diffraction orders may appear. It is very common to use the grating equation to calculate the grating period that cuts off all the diffraction orders except the zero-order. These are called *subwavelength gratings* because the period is less than one wavelength, $\Lambda < \lambda$.

2.9.2 Diffraction Efficiency

The *diffraction efficiency* of a diffraction order is the fraction of power from the applied wave that gets coupled into that diffraction order. Calculating diffraction efficiency follows how reflectance and transmittance were calculated previously. The diffraction efficiencies of the m th diffraction order for both reflection and transmission are written from (2.100) and (2.101) as

$$R_{\text{DE}}(m) = \frac{\phi_{z,\text{ref}}(m)}{\phi_{z,\text{inc}}} = \frac{\frac{1}{2} \text{Re} \left[\frac{|\vec{E}_{0,\text{ref}}(m)|^2 k_{z,\text{ref}}(m)}{\eta_{\text{ref}} |\vec{k}_{\text{ref}}|} \right]}{\frac{1}{2} \text{Re} \left[\frac{|\vec{E}_{0,\text{inc}}|^2 k_{z,\text{inc}}}{\eta_{\text{inc}} |\vec{k}_{\text{inc}}|} \right]} \quad (2.130)$$

$$T_{\text{DE}}(m) = \frac{\phi_{z,\text{trn}}(m)}{\phi_{z,\text{inc}}} = \frac{\frac{1}{2} \text{Re} \left[\frac{|\vec{E}_{0,\text{trn}}(m)|^2 k_{z,\text{trn}}(m)}{\eta_{\text{trn}} |\vec{k}_{\text{trn}}|} \right]}{\frac{1}{2} \text{Re} \left[\frac{|\vec{E}_{0,\text{inc}}|^2 k_{z,\text{inc}}}{\eta_{\text{inc}} |\vec{k}_{\text{inc}}|} \right]} \quad (2.131)$$

Recognizing that $|\vec{k}_{\text{inc}}| = |\vec{k}_{\text{ref}}| = k_0 n_{\text{ref}}$, $|\vec{k}_{\text{trn}}| = k_0 n_{\text{trn}}$, $\eta_{\text{ref}} = \sqrt{\mu_0 \mu_{\text{r,ref}} / \epsilon_0 \epsilon_{\text{r,ref}}}$, $\eta_{\text{trn}} = \sqrt{\mu_0 \mu_{\text{r,trn}} / \epsilon_0 \epsilon_{\text{r,trn}}}$, and $n_{\text{ref}} = \sqrt{\mu_{\text{r,ref}} \epsilon_{\text{r,ref}}}$, and $n_{\text{trn}} = \sqrt{\mu_{\text{r,trn}} \epsilon_{\text{r,trn}}}$, (2.130) and (2.131) reduce to the following for lossless media.

$$R_{\text{DE}}(m) = \frac{|\vec{E}_{0,\text{ref}}(m)|^2}{|\vec{E}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m)}{k_{z,\text{inc}}} \right] \quad (2.132)$$

$$T_{\text{DE}}(m) = \frac{|\vec{E}_{0,\text{trn}}(m)|^2}{|\vec{E}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\mu_{\text{r,ref}}}{\mu_{\text{r,trn}}} \frac{k_{z,\text{trn}}(m)}{k_{z,\text{inc}}} \right] \quad (2.133)$$

In FDFD, the source will almost always be given a unit amplitude so $|\vec{E}_{\text{inc}}|^2 = 1$. The amplitudes of the diffraction orders $\vec{E}_{\text{ref}}(m)$ and $\vec{E}_{\text{trn}}(m)$ will be determined from the FDFD simulation. The z components of the diffraction orders are determined from the dispersion relation in the medium where they exist.

$$k_{z,\text{ref}}(m) = -\sqrt{(k_0 n_{\text{ref}})^2 - k_x^2(m)} \quad (2.134)$$

$$k_{z,\text{trn}}(m) = \sqrt{(k_0 n_{\text{trn}})^2 - k_x^2(m)} \quad (2.135)$$

A negative sign was inserted into (2.134) because the reflected wave is propagating in the $-z$ -direction. Observe from (2.134) and (2.135) that the longitudinal components of the wave vectors can become imaginary. This means the m th diffraction order is cutoff and is not a propagating wave. These are evanescent waves and they do not contribute to power flow away from the diffraction grating. These are ignored in the diffraction efficiency equations by taking only the real part of $k_{z,\text{ref}}(m)$ and $k_{z,\text{trn}}(m)$.

Equations (2.132) and (2.133) are used when electric fields are calculated from the simulation (E mode simulation). If magnetic fields are known instead (H mode simulation), (2.136) and (2.137) should be used instead.

$$R_{\text{DE}}(m) = \frac{|\vec{H}_{0,\text{ref}}(m)|^2}{|\vec{H}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m)}{k_{z,\text{inc}}} \right] \quad (2.136)$$

$$T_{\text{DE}}(m) = \frac{|\vec{H}_{0,\text{trn}}(m)|^2}{|\vec{H}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\epsilon_{\text{r,ref}}}{\epsilon_{\text{r,trn}}} \frac{k_{z,\text{trn}}(m)}{k_{z,\text{inc}}} \right] \quad (2.137)$$

2.9.3 Generalization to Crossed Gratings

When a grating is periodic in both the x - and y -directions at the same time, it is called a *crossed grating*. Diffraction from a crossed grating is illustrated in Figure 2.7. The device has a period in the x -direction given as Λ_x and a period in the y -direction

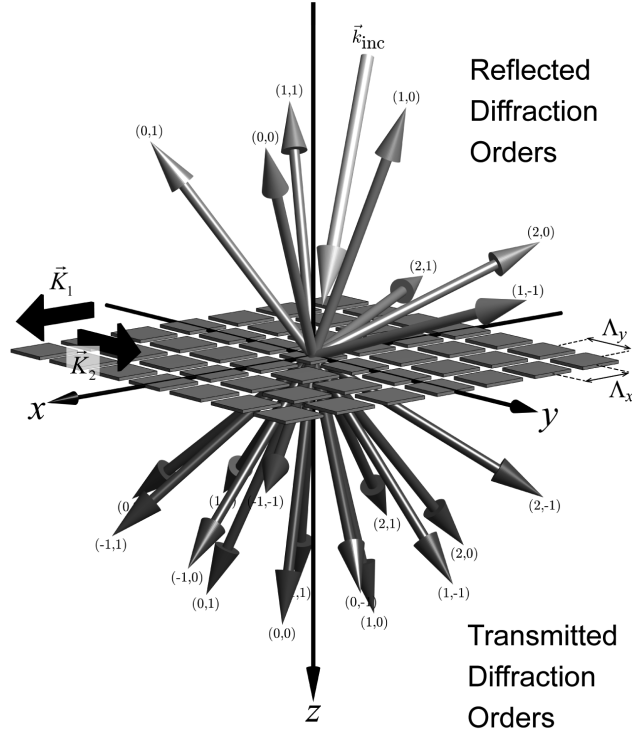


Figure 2.7 Geometry of diffraction from a crossed grating described by grating vectors $\vec{K}_1 = K_x \hat{a}_x$ and $\vec{K}_2 = K_y \hat{a}_y$. Diffraction orders occur within a cone of angles on both sides of the grating and are identified by their two indices (m, n) .

given as Λ_y . The key difference from planar diffraction discussed previously is that diffraction orders can occur in a cone of directions above and below the grating.

The period and symmetry of a crossed grating is described by two grating vectors \vec{K}_1 and \vec{K}_2 . For a rectangular grating, these are defined as

$$\vec{K}_1 = K_x \hat{a}_x \quad \vec{K}_2 = K_y \hat{a}_y \quad (2.138)$$

$$|\vec{K}_1| = K_x = \frac{2\pi}{\Lambda_x} \quad |\vec{K}_2| = K_y = \frac{2\pi}{\Lambda_y} \quad (2.139)$$

For three-dimensional diffraction problems, boundary conditions must be applied to both x and y components of the incident wave vector. This leads to an infinite expansion for both x and y components of the wave vectors associated with the diffraction orders. These are equal for both reflected and transmitted diffraction orders and are given by

$$k_x(m, n) = k_{x, \text{inc}} - m \frac{2\pi}{\Lambda_x} \quad (2.140)$$

$$k_y(m, n) = k_{y, \text{inc}} - n \frac{2\pi}{\Lambda_y} \quad (2.141)$$

Observe for crossed gratings that there are two integers m and n that identify the diffraction orders. That is because diffraction no longer occurs in just a single plane, but fans out in a cone of directions. While the wave vectors of both the reflected and transmitted diffraction orders share the same x and y components, the longitudinal components must be calculated separately because the mediums can be different. These are calculated from the dispersion relation as

$$k_{z,\text{ref}}(m, n) = -\sqrt{(k_0 n_{\text{ref}})^2 - k_x^2(m, n) - k_y^2(m, n)} \quad (2.142)$$

$$k_{z,\text{trn}}(m, n) = \sqrt{(k_0 n_{\text{trn}})^2 - k_x^2(m, n) - k_y^2(m, n)} \quad (2.143)$$

The diffraction efficiencies are calculated using essentially the same equations as for planar diffraction. The difference is that the diffraction efficiencies are calculated for diffraction orders identified by two indices m and n . When diffraction efficiency is calculated from electric fields, the equations are

$$R_{\text{DE}}(m, n) = \frac{|\vec{E}_{0,\text{ref}}(m, n)|^2}{|\vec{E}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m, n)}{k_{z,\text{inc}}} \right] \quad (2.144)$$

$$T_{\text{DE}}(m, n) = \frac{|\vec{E}_{0,\text{trn}}(m, n)|^2}{|\vec{E}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\mu_{r,\text{ref}} k_{z,\text{trn}}(m, n)}{\mu_{r,\text{trn}} k_{z,\text{inc}}} \right] \quad (2.145)$$

When diffraction efficiency is calculated from magnetic fields, the equations are

$$R_{\text{DE}}(m, n) = \frac{|\vec{H}_{0,\text{ref}}(m, n)|^2}{|\vec{H}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m, n)}{k_{z,\text{inc}}} \right] \quad (2.146)$$

$$T_{\text{DE}}(m, n) = \frac{|\vec{H}_{0,\text{trn}}(m, n)|^2}{|\vec{H}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\epsilon_{r,\text{ref}} k_{z,\text{trn}}(m, n)}{\epsilon_{r,\text{trn}} k_{z,\text{inc}}} \right] \quad (2.147)$$

2.10 Waveguides and Transmission Lines

Waveguides for electromagnetic waves are analogous to pipes for fluids [18–21]. They confine electromagnetic waves so that power will not be lost due to the spreading of the wave as it propagates. At radio frequencies, common waveguides include transmission lines like microstrips [22] and rectangular metal waveguides [2]. At optical frequencies, common waveguides include optical fibers [20] and integrated optical waveguides [23]. There exist plenty more types of waveguides including some very interesting and exotic designs, usually intended for specialized applications. Electromagnetic fields must obey Maxwell's equations so the fields cannot take on any configuration they please. Due to the rules defined by Maxwell's equations,

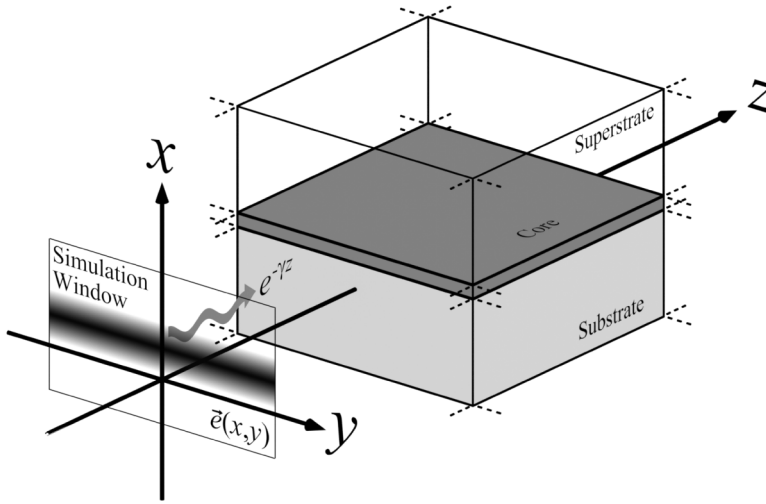
the fields can only take on certain discrete configurations called *guided modes*. A guided mode can have a *cutoff frequency*. At frequencies less than the cutoff frequency, the guided mode is said to be *cutoff* and will decay very quickly along the waveguide if any attempt is made to excite the mode. Not all guided modes have a cutoff frequency and will still be guided modes even as the frequency approaches zero. At a fixed frequency, waveguides can sometimes support multiple modes. The guided mode supported at the lowest frequency is called the *fundamental mode*. The fundamental mode is generally the most important of the guided modes. Single-mode waveguides operate at a frequency where only the fundamental mode is supported by the waveguide.

Transmission lines are waveguides in every sense, but they are a special class of waveguides that tend to be thought of more as a circuit element or a simple electrical interconnect. For a waveguide to be considered a transmission line, it must have at least two conductors. The fundamental mode in transmission lines has no cutoff frequency and can operate down to zero frequency. When a transmission line contains only a single homogeneous dielectric around the conductors, it can support a completely *transverse electromagnetic* (TEM) mode where neither the electric nor magnetic field has a vector component in the direction of propagation.

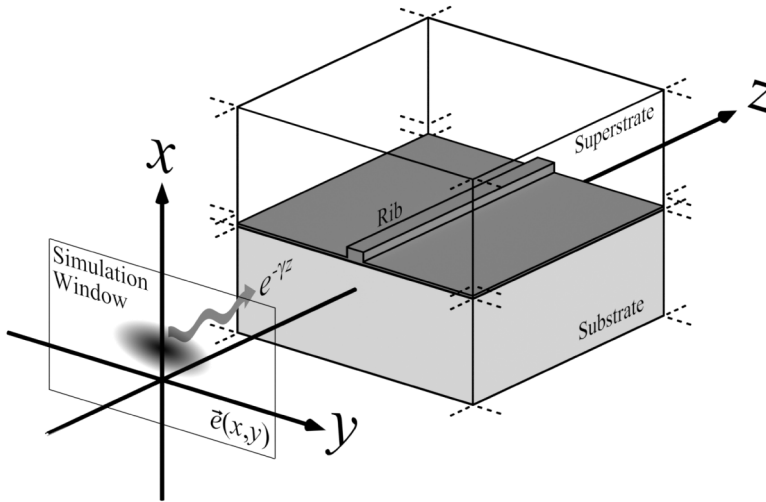
2.10.1 Waveguide Modes and Parameters

Waveguides can be divided into two broad categories, slab waveguides and channel waveguides. These are illustrated in the context of two different integrated optical waveguides in Figure 2.8. A *slab waveguide* is shown in Figure 2.8(a) and confines waves along only one axis. In this case, waves will not spread out in the x -direction as they propagate through the waveguide, but they will spread out in both the y - and z -directions because they are only confined in the x -direction. If the guided mode propagates only in the z -direction and not the y -direction, the ripples of the wave will only move in the z -direction. The field of the guided mode will extend infinitely and uniformly outward in the y -direction. The guided mode shown for the slab waveguide is something close to a Gaussian distribution along the x -axis and is uniform along the y -axis. The basic dielectric slab waveguide is composed of a high refractive index core surrounded by two mediums with a lower refractive index, but many other designs are possible. Below the core is the *substrate* medium and above the core is the *superstrate* medium. A *channel waveguide* is shown in Figure 2.8(b) and confines waves along two axes. In this case, waves are confined along both the x - and y -axes and propagate in the z -direction. The guided mode shown for the channel waveguide is a single spot due to the wave being confined in both x - and y -directions. The specific channel waveguide depicted in Figure 2.8(b) is a rib waveguide that is constructed a bit like the slab waveguide in Figure 2.8(a), but part of the high-index region is made thicker to form a rib. The rib provides index contrast in the x - and y -directions to confine waves in both the x - and y -directions.

Figure 2.8 also illustrates the basic geometry for analyzing waveguides. While only a dielectric slab and rib waveguide are shown, virtually all waveguides can be analyzed just like this. The typical analysis lets the z -direction be the direction of propagation along the waveguide. This makes the xy plane be the cross section of the waveguide. Even though the waveguide is three-dimensional, calculating



(a) Slab waveguide



(b) Channel waveguide

Figure 2.8 Comparison of slab and channel waveguides. (a) Slab waveguides confine propagation along a single axis. (b) Channel waveguides confine propagation along two axes.

guided modes in straight channel waveguides reduces to a two-dimensional problem because it is only the cross section that has to be analyzed. Calculating guided modes in slab waveguides reduces to a one-dimensional problem because the cross section exists only in a single direction.

The function describing what the electric field of a guided mode looks like in the xy plane is labeled as $\vec{e}(x,y)$ in Figure 2.8. The peak amplitude of this function has no meaning and can be set to anything. Sometimes $\vec{e}(x,y)$ is normalized so that the peak value is 1. At other times, it is normalized so that $\int_y \int_x \vec{e}(x,y) dx dy = 1$. The

function $\vec{e}(x, y)$ is simply a description of the relative amplitude and direction of the electric field throughout the xy plane at $z = 0$. To determine what a guided mode looks like at any other position z , the complex propagation constant γ must also be calculated. Given γ , the electric field at any position is written as

$$\vec{E}(x, y, z) = \vec{e}(x, y)\exp(-\gamma z) \quad (2.148)$$

The complex propagation constant γ is a complex number with real and imaginary parts defined as

$$\gamma = \alpha + j\beta \quad (2.149)$$

The terms α and β have physical meaning that can be determined by substituting (2.149) into (2.148) and expanding the exponential. This gives

$$\vec{E}(x, y, z) = \vec{e}(x, y)\exp(-\alpha z)\exp(-j\beta z) \quad (2.150)$$

From (2.150), it is observed that α describes the decay of the guided mode that would occur due to the guided mode being leaky or due to ohmic loss in the materials the waveguide is made of. The parameter α is called the *attenuation coefficient*. In most guided mode calculations, the loss is ignored and the waveguides are not leaky so $\alpha = 0$. It can also be observed that β leads to the guided mode oscillating with distance z due to the exponential having an imaginary argument. The parameter β is called the *phase constant* and describes how quickly the guided mode accumulates phase as it propagates. As long as the waveguide is straight and possesses no discontinuities, the picture of the mode $\vec{e}(x, y)$ does not change and the mode simply accumulates phase and/or decays as it propagates. In this sense, a propagating mode is quite boring!

In photonics, sometimes the effective refractive index n_{eff} is used to characterize propagation instead of the complex propagation constant γ . The two parameters are almost synonymous and are related through $\gamma = jk_0 n_{\text{eff}}$. The meaning of the *effective refractive index* is that a guided mode will accumulate phase at the same rate as a plane wave propagating in an infinite and homogeneous medium of refractive index n_{eff} . In dielectric waveguides, the effective refractive index is very close to the average refractive index calculated over the area of the guided mode. The concept of the effective refractive index can be used to reduce some complicated three-dimensional simulations down to simpler two-dimensional simulations. This will be discussed in more detail in Chapter 6 when the effective index method is discussed.

All of the information about the guided mode is contained in $\vec{e}(x, y)$ and γ (or n_{eff}). If the waveguide supports more than one guided mode, each mode will have its own pair of parameters describing it because each mode will look different and propagate differently. The m th guided mode will have $\vec{e}_m(x, y)$ and γ_m (or $n_{m,\text{eff}}$). The magnetic field can be calculated directly from the electric field and has the same basic form given in (2.151). The magnetic field component of a guided mode has the same complex propagation constant γ , but an entirely different picture $\vec{h}(x, y)$ in the xy plane.

$$\vec{H}(x, y, z) = \vec{h}(x, y) \exp(-\gamma z) \quad (2.151)$$

The purpose of a guided-mode calculation is simply to calculate $\vec{e}_m(x, y)$, γ_m (or $n_{m,\text{eff}}$), and possibly $\vec{h}_m(x, y)$ for each guided mode. The analysis only calculates what guided modes are supported by the waveguide, not what guided modes may actually be excited or their amplitudes relative to the other modes. A scattering simulation with a source is required in order to determine which of the guided modes may be excited and what the amplitudes of the modes would be. Scattering simulations with waveguides will be discussed in Chapter 8.

2.10.2 Transmission Line Parameters

There are many types of transmission lines to meet a wide array of applications [18, 24]. Two different transmission lines are illustrated in Figure 2.9. The microstrip transmission line in Figure 2.9(a) is composed of a strip of metal sitting on top of a dielectric substrate with a ground plane underneath the substrate. The coaxial transmission line in Figure 2.9(b) is composed of an inner conductor and an outer conductor with a dielectric separating the two conductors. Fortunately, almost all transmission lines can be accurately represented using the simple RLGC equivalent circuit model illustrated in Figure 2.9(c) [24]. While the equivalent circuit is shown

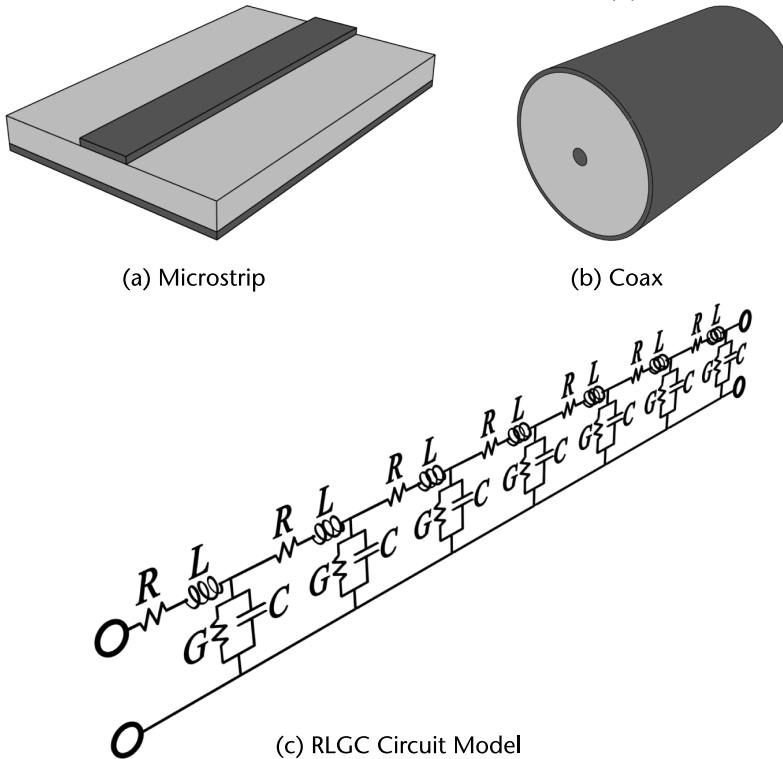


Figure 2.9 (a) Microstrip transmission line. (b) Coaxial transmission line. (c) RLGC equivalent circuit model for a transmission line.

to be composed of discrete circuit elements R , L , G and C , the elements are truly distributed parameters that manifest themselves smoothly and continuously across the line. R has units of ohms per meter (Ω/m) and is the resistance caused primarily by the resistivity of the conductors. L has units of Henries per meter (H/m) and is the inductance caused by the magnetic energy stored around the line. G has units of Siemens per meter ($1/\Omega \cdot \text{m}$) and arises due to the conductivity in the dielectric separating the conductors. C has units of Farads per meter (F/m) and is the capacitance caused by the ability to store electric energy between the lines.

Despite being distributed parameters, the transmission line can still be modeled with discrete circuit elements in the form of a series of repeated RLGC circuits, as illustrated in Figure 2.9(c). The model works when the length of the RLGC circuit is made to approach zero. From this model, the characteristic impedance Z_0 and complex propagation constant γ can be derived. Applying Kirchoff's voltage law (KVL) [25, 26] around the outer loop and applying Kirchoff's current law (KCL) [25, 26] at the center node of the RLGC circuit gives the following equations.

$$-\frac{dV(z)}{dz} = (R + j\omega L)I(z) \quad (2.152)$$

$$-\frac{dI(z)}{dz} = (G + j\omega C)V(z) \quad (2.153)$$

Equations (2.152) and (2.153) are sometimes called the *telegrapher equations* [1, 24]. They are somewhat analogous to Maxwell's curl equations in which it is the interaction between these two equations that produces waves. Combining the equations leads to the wave equation on the transmission line that can be written in terms of just the voltage $V(z)$ or just the current $I(z)$ as follows.

$$\frac{d^2V(z)}{dz^2} - (R + j\omega L)(G + j\omega C)V(z) = 0 \quad (2.154)$$

$$\frac{d^2I(z)}{dz^2} - (R + j\omega L)(G + j\omega C)I(z) = 0 \quad (2.155)$$

The collection of terms $(R + j\omega L)(G + j\omega C)$ defines the complex propagation constant γ . The general solution to both of the wave equations is

$$V(z) = V_0^+ \exp(-\gamma z) + V_0^- \exp(\gamma z) \quad (2.156)$$

$$I(z) = I_0^+ \exp(-\gamma z) + I_0^- \exp(\gamma z) \quad (2.157)$$

where

$$\gamma = \alpha + j\beta = (R + j\omega L)(G + j\omega C) \quad (2.158)$$

In these equations, V_0^+ is the complex amplitude of the voltage component of the forward wave, V_0^- is the complex amplitude of the voltage component of the backward wave, I_0^+ is the complex amplitude of the current component of the forward wave, and I_0^- is the complex amplitude of the current component of the backward

wave. The parameter α is the attenuation coefficient and β is the phase constant. Equation (2.158) can be solved for α and β to write them in terms of the RLGC parameters. The equations for α and β are almost identical and differ only by the sign of the $(RG - \omega^2 LC)$ term.

$$\alpha = \sqrt{\frac{(RG - \omega^2 LC) + \sqrt{(R^2 + \omega^2 L^2)(G^2 + \omega^2 C^2)}}{2}} \quad (2.159)$$

$$\beta = \sqrt{\frac{-(RG - \omega^2 LC) + \sqrt{(R^2 + \omega^2 L^2)(G^2 + \omega^2 C^2)}}{2}} \quad (2.160)$$

The *characteristic impedance* Z_0 quantifies the amplitude and phase relationship between the voltage and current at any point along the transmission line. An expression for Z_0 in terms of the RLGC parameters is derived by substituting (2.156) and (2.157) into (2.152) and (2.153), combining the two new equations, and then solving for V_0^+/I_0^+ . This long derivation gives

$$Z_0 = \frac{V_0^+}{I_0^+} = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \quad (2.161)$$

2.11 Scalability of Maxwell's Equations

There is no fundamental length scale in electromagnetics. This means that a 1-m wavelength interacting with a 0.5-m device will behave exactly like a 3-m wavelength interacting with the same device scaled to 1.5 m. This assumes the material properties are the same at both wavelengths. The concept of scalability is illustrated in Figure 2.10. While a schematic symbol for an antenna was used, the scaling concept applies to all electromagnetic devices.

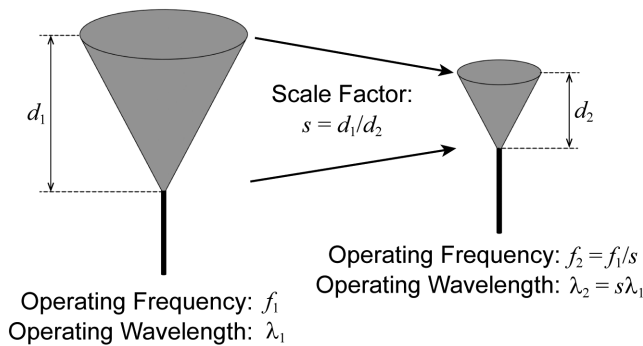


Figure 2.10 Scalability in electromagnetics.

Scalability has two big implications for electromagnetic simulations. First, the frequency, wavelength, and dimensions of a device are artificial concepts to an electromagnetic simulation. These quantities may be specified at the start of a simulation, but they are quickly normalized so the only parameters known to the simulation are relative permittivity, relative permeability, and size relative to wavelength. Each of these are unitless quantities that carry no information about the actual frequency, wavelength, or physical size of the device being simulated. When the simulation is finished, the results are often denormalized back to whatever frequency or physical dimensions were initially specified, but these are not part of the simulation itself. Second, if a device is simulated, or if a device is identified in the literature, which exhibits the desired behavior but at the wrong frequency, it is only a matter of scaling the dimensions to get the same behavior at a different frequency.

2.12 Numerical Solution to Maxwell's Equations

The numerical solution to Maxwell's equations will be obtained using the finite-difference method. The finite-difference method will convert Maxwell's equations to a single matrix equation that can be solved numerically. The matrix equation can have one of two forms: (1) an eigenvalue problem and (2) a scattering problem. For eigenvalue problems, the matrix equation will have the form of

$$\mathbf{A}\mathbf{f} = \nu\mathbf{f} \quad (2.162)$$

In (2.162), the matrix \mathbf{A} enforces Maxwell's equations onto the fields stored in the column vector \mathbf{f} . The eigenvalue problem is essentially a test. If performing the operation of \mathbf{A} on the function stored in \mathbf{f} gives the function \mathbf{f} again scaled by a constant ν , then the function in \mathbf{f} is a valid solution to the eigenvalue problem. The solution \mathbf{f} is an eigenvector and the constant ν is an eigenvalue. Eigenvectors and eigenvalues come in pairs and should always be kept together. When solved numerically, M solutions to the eigenvalue problem are calculated when the matrix \mathbf{A} is size $M \times M$. There is no source or excitation associated with eigenvalue problems. When analyzing waveguides, for example, the solutions are the possible modes supported by the waveguide. Eigenvalue problems only calculate what modes could propagate in the waveguide. Calculating what modes may actually be propagating in a waveguide requires a source to determine. Eigenvalue problems will be formulated and solved in Chapter 6 to analyze waveguides and in Chapter 7 to calculate photonic bands.

For scattering problems, the matrix equation will have the form of

$$\mathbf{A}\mathbf{f} = \mathbf{b} \quad (2.163)$$

In (2.163), the matrix \mathbf{A} enforces Maxwell's equations like the \mathbf{A} matrix in (2.162). In contrast to eigenvalue problems, scattering problems require a source and produce only a single solution. The source is encoded into the column vector \mathbf{b} . Scattering problems will be formulated and solved in Chapters 8 to 10 for a variety of devices and applications.

References

- [1] Sadiku, M. N. O., *Elements of Electromagnetics*, Seventh Edition, New York: Oxford University Press, 2018.
- [2] Balanis, C. A., *Advanced Engineering Electromagnetics*, New York: Wiley, 1989.
- [3] Fleisch, D., *A Student's Guide to Maxwell's Equations*, New York: Cambridge University Press, 2008.
- [4] Huray, P. G., *Maxwell's Equations*, Hoboken, NJ: John Wiley & Sons, 2011.
- [5] Maxwell, J. C., "On Physical Lines of Force," *Philosophical Magazine*, Vol. 90, No. S1, 2010, pp. 11–23.
- [6] Simovski, C., "Material Parameters of Metamaterials (A Review)," *Optics and Spectroscopy*, Vol. 107, No. 5, 2009, pp. 726–753.
- [7] Rumpf, R. C., "Engineering the Dispersion and Anisotropy of Periodic Electromagnetic Structures," *Solid State Physics*, Vol. 66, 2015, pp. 213–300.
- [8] Mackay, T. G., and A. Lakhtakia, *Electromagnetic Anisotropy and Bianisotropy: A Field Guide*, Singapore: World Scientific, 2019.
- [9] Brand, L., *Vector and Tensor Analysis*, Mineola, NY: Dover Publications, 2020.
- [10] Rewienski, M., and M. Mrozowski, "An Iterative Algorithm for Reducing Dispersion Error on Yee's Mesh in Cylindrical Coordinates," *IEEE Microwave and Guided Wave Letters*, Vol. 10, No. 9, 2000, pp. 353–355.
- [11] Xiao, J., H. Ni, and X. Sun, "Full-Vector Mode Solver for Bending Waveguides Based on the Finite-Difference Frequency-Domain Method in Cylindrical Coordinate Systems," *Optics Letters*, Vol. 33, No. 16, 2008, pp. 1848–1850.
- [12] Aghaie, K. Z., S. Fan, and M. J. Dignonnet, "Birefringence Analysis of Photonic-Bandgap Fibers Using the Hexagonal Yee's Cell," *IEEE J. of Quantum Electronics*, Vol. 46, No. 6, 2010, pp. 920–930.
- [13] Guo, S., *et al.*, "Photonic Band Gap Analysis Using Finite-Difference Frequency-Domain Method," *Optics Express*, Vol. 12, No. 8, 2004, pp. 1741–1746.
- [14] Speiser, D., K. Williams, and S. Caparrini, *Discovering the Principles of Mechanics 1600–1800: Essays*, Basel–Boston, MA: Birkhäuser, 2008.
- [15] Goldstein, D. H., *Polarized Light*, Boca Raton, FL: CRC Press, 2017.
- [16] Hecht, E., *Optics*, Fifth Edition, Boston, MA: Pearson Education, Inc., 2017.
- [17] Loewen, E. G., and E. Popov, *Diffraction Gratings and Applications*: CRC Press, 2018.
- [18] Kuester, E. F., *Theory of Waveguides and Transmission Lines*, Boca Raton, FL: CRC Press, 2020.
- [19] Okamoto, K., *Fundamentals of Optical Waveguides*, San Diego, CA: Academic Press, 2006.
- [20] Ōkoshi, T., *Optical Fibers*, New York: Academic Press, 1982.
- [21] Olshlager, F., *Electromagnetic Waveguides and Transmission Lines*, Oxford, U.K.: Oxford University Press, 1999.
- [22] Poh, S. Y., W. C. Chew, and J. A. Kong, "Approximate Formulas for Line Capacitance and Characteristic Impedance of Microstrip Line," *IEEE Trans. on Microwave Theory and Techniques*, Vol. 29, No. 2, 1981, pp. 135–142.
- [23] Selvaraja, S. K., and P. Sethi, "Review on Optical Waveguides," *Emerging Waveguide Technology*, Vol. 95, 2018.
- [24] Pozar, D. M., *Microwave Engineering*, Hoboken, NJ: John Wiley & Sons, 2011.
- [25] Oldham, K. T. S., *The Doctrine of Description: Gustav Kirchhoff, Classical Physics, and the "Purpose of all Science" 19th-Century Germany*, Berkeley, CA: University of California, 2008.
- [26] Alexander, C. K., and M. N. O. Sadiku, *Fundamentals of Electric Circuits*, New York: McGraw-Hill Higher Education, 2007.

The Finite-Difference Method

This chapter describes how functions can be made discrete and how differential equations can be solved using the finite-difference method. Estimating derivatives of discrete functions using finite differences is introduced, and boundary conditions for calculating the derivatives at the boundaries of the discrete functions are described. With this background, the concept of derivative matrices is introduced that will allow differential equations to be written as matrix equations almost effortlessly. The chapter ends by discussing how to solve differential equations using the finite-difference method and how to handle multivariable systems.

3.1 Introduction

The finite-difference frequency-domain (FDFD) method is useful for solving Maxwell's equations where analytical solutions may be highly restricted or impossible. In such situations, a numerical approach is needed and the finite-difference method is chosen for this purpose. Adopting a numerical solution allows FDFD to solve virtually any simulation problem. The *finite-difference method* is perhaps the easiest and most intuitive numerical technique for solving differential equations [1]. With practice, new equations can often be solved in mere minutes. The primary drawback of the finite-difference method is its poor efficiency compared to other methods.

In order to implement the finite-difference method, functions are made discrete. That is, function values such as electric fields will only be stored at discrete points using the grids discussed in Chapter 1. Instead of having analytical equations, functions are stored as arrays of discrete numbers as shown in Figure 3.1. While interpolation techniques could be used to calculate the function at positions between the discrete points, it is most efficient to implement the finite-difference method so that interpolations are not needed.

It is important to realize that the finite-difference method does not find analytical solutions. It is given discrete data and calculates the solution as discrete data. However, the discrete solution can suggest the form of an analytical answer where one may exist. It is also very common to become so good at obtaining numerical solutions that way too little thought is ever given to analytical solutions. Do not let strong numerical skills stop you from thinking logically about a problem before solving it numerically!

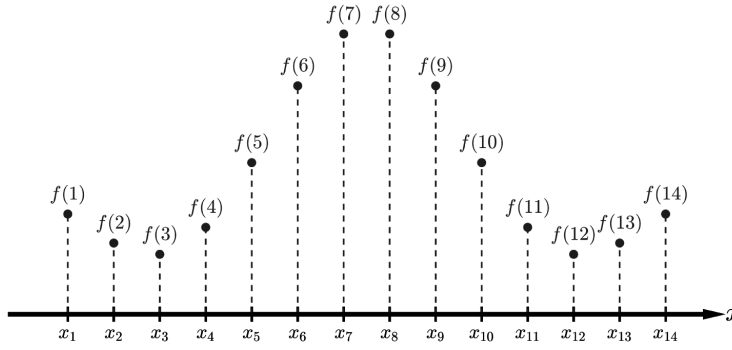


Figure 3.1 A discrete function $f(n)$ composed of 14 points.

3.2 Finite-Difference Approximations

To solve differential equations and analyze functions, it is necessary to evaluate their derivatives. At first glance, this may not seem possible if no analytical expressions exist for the functions. For discrete functions, derivatives can be estimated using finite-difference approximations. In the most general words, a *finite-difference approximation* is a weighted sum of the function values in close proximity to the point where a derivative is being estimated [1]. The trick is determining the weights in the sum that correctly estimate derivatives.

A finite-difference approximation is best explained with an example. Suppose a function $f(x)$ is made discrete and stored in an array $f(n)$ with array index n . Figure 3.2 shows three points of the discrete function. In this figure, the spacing between the discrete points along the x -axis is uniform and equal to Δx . The light gray line is the original continuous function and is shown here to help illustrate the error in the estimation. In practice, only the discrete function is known and there would be no analytical function to plot.

Now suppose it is desired to estimate the first derivative at the second point $n = 2$. The first derivative is the slope, which is calculated as “rise divided by run.” A simple way to estimate the slope at this second point is to calculate the slope of the line connecting the first and third points. The rise in this case, or how much the function increases, is $f(3) - f(1)$. The run, or the distance between the first and third points, is $x_3 - x_1 = 2\Delta x$. Putting these together, the derivative at the second point can be estimated according to

$$\frac{df(2)}{dx} \cong \frac{f(3) - f(1)}{2\Delta x} \quad (3.1)$$

Equation (3.1) is a finite-difference approximation of the first-order derivative at $n = 2$. The solid black lines in Figure 3.2 show the exact slope calculated from the analytical function and the estimated slope calculated from the finite-difference estimation. In practice, the exact slope is rarely ever known because the analytical function is rarely ever known. The exact slope is only shown here for illustration purposes. It is clear that the estimated slope is not perfectly equal to the exact slope. The angle between the slopes represents the numerical error introduced by

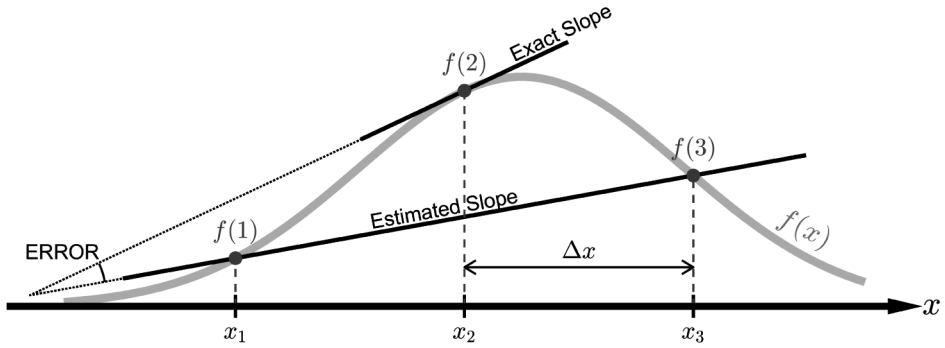


Figure 3.2 Illustration of a finite-difference approximation and associated numerical error.

the finite-difference approximation. In a typical problem being solved by the finite-difference method, thousands to millions of finite-difference approximations will be made at the same time. Each finite-difference approximation produces numerical errors, and the errors can even trickle into the other finite-difference approximations to spread and amplify. It is healthy to become very paranoid about numerical errors and to take precautions to minimize and control the errors.

In the example illustrated in Figure 3.2, the error arises because the discrete points are spaced too far apart to resolve the function accurately. This implies that error can be reduced by reducing the distance Δx between the discrete points. However, this requires more points to be used and more calculations to be performed when the entire function is being analyzed. This will reduce the overall efficiency of the method by increasing the memory required to solve the problem and increase the time it takes for the calculations to be performed. Accuracy versus efficiency is the fundamental tradeoff for virtually all numerical methods and simulation techniques. A tremendous amount of research has been devoted to getting away with fewer points in a simulation without sacrificing accuracy. If you are a beginner in computational electromagnetics, it may take you a few weeks to understand and implement a new method, but you will spend the rest of your career battling the tradeoff between accuracy and efficiency.

3.2.1 Deriving Expressions for Finite-Difference Approximations

Equation (3.1) estimates the first derivative at $n = 2$ from the function values from $n = 1$ to $n = 3$. What if a different derivative is desired? What if the location of the points were different? What if it is desired to calculate the derivative at a different location than $n = 2$? Any of these changes will require a different expression to estimate the derivative. This section will describe a simple technique to derive expressions that estimate the function or any of its derivatives at any position and from any distribution of points. The method works by fitting a polynomial to the distribution of points and then using the polynomial to write the expressions that approximate the function or one of its derivatives. Start with the following N th order polynomial.

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_Nx^N \quad (3.2)$$

The coefficients a_0 to a_N can be found by fitting the polynomial to a set of $N + 1$ points, expressed as x_1 to x_{N+1} . To do this, (3.2) is written for each of the $N + 1$ points as

$$\begin{aligned} f(x_1) &= a_0 + a_1x_1 + a_2x_1^2 + \cdots + a_Nx_1^N \\ f(x_2) &= a_0 + a_1x_2 + a_2x_2^2 + \cdots + a_Nx_2^N \\ &\vdots \\ f(x_{N+1}) &= a_0 + a_1x_{N+1} + a_2x_{N+1}^2 + \cdots + a_Nx_{N+1}^N \end{aligned} \quad (3.3)$$

The set of equations in (3.3) can be written in matrix form as

$$\mathbf{f} = \mathbf{X}\mathbf{a} \quad (3.4)$$

where

$$\mathbf{f} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ \vdots \\ f_{N+1} \end{bmatrix} \quad (3.5)$$

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^N \\ 1 & x_2 & x_2^2 & \cdots & x_2^N \\ 1 & x_3 & x_3^2 & \cdots & x_3^N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{N+1} & x_{N+1}^2 & \cdots & x_{N+1}^N \end{bmatrix} \quad (3.6)$$

$$\mathbf{a} = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} \quad (3.7)$$

To write simpler equations, the function $f(x)$ evaluated at the discrete position x_m will be written as f_m . Numerical values for the discrete function values f_m placed into the column vector \mathbf{f} are not known and remain symbolic when deriving expressions for finite-difference approximations. The matrix \mathbf{X} in (3.6) has the form of a *Vandermonde matrix* [2, 3] and contains the positions of the discrete points raised to different powers along its columns. The polynomial coefficients to be calculated are stored in the column vector \mathbf{a} . They are found by solving (3.4) for \mathbf{a} and extracting them from the column vector.

$$\mathbf{a} = \mathbf{X}^{-1}\mathbf{f} \quad (3.8)$$

After the polynomial coefficients are calculated, (3.2) can be directly used to interpolate the function at any position x . Expressions to estimate any of the

derivatives at position x can be found by differentiating (3.2) with respect to x . The polynomial and its first two derivatives are

$$\begin{aligned} f(x) &= a_0 + a_1x + a_2x^2 + \cdots + a_Nx^N \\ f'(x) &= a_1 + 2a_2x + \cdots + Na_Nx^{N-1} \\ f''(x) &= 2a_2 + 6a_3x + \cdots + N(N-1)a_Nx^{N-2} \\ &\vdots \end{aligned} \quad (3.9)$$

After close inspection of (3.9), the equations will simplify considerably if the function or its derivatives are evaluated at $x = 0$. Setting $x = 0$ reduces the expressions in (3.9) to

$$\begin{aligned} f(0) &= a_0 \\ f'(0) &= a_1 \\ f''(0) &= 2a_2 \\ &\vdots \\ f^{(n)}(0) &= (n!)a_n \end{aligned} \quad (3.10)$$

Equation (3.10) implies that to derive an expression for the n th derivative, at least $n + 1$ points are needed so that the polynomial coefficients a_0 to a_n can be calculated. A finite-difference approximation derived from $n + 1$ points is said to be n th-order accurate because it is calculated from an n th-order polynomial.

The above discussion implies a simple procedure to derive expressions for finite-difference approximations. Step 1 identifies $N + 1$ coordinates from which to estimate the function or one of its derivatives. The position x_{fd} where the function or one of its derivatives is to be estimated is subtracted from the list of coordinates. This is done so that (3.9) can be evaluated at $x = 0$. Putting the shifted coordinates into the column vector \mathbf{x} gives

$$\text{Step 1} \quad \mathbf{x} = \begin{bmatrix} x_1 - x_{\text{fd}} \\ x_2 - x_{\text{fd}} \\ x_3 - x_{\text{fd}} \\ \vdots \\ x_{N+1} - x_{\text{fd}} \end{bmatrix} \quad (3.11)$$

Step 2 builds the matrix \mathbf{X} in (3.6) using the shifted coordinates stored in the column vector \mathbf{x} . A useful way to express \mathbf{X} is given in (3.12). Observe that the first column in \mathbf{X} contains all 1's, or \mathbf{x}^0 . The second column in \mathbf{X} is \mathbf{x}^1 , the third column in \mathbf{X} is \mathbf{x}^2 , and so on. This suggests an easy way to build the matrix \mathbf{X} in MATLAB by inserting \mathbf{x} raised to different powers into the columns of \mathbf{X} .

$$\text{Step 2} \quad \mathbf{X} = \begin{bmatrix} 1 \\ 1 \\ \mathbf{x} & \mathbf{x}^2 & \cdots & \mathbf{x}^N \\ \vdots \\ 1 \end{bmatrix} \quad (3.12)$$

Step 3 inverts the matrix \mathbf{X} to obtain the matrix \mathbf{Y} . It will be shown that the coefficients of the finite-difference approximations are contained along the rows of \mathbf{Y} , but still need to be multiplied by the constant $n!$, where n is the row number containing the finite-difference coefficients.

$$\text{Step 3} \quad \mathbf{Y} = \mathbf{X}^{-1} \quad (3.13)$$

Step 4 calculates the polynomial coefficients as $\mathbf{a} = \mathbf{Y}\mathbf{f}$. The polynomial coefficients are extracted from the column vector \mathbf{a} as

$$\begin{aligned} \text{Step 4} \quad a_0 &= y_{11}f_1 + y_{12}f_2 + y_{13}f_3 + \cdots y_{1,N+1}f_{N+1} \\ a_1 &= y_{21}f_1 + y_{22}f_2 + y_{23}f_3 + \cdots y_{2,N+1}f_{N+1} \\ a_2 &= y_{31}f_1 + y_{32}f_2 + y_{33}f_3 + \cdots y_{3,N+1}f_{N+1} \\ &\vdots \\ a_N &= y_{N+1,1}f_1 + y_{N+1,2}f_2 + y_{N+1,3}f_3 + \cdots y_{N+1,N+1}f_{N+1} \end{aligned} \quad (3.14)$$

Step 5 writes the expression to interpolate the function or estimate one of its derivatives from these polynomial coefficients using (3.10). This is

$$\begin{aligned} \text{Step 5} \quad f(0) &= y_{11}f_1 + y_{12}f_2 + y_{13}f_3 + \cdots y_{1,N+1}f_{N+1} \\ f'(0) &= y_{21}f_1 + y_{22}f_2 + y_{23}f_3 + \cdots y_{2,N+1}f_{N+1} \\ f''(0) &= 2y_{31}f_1 + 2y_{32}f_2 + 2y_{33}f_3 + \cdots 2y_{3,N+1}f_{N+1} \\ &\vdots \\ f^{(m)}(0) &= (m!)y_{m+1,1}f_1 + (m!)y_{m+1,2}f_2 + (m!)y_{m+1,3}f_3 + \cdots (m!)y_{m+1,N+1}f_{N+1} \end{aligned} \quad (3.15)$$

The terms multiplying the discrete function values in (3.15) are called the *finite-difference coefficients*. Information from the position of the points and the position where the expression is estimating the derivative is encoded into these terms. With practice, the procedure can stop at Step 3 and the finite-difference coefficients can be taken directly from the rows of \mathbf{Y} . Be careful to multiply the row in \mathbf{Y} by the correct constant if a second derivative or higher is being derived.

3.2.2 Example #1—Interpolations and Derivatives from Three Points

It is very common to solve problems that contain only a single variable with finite-differences written from a span of three points on a grid. If the function or one of its derivatives is to be calculated at the middle point of three points with uniform spacing Δx , the following equations summarize the results of each step of the derivation procedure described above.

$$\text{Step 1} \quad \mathbf{x} = \begin{bmatrix} -\Delta x \\ 0 \\ +\Delta x \end{bmatrix} \quad (3.16)$$

$$\text{Step 2} \quad \mathbf{X} = \begin{bmatrix} 1 & -\Delta x & \Delta x^2 \\ 1 & 0 & 0 \\ 1 & +\Delta x & \Delta x^2 \end{bmatrix} \quad (3.17)$$

$$\text{Step 3} \quad \mathbf{Y} = \begin{bmatrix} 0 & 1 & 0 \\ -\frac{1}{2\Delta x} & 0 & \frac{1}{2\Delta x} \\ \frac{1}{2\Delta x^2} & -\frac{1}{\Delta x^2} & \frac{1}{2\Delta x^2} \end{bmatrix} \quad (3.18)$$

$$\begin{aligned} \text{Step 4} \quad a_0 &= 0f_1 + 1f_2 + 0f_3 \\ a_1 &= -\frac{1}{2\Delta x}f_1 + 0f_2 + \frac{1}{2\Delta x}f_3 \\ a_2 &= \frac{1}{2\Delta x^2}f_1 - \frac{1}{\Delta x^2}f_2 + \frac{1}{2\Delta x^2}f_3 \end{aligned} \quad (3.19)$$

$$\text{Step 5a} \quad f_2 \approx f_2 \quad (3.20)$$

$$\text{Step 5b} \quad \frac{df_2}{dx} \approx \frac{f_3 - f_1}{2\Delta x} \quad (3.21)$$

$$\text{Step 5c} \quad \frac{d^2f_2}{dx^2} \approx \frac{f_3 - 2f_2 + f_1}{\Delta x^2} \quad (3.22)$$

Equation (3.20) gives an expression to interpolate the function value at $n = 2$ from the function values at $n = 1$ to $n = 3$. Since the function value at $n = 2$ is one of the known function values, the interpolation expression is just f_2 . This obvious answer is a great check to verify the calculations were performed correctly. Equation (3.21) estimates the first derivative at $n = 2$. It should be recognized that this is the same as that in (3.1) which was derived by slope. Last, (3.22) estimates the second derivative at $n = 2$. In this case, information from all three function values were needed in order to assess the curvature of the function quantified by the second derivative.

What if the same derivation is to be performed, but it is desired to get an expression for the function and its derivatives at $n = 1$ from the function values from $n = 1$ to $n = 3$? This derivation gives

$$\text{Step 1} \quad \mathbf{x} = \begin{bmatrix} 0 \\ \Delta x \\ 2\Delta x \end{bmatrix} \quad (3.23)$$

$$\text{Step 2} \quad \mathbf{X} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & \Delta x & \Delta x^2 \\ 1 & 2\Delta x & 4\Delta x^2 \end{bmatrix} \quad (3.24)$$

$$\text{Step 3} \quad \mathbf{Y} = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{3}{2\Delta x} & \frac{2}{\Delta x} & -\frac{1}{2\Delta x} \\ \frac{1}{2\Delta x^2} & -\frac{1}{\Delta x^2} & \frac{1}{2\Delta x^2} \end{bmatrix} \quad (3.25)$$

$$\text{Step 4} \quad a_0 = 1f_1 + 0f_2 + 0f_3 \quad (3.26)$$

$$a_1 = -\frac{3}{2\Delta x}f_1 + \frac{2}{\Delta x}f_2 - \frac{1}{2\Delta x}f_3$$

$$a_2 = \frac{1}{2\Delta x^2}f_1 - \frac{1}{\Delta x^2}f_2 + \frac{1}{2\Delta x^2}f_3$$

$$\text{Step 5a} \quad f_1 \approx f_1 \quad (3.27)$$

$$\text{Step 5b} \quad \frac{df_1}{dx} \approx \frac{-f_3 + 4f_2 - 3f_1}{2\Delta x} \quad (3.28)$$

$$\text{Step 5c} \quad \frac{d^2f_1}{dx^2} \approx \frac{f_3 - 2f_2 + f_1}{\Delta x^2} \quad (3.29)$$

Observe that the interpolation in (3.27) is correct. Observe that the first derivative in (3.28) now contains three terms instead of two due to estimating the derivative at the first point instead of the second. Last, observe (3.29) is the same as (3.22) despite the derivative being estimated at a different point. This happens because there is no other way to estimate a second derivative from only three points.

3.2.3 Example #2—Interpolations and Derivatives from Two Points

All of the approximations in this book for FDFD will estimate the first derivatives at the center of two adjacent points on the grid. It is trivial to show that the finite-difference approximation is the slope between the two points.

$$\frac{df_{1.5}}{dx} \approx \frac{f_2 - f_1}{\Delta x} \quad (3.30)$$

This finite-difference approximation can also be derived following the steps outlined in this chapter. This gives

$$\text{Step 1} \quad \mathbf{x} = \begin{bmatrix} -\frac{\Delta x}{2} \\ +\frac{\Delta x}{2} \end{bmatrix} \quad (3.31)$$

$$\text{Step 2} \quad \mathbf{X} = \begin{bmatrix} 1 & -\frac{\Delta x}{2} \\ 1 & +\frac{\Delta x}{2} \end{bmatrix} \quad (3.32)$$

$$\text{Step 3} \quad \mathbf{Y} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ -\frac{1}{\Delta x} & +\frac{1}{\Delta x} \end{bmatrix} \quad (3.33)$$

$$\begin{aligned} \text{Step 4} \quad a_0 &= \frac{1}{2}f_1 + \frac{1}{2}f_2 \\ a_1 &= -\frac{1}{\Delta x}f_1 + \frac{1}{\Delta x}f_2 \end{aligned} \quad (3.34)$$

$$\text{Step 5a} \quad f_{1.5} \approx \frac{f_2 + f_1}{2} \quad (3.35)$$

$$\text{Step 5b} \quad \frac{df_{1.5}}{dx} \approx \frac{f_2 - f_1}{\Delta x} \quad (3.36)$$

Equation (3.35) is averaging the function values at $n = 1$ and $n = 2$ to interpolate the function at $n = 1/2$. The first derivative in (3.36) is the same as (3.30) and is the finite-difference approximation that will be used throughout this book for FDFD. It is a second-order accurate finite-difference approximation of the first derivative. It is not possible to derive an expression for any higher derivatives without using more points.

3.2.4 Example #3—Interpolations and Derivatives from Four Points

Suppose it is desired to estimate the function value or one of its derivatives at the center of four points. This will be derived following the steps outlined in this section.

$$\text{Step 1} \quad \mathbf{x} = \begin{bmatrix} -\frac{3\Delta x}{2} \\ -\frac{\Delta x}{2} \\ +\frac{\Delta x}{2} \\ +\frac{3\Delta x}{2} \end{bmatrix} \quad (3.37)$$

$$\text{Step 2} \quad \mathbf{X} = \begin{bmatrix} 1 & -\frac{3\Delta x}{2} & \frac{9\Delta x^2}{4} & -\frac{27\Delta x^3}{8} \\ 1 & -\frac{\Delta x}{2} & \frac{\Delta x^2}{4} & -\frac{\Delta x^3}{8} \\ 1 & +\frac{\Delta x}{2} & \frac{\Delta x^2}{4} & +\frac{\Delta x^3}{8} \\ 1 & +\frac{3\Delta x}{2} & \frac{9\Delta x^2}{4} & +\frac{27\Delta x^3}{8} \end{bmatrix} \quad (3.38)$$

$$\text{Step 3} \quad \mathbf{Y} = \begin{bmatrix} -\frac{1}{16} & \frac{9}{16} & \frac{9}{16} & -\frac{1}{16} \\ \frac{1}{24\Delta x} & -\frac{9}{8\Delta x} & +\frac{9}{8\Delta x} & -\frac{1}{24\Delta x} \\ \frac{1}{4\Delta x^2} & -\frac{1}{4\Delta x^2} & -\frac{1}{4\Delta x^2} & \frac{1}{4\Delta x^2} \\ -\frac{1}{6\Delta x^3} & \frac{1}{2\Delta x^3} & -\frac{1}{2\Delta x^3} & \frac{1}{6\Delta x^3} \end{bmatrix} \quad (3.39)$$

$$\begin{aligned}
\text{Step 4} \quad a_0 &= -\frac{1}{16}f_1 + \frac{9}{16}f_2 + \frac{9}{16}f_3 - \frac{1}{16}f_4 \\
a_1 &= \frac{1}{24\Delta x}f_1 - \frac{27}{24\Delta x}f_2 + \frac{27}{24\Delta x}f_3 - \frac{1}{24\Delta x}f_4 \\
a_2 &= \frac{1}{4\Delta x^2}f_1 - \frac{1}{4\Delta x^2}f_2 - \frac{1}{4\Delta x^2}f_3 + \frac{1}{4\Delta x^2}f_4 \\
a_3 &= -\frac{1}{6\Delta x^3}f_1 + \frac{3}{6\Delta x^3}f_2 - \frac{3}{6\Delta x^3}f_3 + \frac{1}{6\Delta x^3}f_4
\end{aligned} \tag{3.40}$$

$$\text{Step 5a} \quad f_{2.5} \approx \frac{-f_4 + 9f_3 + 9f_2 - f_1}{16} \tag{3.41}$$

$$\text{Step 5b} \quad \frac{df_{2.5}}{dx} \approx \frac{-f_4 + 27f_3 - 27f_2 + f_1}{24\Delta x} \tag{3.42}$$

$$\text{Step 5c} \quad \frac{d^2f_{2.5}}{dx^2} \approx \frac{f_4 - f_3 - f_2 + f_1}{2\Delta x^2} \tag{3.43}$$

$$\text{Step 5d} \quad \frac{d^3f_{2.5}}{dx^3} \approx \frac{f_4 - 3f_3 + 3f_2 - f_1}{\Delta x^3} \tag{3.44}$$

Throughout this book, FDFD will be based on the finite-difference approximation in (3.36). However, it is possible to base FDFD on the finite-difference approximation in (3.42). A fourth-order polynomial was used so this is a fourth-order accurate finite-difference approximation. In principle, the improved accuracy should allow FDFD problems to be solved with fewer points on the grid and give answers with less numerical dispersion.

3.3 Numerical Differentiation

Numerical differentiation uses finite-difference approximations to calculate the derivative of a discrete function [1]. Numerical differentiation is not the finite-difference method. The finite-difference method uses finite-difference approximations to solve differential equations, not to calculate derivatives. This section is not yet talking about the finite-difference method, but many of the concepts involved in the finite-difference method are easier to learn in the context of numerical differentiation.

In order to calculate the derivative at every point of a discrete function, each point where the derivative is to be calculated requires its own finite-difference approximation. This is illustrated in Figure 3.3 for a discrete function composed of 14 different points. To do this, (3.1) was applied to all 14 points and the resulting finite-difference approximations were written above the points.

In a computer code, it would be inefficient to have a separate line of code to calculate each finite-difference. Instead, a `for` loop is used to make the code more compact. In MATLAB, the code to perform the numerical differentiation in Figure 3.3 would look something like

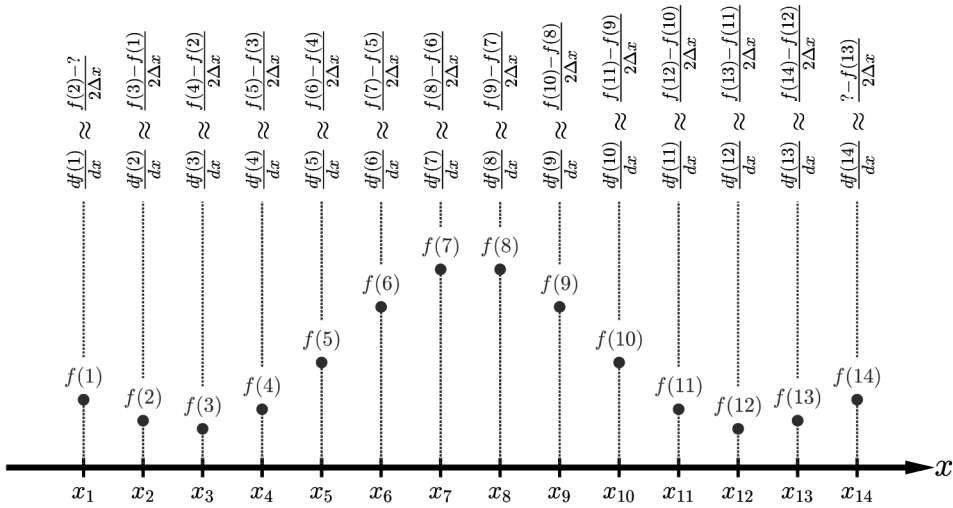


Figure 3.3 Illustration of finite-difference approximations used in numerical differentiation.

```
for n = 1 : 14
    fder(n) = (f(n+1) - f(n-1))/(2*dx);
end
```

3.4 Numerical Boundary Conditions

If you were following the previous section closely enough, perhaps you caught a serious problem that will arise at the first and last points that will cause the numerical differentiation to fail. The finite-difference approximations at the end points of the discrete function required function values $f(0)$ and $f(15)$. These do not exist and so they cannot be used in any finite-difference approximations. However, something must still be done to calculate the derivatives at the two end points. The manner in which the end points are handled is called a *numerical boundary condition* [1]. Great care must be taken when choosing the numerical boundary condition to ensure that it is consistent with the physics of what is being analyzed or simulated. There are many options for boundary conditions, but the two most common for FDFD are the Dirichlet and the periodic boundary conditions (PBCs) described below.

3.4.1 Dirichlet Boundary Conditions

Perhaps the simplest way to handle a boundary is the *Dirichlet boundary condition*, which assumes the discrete function value outside of the grid is a fixed value [4]. Throughout this book, that fixed value will be zero. The finite-difference approximations across an N -point grid using Dirichlet boundary conditions are summarized in (3.45). The first finite-difference approximation is only used at the first point $n = 1$, the second finite-difference approximation is used at all points except at the two boundaries, and the third finite-difference approximation is only used at the

last point $n = N$. The points at the boundaries of the grid get their own special and unique finite-difference approximations. They are very lucky points!

$$\frac{df(n)}{dx} \approx \begin{cases} \frac{f(2) - 0}{2\Delta x} & n = 1 \\ \frac{f(n+1) - f(n-1)}{2\Delta x} & 2 \leq n \leq N-1 \\ \frac{0 - f(N-1)}{2\Delta x} & n = N \end{cases} \quad (3.45)$$

3.4.2 Periodic Boundary Conditions

Another option is a PBC [5]. If it is known that the discrete function $f(n)$ repeats every 14 points, then $f(14)$ can be used in place of $f(0)$ and $f(1)$ can be used in place of $f(15)$. The finite-difference approximations across an N -point grid using PBCs is summarized in (3.46). The first finite-difference approximation is only used at the first point $n = 1$, the second finite-difference approximation is used at all points except at the two grid boundaries, and the third finite-difference approximation is only used at the last point $n = N$. Like before, the points at the boundaries of the grid get their own special and unique finite-difference approximations. The PBC in (3.46) will be modified in Chapter 4 to account for phase of a wave across the grid.

$$\frac{df(n)}{dx} \approx \begin{cases} \frac{f(2) - f(N)}{2\Delta x} & n = 1 \\ \frac{f(n+1) - f(n-1)}{2\Delta x} & 2 \leq n \leq N-1 \\ \frac{f(1) - f(N-1)}{2\Delta x} & n = N \end{cases} \quad (3.46)$$

3.5 Derivative Matrices

It is possible to build a matrix \mathbf{D}_x that calculates the numerical derivative of a discrete function $f(n)$ with respect to x when the discrete function is stored in a column vector \mathbf{f} . The numerical derivative would then be stored in another column vector \mathbf{f}' . Equation (3.47) expresses this numerical calculation where \mathbf{D}_x premultiplies \mathbf{f} to get \mathbf{f}' . This section will discuss how to construct a derivative matrix \mathbf{D}_x that correctly performs a numerical differentiation.

$$\mathbf{f}' = \mathbf{D}_x \mathbf{f} \quad (3.47)$$

The manner in which the derivative matrix \mathbf{D}_x is constructed for the example depicted in Figure 3.3 is shown in Figure 3.4. The column vector \mathbf{f}' on the left side of the matrix equation is populated with the finite-difference expressions that need

to be calculated. In this case, Dirichlet boundary conditions were used. The column vector \mathbf{f} on the far-right side of the matrix equation is populated with the discrete function values at each point on the grid. The large square matrix \mathbf{D}_x is populated last with values so that the matrix multiplication on the right side gives the column vector on the left side. When determining the elements of the derivative matrix, it is a common practice to first write the matrix equation with a blank derivative matrix and second to populate the derivative matrix.

It turns out that \mathbf{D}_x is a banded matrix containing numerical values along only two of its diagonals. This makes constructing the derivative matrix very easy, especially in MATLAB that has built-in functions for inserting diagonals into matrices. Also, observe that most of the elements in the 14×14 derivative matrix are zero. Matrices typically encountered in the finite-difference method are of size $10k \times 10k$ and larger. Tremendous memory savings can be achieved by storing derivative matrices as sparse matrices. Always construct and store derivative matrices as sparse matrices! Do not worry, MATLAB makes working with sparse matrices very easy.

From Figure 3.4, the derivative matrix with Dirichlet boundary conditions can be extracted from the matrix equation and written more generically as

$$\mathbf{D}_x = \frac{1}{2\Delta x} \begin{bmatrix} & +1 & & & & \\ -1 & & +1 & & & \\ & -1 & & \ddots & & \\ & & \ddots & & +1 & \\ & & & -1 & & +1 \\ & & & & -1 & \end{bmatrix} \quad \text{Dirichlet boundary conditions} \quad (3.48)$$

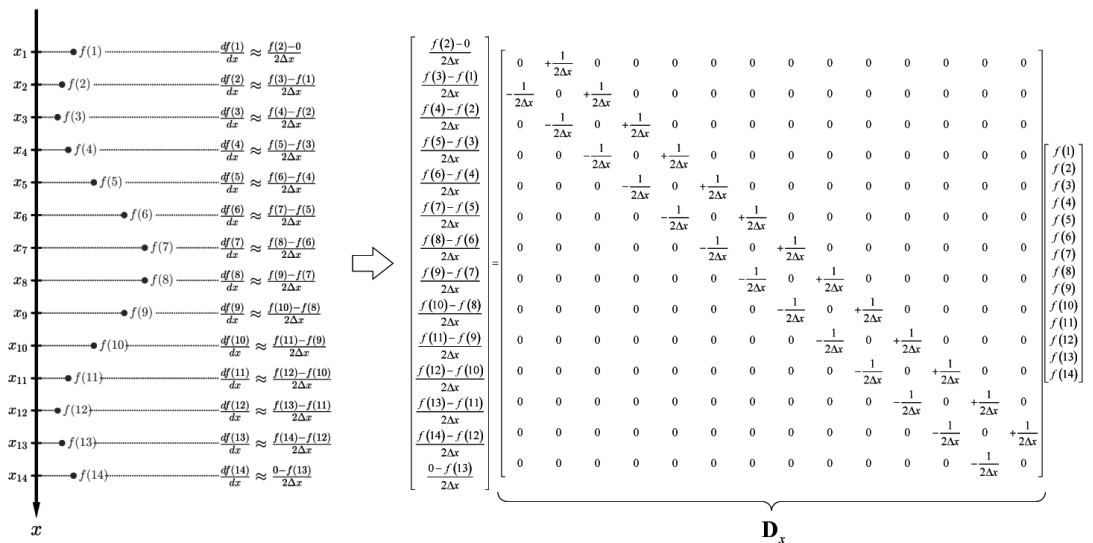


Figure 3.4 Construction of the derivative matrix using Dirichlet boundary conditions.

This derivative matrix \mathbf{D}_x has a very convenient form due to being composed of just two uniform diagonals and a constant $1/2\Delta x$ on the outside. This makes it very easy to construct as a sparse matrix in MATLAB using the following code.

```
Z = sparse(N,N);
DX = spdiags(-ones(N-1,1),-1,Z);
DX = spdiags(+ones(N,1),+1,DX);
DX = DX/(2*dx);
```

This convenient form was possible due to uniform spacing between points. It is possible to have nonuniform spacing in FDFD, but the derivative matrices are more complicated to build. For a uniform grid, building the derivative matrix begins by declaring the matrix \mathbf{Z} to be a sparse $N \times N$ matrix filled with all zeros. The function `spdiags()` inserts and extracts diagonals in sparse matrices, but is only being used to insert diagonals here. It is given three input arguments. The first input argument is a one-dimensional array of numbers to insert along the diagonal. The second input argument is the diagonal number to insert numbers into, where 0 indicates the center diagonal, positive integers identify diagonals above the center diagonal, and negative integers identify diagonals below the center diagonal. The third input argument is the matrix to insert the diagonal into. The first call to `spdiags()` inserts the -1 diagonal into the zeros matrix \mathbf{Z} . The second call to `spdiags()` inserts the $+1$ diagonal into the \mathbf{DX} matrix instead of the \mathbf{Z} matrix so that the first diagonal is retained. The last line of code divides all of the elements by $2 \cdot dx$.

This example can be repeated to demonstrate incorporating PBCs. This is illustrated in Figure 3.5. In this case, additional terms are added to the upper-rightmost element in the matrix and lower-leftmost element because the PBCs required them at the first and last points of the discrete function.

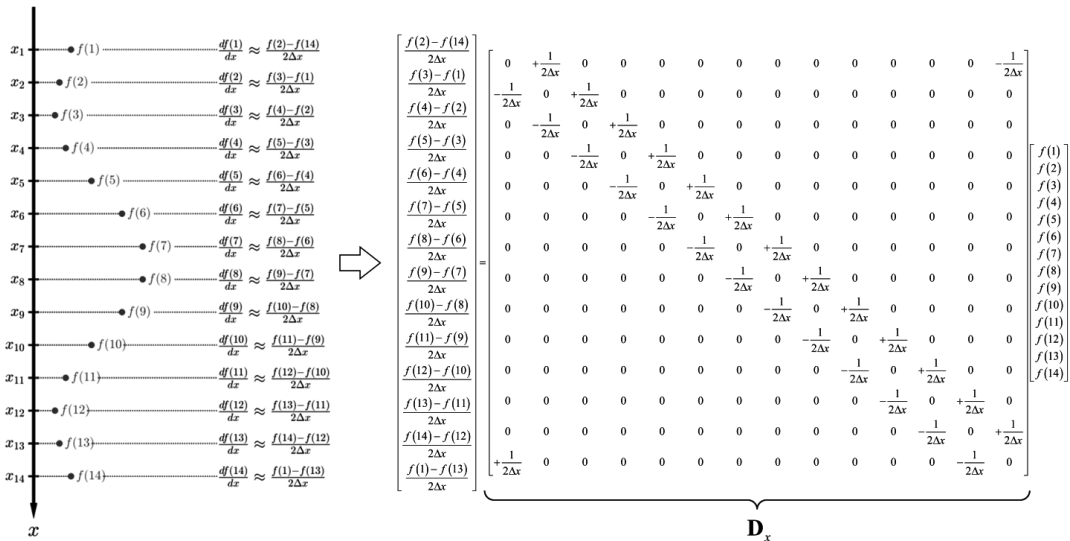


Figure 3.5 Construction of the derivative matrix using PBCs.

From Figure 3.5, the derivative matrix with PBCs can be extracted from the matrix equation and written more generically as

$$\mathbf{D}_x = \frac{1}{2\Delta x} \begin{bmatrix} & +1 & & & -1 \\ -1 & & +1 & & \\ & -1 & & \ddots & \\ & & \ddots & & +1 \\ & & & -1 & +1 \\ +1 & & & -1 & \end{bmatrix} \quad \text{Periodic boundary conditions} \quad (3.49)$$

The MATLAB code to build this matrix is similar to the case for Dirichlet boundary conditions, but two additional numbers must be inserted into the matrix to incorporate the PBCs.

```
Z = sparse(N,N);
DX = spdiags(-ones(N-1,1),-1,Z);
DX = spdiags(+ones(N,1),+1,DX);
DX(1,N) = -1;
DX(N,1) = +1;
DX = DX/(2*dx);
```

Derivative matrices can be used to perform numerical differentiation following (3.47), but this is not their real purpose. Numerical differentiation is more efficiently performed with a `for` loop as previously discussed. Instead, the derivative matrices will be used in the next section to convert differential equations into matrix equations for numerical solution. Using derivative matrices to perform numerical differentiation is a good way to verify that they work correctly. If the derivative matrix was already constructed for other reasons, using them to perform numerical differentiation is a good application.

3.6 Finite-Difference Approximation of Differential Equations

It is possible to convert a differential equation into a set of algebraic equations if the functions in the differential equation are made discrete. This allows the derivatives to be approximated using finite differences and the entire differential equation can be made discrete. The discrete form of the differential equation is written once for each point on the grid and the resulting set of equations is written as a single matrix equation. In the end, an analytical differential equation is converted into a numerical matrix equation. Converting an analytical differential equation into a matrix equation is the first half of the finite-difference method. The second half solves the matrix equation so that a solution is obtained. This section will focus on the first half of the finite-difference method. All of this work is done on paper or outside of the computer code. The final equations that come out of this work are what are used in the computer code to build the matrices that will solve the problem. When the finite-difference method is done properly, the computer code will be very simple, compact, and easy to read and understand.

Equipped with the knowledge of derivative matrices, converting a differential equation into a matrix equation becomes incredibly simple. Differential equations are converted to a matrix equation one term at a time. Each term in the differential equation becomes either a matrix or a column vector in the matrix equation. If the term is an unknown function or the excitation of a differential equation, it becomes a column vector in the matrix equation. All other terms are linear operations that become square matrices in the matrix equation.

This is best explained through an example. A generic differential equation to be converted to matrix form is specified in (3.50). In this equation, the function $f(x)$ is unknown so its discrete version is stored in the column vector \mathbf{f} . The function $b(x)$ is the excitation so its discrete version is stored in the column vector \mathbf{b} . The two remaining terms d/dx and $c(x)$ are operations and are represented by square matrices \mathbf{D}_x and \mathbf{C} . At a first glance, it may seem incorrect to call the function $c(x)$ an operation. However, it is performing the operation of a point-by-point multiplication of the known function $c(x)$ with the unknown function $f(x)$.

$$\frac{df(x)}{dx} + c(x)f(x) = b(x) \quad (3.50)$$

Given the differential equation, it is converted term-by-term into a matrix equation as shown in Figure 3.6. The derivative operation d/dx is written as a derivative matrix \mathbf{D}_x . The unknown function $f(x)$ is written as a column vector \mathbf{f} , the point-by-point multiplication with $c(x)$ is written as a square matrix \mathbf{C} , and the excitation function $b(x)$ is written as a column vector \mathbf{b} . Note that bold uppercase letters represent matrices and bold lowercase letters represent column vectors.

The final matrix equation expands to (3.51). Observe that the point-by-point multiplication matrix \mathbf{C} is a diagonal matrix with the discrete values of $c(n)$ placed along the center diagonal. All other elements in the \mathbf{C} matrix are zero. With practice, converting differential equations to matrix equations becomes nearly effortless.

$$\frac{1}{2\Delta x} \begin{bmatrix} & +1 & & \\ -1 & & \ddots & \\ & \ddots & & \\ & & -1 & +1 \end{bmatrix} \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(N) \end{bmatrix} + \begin{bmatrix} c(1) & & & \\ & c(2) & & \\ & & \ddots & \\ & & & c(N) \end{bmatrix} \begin{bmatrix} f(1) \\ f(2) \\ \vdots \\ f(N) \end{bmatrix} = \begin{bmatrix} b(1) \\ b(2) \\ \vdots \\ b(N) \end{bmatrix} \quad (3.51)$$

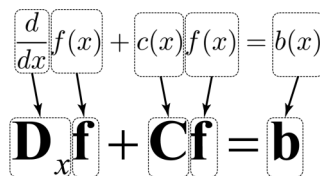


Figure 3.6 Term-by-term conversion of a differential equation into a matrix equation.

3.7 Solving Matrix Differential Equations

Before the matrix differential equation can be solved, it must be cast into the standard form of $\mathbf{A}\mathbf{f} = \mathbf{b}$. From Figure 3.6, the matrix differential equation is

$$\mathbf{D}_x \mathbf{f} + \mathbf{C}\mathbf{f} = \mathbf{b} \quad (3.52)$$

To put this in a standard form, the column vector \mathbf{f} is factored out on the left-hand side of the matrix equation.

$$(\mathbf{D}_x + \mathbf{C})\mathbf{f} = \mathbf{b} \quad (3.53)$$

Next, let $\mathbf{A} = \mathbf{D}_x + \mathbf{C}$ and the matrix equation is in standard form.

$$\mathbf{A}\mathbf{f} = \mathbf{b} \quad (3.54)$$

$$\mathbf{A} = \mathbf{D}_x + \mathbf{C} \quad (3.55)$$

Equation (3.54) is easily solved either by direct LU decomposition [1] or via an iterative technique [6, 7]. MATLAB makes obtaining a solution by either method very easy. The solution \mathbf{f} is written symbolically as

$$\mathbf{f} = \mathbf{A}^{-1}\mathbf{b} \quad (3.56)$$

It is important to note that explicitly calculating matrix inverses is quite rare in numerical methods. The matrix inverse in (3.56) represents a predivision by \mathbf{A} instead of a premultiplication by the matrix inverse \mathbf{A}^{-1} . The difference between these two interpretations is huge in terms of the number of computations required. If you are ever calculating a matrix inverse, reconsider this five or six times to ensure explicitly calculating the inverse is truly necessary. In MATLAB, the predivision is called *backward division* and is written as $\mathbf{f} = \mathbf{A} \backslash \mathbf{b}$. Do not ever solve this equation as $\mathbf{f} = \text{inv}(\mathbf{A}) * \mathbf{b}$! Following the above procedure, solving a differential equation in MATLAB is very easy. With some practice, solving an entirely new differential equation can be accomplished in mere minutes.

3.7.1 Example—Solving a Single-Variable Differential Equation

Suppose it is desired to calculate the numerical solution to the following differential equation in the range $0 \leq x \leq 10$.

$$\frac{df}{dx} = -\frac{1}{3} \quad f(0) = 1 \quad (3.57)$$

Analytically, this is easily solved by integrating the differential equation to get $f(x) = b - x/3$. The constant b is found by applying the initial condition $f(0) = 1$ to get $b = 1$. This gives the final solution to be

$$f(x) = 1 - \frac{x}{3} \quad (3.58)$$

The MATLAB code that solves (3.57) numerically can be downloaded at <https://empossible.net/fdfdbook/>. The file is named `Chapter3_fdm1d.m`. The first line simply displays this file name as a comment. Lines 3 to 6 initialize MATLAB as described in Chapter 1. Lines 8 to 13 define and calculate the grid for the numerical solution. The variables `a` and `b` define the bounds for the solution, `Nx` is the number of discrete points, `xa` is an array containing the positions of the discrete points along the x -axis, and `dx` is the spacing between the points. Lines 15 to 19 build the derivative matrix `DX` that implements the finite-difference approximation summarized in (3.59). Observe that the finite-difference equation at the last point $n = N$ is modified to calculate the derivative from the two adjacent last points separated by Δx .

$$\frac{df}{dx} \cong \begin{cases} \frac{f(n+1) - f(n-1)}{2\Delta x} & n < N \\ \frac{f(N) - f(N-1)}{\Delta x} & n = N \end{cases} \quad (3.59)$$

Line 16 creates a one-dimensional array `d` containing all values of $1/2\Delta x$ that will be used to insert into the diagonals of the derivative matrix. Line 17 inserts $-d$ along the -1 diagonal and line 18 inserts $+d$ along the $+1$ diagonal. Line 19 overwrites the last row in `DX` with the finite-difference approximation to be used only at $n = N$. At the same time, this line places $-1/dx$ at `DX(Nx, Nx-1)` and places $+1/dx$ at `DX(Nx, Nx)`. Lines 21 to 23 build the matrix equation to be solved without the initial value $f(0) = 1$ incorporated yet. The standard form of the matrix equation from (3.57) is $\mathbf{A}\mathbf{f} = \mathbf{b}$ where $\mathbf{A} = \mathbf{D}_x$ and \mathbf{b} is a column vector with all elements equal to $-1/3$. Lines 25 to 28 incorporate the boundary value $f(0) = 1$ into the matrix equation. Line 26 sets the first row in \mathbf{A} to all zeros to completely erase the finite-difference equation associated with the first point on the grid. Line 27 sets the diagonal element in the first row to 1. The value that $f(1)$ is set equal to is inserted into the first element of \mathbf{b} on line 28 to make the first discrete equation in the matrix equation $f(1) = 1$ instead of a finite-difference equation. Line 31 solves for the column vector \mathbf{f} using backward division in MATLAB. Lines 33 to 35 plot the numerical solution which is shown in Figure 3.7. This matches exactly the analytical solution derived

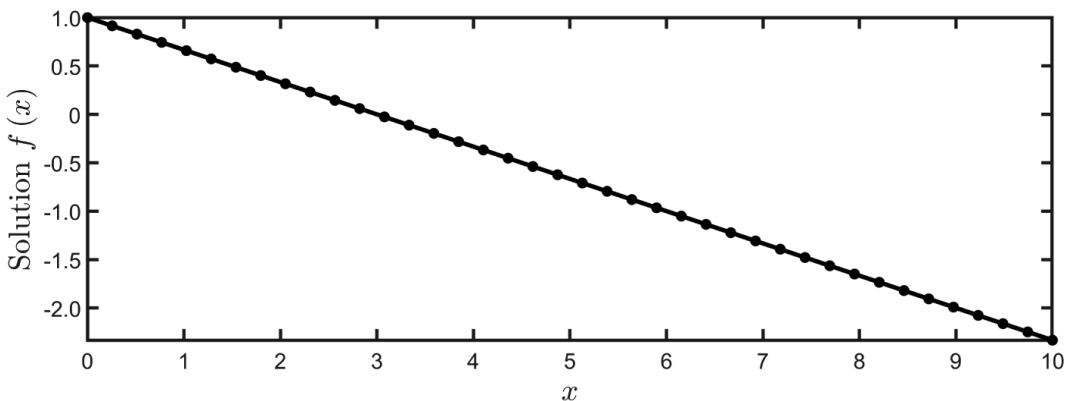


Figure 3.7 Plot of finite-difference method solution to $df/dx = -1/3$ given $f(0) = 1$.

in (3.58) and demonstrates how simple the MATLAB code can be for implementing the finite-difference method.

3.8 Multiple Variables and Staggered Grids

Suppose it is desired to solve the following system of two coupled differential equations of two unknowns $f(x)$ and $g(x)$ using the finite-difference method.

$$\frac{df(x)}{dx} + u(x)g(x) = b_1(x) \quad (3.60)$$

$$\frac{dg(x)}{dx} + v(x)f(x) = b_2(x) \quad (3.61)$$

It is entirely possible to solve this system of equations using only the concepts taught previously. However, there is a better way to handle the finite-difference approximations that gives greater accuracy and improves numerical efficiency. From Figure 3.2, it was clear that more closely spaced points will more accurately estimate a derivative. The finite-difference approximations up to now spanned $2\Delta_x$ across the grid as illustrated in Figure 3.2. Equation (3.62) shows a way to calculate a finite-difference that only spans Δ_x across the grid. This greatly improves accuracy, but the result from this derivative is defined at the midpoint. Interpreting the finite-difference approximation in (3.62) as a central finite difference, the derivative is estimated at the location $n + 1/2$, which is midway between the grid points n and $n + 1$.

$$\frac{df(n + 1/2)}{dx} \approx \frac{f(n + 1) - f(n)}{\Delta x} \quad (3.62)$$

Applying this concept to (3.60) gives a discrete differential equation of the following form.

$$\frac{f(n + 1) - f(n)}{\Delta x} + u(n)g(n) \stackrel{?}{=} b_1(n) \quad (3.63)$$

Equation (3.63) has a serious problem that can lead to instability and inaccuracy of the results. The terms $u(n)g(n)$ and $b_1(n)$ give values defined at n , but the finite-difference approximation gives a value defined at $n + 1/2$. All terms in a finite-difference equation should be defined at the same point in space. One possible way to correct this is to use interpolation to calculate the second two terms in (3.63) at the point $n + 1/2$.

$$\frac{f(n + 1) - f(n)}{\Delta x} + \frac{u(n + 1)g(n + 1) + u(n)g(n)}{2} = \frac{b_1(n + 1) + b_1(n)}{2} \quad (3.64)$$

This can be made to work, but it involves more calculations than is necessary and there is a better way. Instead, the discrete functions f , v , and b_2 are defined as usual at points 1, 2, 3, ..., N on the grid. However, the discrete functions g , u , and

b_1 are defined at midpoints 1.5, 2.5, 3.5, ..., $N + 0.5$ on the grid. Given the staggering, the discrete form of the two differential equations becomes

$$\frac{f(n+1) - f(n)}{\Delta x} + u\left(n + \frac{1}{2}\right)g\left(n + \frac{1}{2}\right) = b_1\left(n + \frac{1}{2}\right) \quad (3.65)$$

$$\frac{g\left(n + \frac{1}{2}\right) - g\left(n - \frac{1}{2}\right)}{\Delta x} + v(n)f(n) = b_2(n) \quad (3.66)$$

These equations have the simplicity of (3.63) but provide accuracy exceeding anything else because the finite-difference approximations only span a single Δx . The notation of using array indices like $n + 1/2$ can be confusing. In this book, the array indices will only ever be written as integers to match the array indices used in the computer codes. The reader will have to remember which functions are stored at the offset locations. In addition, the following notation will be adopted for discrete functions to make the discrete equations more compact and more clearly identify which terms are discrete.

$$\frac{f|_{n+1} - f|_n}{\Delta x} + u|_n g|_n = b_1|_n \quad (3.67)$$

$$\frac{g|_{n+1} - g|_n}{\Delta x} + v|_n f|_n = b_2|_n \quad (3.68)$$

Two discrete functions $f|_n$ and $g|_n$ staggered across a grid is illustrated in Figure 3.8. Even though $f|_n$ and $g|_n$ have the same array index n , observe that they are located $\Delta x/2$ apart from each other.

To convert to matrix form, (3.67) is written once for every point on the grid, and this set of N equations is written in matrix form in (3.69). Similarly, (3.68) is written once for every point on the grid, and this second set of N equations is written in matrix form in (3.70).

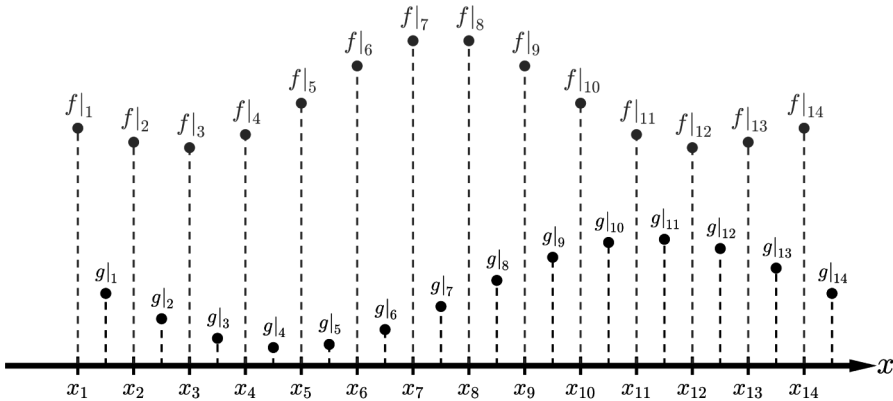


Figure 3.8 The position of two discrete functions is staggered across the grid.

$$\mathbf{D}_x^f \mathbf{f} + \mathbf{U} \mathbf{g} = \mathbf{b}_1 \quad (3.69)$$

$$\mathbf{D}_x^g \mathbf{g} + \mathbf{V} \mathbf{f} = \mathbf{b}_2 \quad (3.70)$$

The superscripts on the derivative matrices identify for which function they calculate the derivatives. Due to the staggering of the discrete functions, the derivative matrices for \mathbf{f} and \mathbf{g} are different from each other. Following the procedure outlined previously in this chapter, the contents of the derivative matrices can be determined. For the present example, the derivative matrices incorporating Dirichlet boundary conditions are provided in (3.71) and (3.72).

$$\frac{1}{\Delta x} \begin{bmatrix} -1 & +1 & & & \\ & -1 & \ddots & & \\ & & \ddots & +1 & \\ & & & -1 & \\ & & & & -1 \end{bmatrix} \begin{bmatrix} f|_1 \\ f|_2 \\ \vdots \\ f|_N \end{bmatrix} + \begin{bmatrix} u_1 & & & \\ & u_2 & & \\ & & \ddots & \\ & & & u_N \end{bmatrix} \begin{bmatrix} g|_1 \\ g|_2 \\ \vdots \\ g|_N \end{bmatrix} = \begin{bmatrix} b_1|_1 \\ b_1|_2 \\ \vdots \\ b_1|_N \end{bmatrix} \quad (3.71)$$

$$\frac{1}{\Delta x} \begin{bmatrix} +1 & & & & \\ -1 & +1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & -1 & +1 \end{bmatrix} \begin{bmatrix} g|_1 \\ g|_2 \\ \vdots \\ g|_N \end{bmatrix} + \begin{bmatrix} v_1 & & & \\ & v_2 & & \\ & & \ddots & \\ & & & v_N \end{bmatrix} \begin{bmatrix} f|_1 \\ f|_2 \\ \vdots \\ f|_N \end{bmatrix} = \begin{bmatrix} b_2|_1 \\ b_2|_2 \\ \vdots \\ b_2|_N \end{bmatrix} \quad (3.72)$$

Observe the difference between the derivative matrices \mathbf{D}_x^f and \mathbf{D}_x^g . While they look similar, the +1's and -1's are inserted along different diagonals. When Dirichlet or PBCs are used, the derivative matrices can be related through

$$\mathbf{D}_x^g = -(\mathbf{D}_x^f)^H \quad (3.73)$$

In this equation, the superscript H indicates a Hermitian transpose, which is an ordinary transpose followed by calculating the complex conjugate of each element in the matrix. This relation means that after one derivative matrix is constructed, the other can be calculated immediately from it, simplifying the computer code. It may seem strange to use a complex conjugate when the derivative matrices contain only real numbers. In Chapter 4, PBCs will involve complex numbers in order to incorporate a source at an oblique angle of incidence. Be careful because the relation in (3.73) does not hold for all boundary conditions!

After arriving at (3.69) and (3.70), there are two options for solving the system of equations. The least preferred is to combine the two matrix equations into a single block matrix equation where it assumes the standard form.

$$\begin{bmatrix} \mathbf{D}_x^f & \mathbf{U} \\ \mathbf{V} & \mathbf{D}_x^g \end{bmatrix} \begin{bmatrix} \mathbf{f} \\ \mathbf{g} \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} \quad (3.74)$$

This equation can then be solved using standard matrix division algorithms [1]. In this book, the preferred solution method utilizes smaller matrices so it tends to be faster and more memory efficient. First, (3.69) is solved for \mathbf{g} .

$$\mathbf{g} = \mathbf{U}^{-1}(\mathbf{b}_1 - \mathbf{D}_x^f \mathbf{f}) \quad (3.75)$$

Second, this expression for \mathbf{g} is substituted into (3.70) to get a single equation in terms of just \mathbf{f} .

$$\mathbf{D}_x^g \mathbf{U}^{-1}(\mathbf{b}_1 - \mathbf{D}_x^f \mathbf{f}) + \mathbf{V} \mathbf{f} = \mathbf{b}_2 \quad (3.76)$$

Third, (3.76) is put in standard form $\mathbf{A} \mathbf{f} = \mathbf{b}$ and solved for \mathbf{f} .

$$\mathbf{f} = \mathbf{A}^{-1} \mathbf{b} \quad (3.77)$$

$$\mathbf{A} = \mathbf{V} - \mathbf{D}_x^g \mathbf{U}^{-1} \mathbf{D}_x^f \quad \mathbf{b} = \mathbf{b}_2 - \mathbf{D}_x^g \mathbf{U}^{-1} \mathbf{b}_1 \quad (3.78)$$

Given the solution \mathbf{f} , the solution for \mathbf{g} is obtained afterward using (3.75). This last step is an example where it is good practice to perform numerical differentiation $\mathbf{D}_x^f \mathbf{f}$ using a derivative matrix. The derivative matrix was already calculated for another purpose.

3.8.1 Example—Solving a Multivariable Problem

Suppose it is desired to solve the following set of coupled differential equations within the interval $0 \leq x \leq 10$.

$$\frac{df(x)}{dx} + 3g(x) = 0 \quad (3.79)$$

$$\frac{dg(x)}{dx} - 2f(x) = 0 \quad (3.80)$$

$$f(0) = 10 \quad f(10) = 1 \quad (3.81)$$

The first step is to write the differential equations in matrix form using the concept of derivative matrices. By inspection, (3.79) and (3.80) become

$$\mathbf{D}_x^f \mathbf{f} + 3\mathbf{g} = \mathbf{0} \quad (3.82)$$

$$\mathbf{D}_x^g \mathbf{g} - 2\mathbf{f} = \mathbf{0} \quad (3.83)$$

Equation (3.82) is solved for \mathbf{g} to get

$$\mathbf{g} = -\frac{1}{3} \mathbf{D}_x^f \mathbf{f} \quad (3.84)$$

This expression for \mathbf{g} is substituted into (3.83) to get

$$\mathbf{D}_x^g \left(-\frac{1}{3} \mathbf{D}_x^f \mathbf{f} \right) - 2\mathbf{f} = \mathbf{0} \quad (3.85)$$

Equation (3.85) is cast into the standard form of $\mathbf{A}\mathbf{f} = \mathbf{0}$.

$$(\mathbf{D}_x^g \mathbf{D}_x^f + 6\mathbf{I})\mathbf{f} = \mathbf{0} \quad (3.86)$$

This is as far as the problem can be taken on paper. The rest is handled in MATLAB and the code to obtain and plot the solution can be downloaded at <https://empossible.net/fdfdbook/>. The filename is `Chapter3_mvfdm.m`. Line 1 is a comment with this name of the program. Lines 3 to 6 initialize MATLAB. The grid for the problem is calculated on lines 8 to 13 and is the same as the single variable example covered in Section 3.7.1. The derivative matrices for the staggered grid are constructed on lines 15 to 22. Line 16 builds a one-dimensional array containing $1/\Delta x$ throughout the entire array. Lines 18 to 20 build the derivative matrix \mathbf{D}_x^f that operates on the function $f(x)$. In MATLAB, the derivative matrix is given the variable name `DFX`. Line 18 initializes `DFX` as a sparse matrix of size $N_x \times N_x$. Line 19 adds the center diagonal containing all $-1/\Delta x$. Line 20 adds the upper diagonal containing all $+1/\Delta x$. After `DFX` is constructed correctly, the derivative matrix \mathbf{D}_x^g is calculated immediately from it using the relation in (3.73) on line 22. The new derivative matrix is given the variable name `DGX`.

Given the derivative matrices, the matrix equation in (3.86) is constructed on lines 24 to 27. The boundary values for $f(x)$ are incorporated into the matrix equation on lines 29 to 36. Lines 30 to 32 replace the finite-difference equation associated with the first point on the grid with the equation $f|_1 = 10$. Lines 34 to 36 replace the finite-difference equation associated with the last point on the grid with the equation $f|_{N_x} = 1$. The matrix equation $\mathbf{A}\mathbf{f} = \mathbf{b}$ is solved on line 39. Given the solution for \mathbf{f} , the solution for \mathbf{g} is calculated immediately from \mathbf{f} using (3.84). This happens on line 40.

Lines 42 to 51 visualize the results. A legend is provided to distinguish the lines. The final plot from this analysis is provided in Figure 3.9. The answer looks

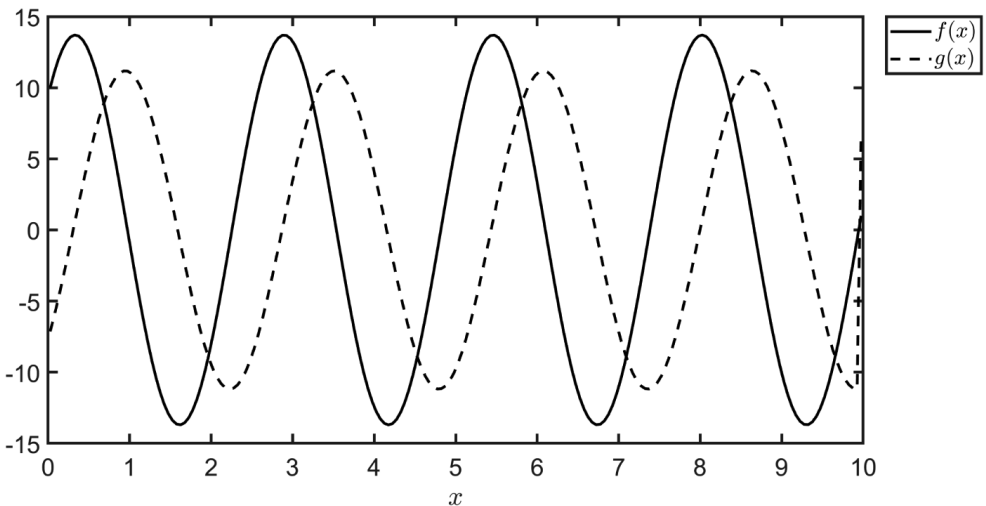


Figure 3.9 Results obtained from multivariable finite-difference analysis.

like perfect sine waves. While no analytical solution is obtained using the finite-difference method, the results give a huge hint about the answer when solving differential equations analytically.

References

- [1] Chapra, S. C., and R. P. Canale, *Numerical Methods for Engineers*, Seventh Edition, New York: McGraw-Hill Education, 2015.
- [2] Klinger, A., "The Vandermonde Matrix," *The American Mathematical Monthly*, Vol. 74, No. 5, 1967, pp. 571–574.
- [3] Rushanan, J. J., "On the Vandermonde Matrix," *The American Mathematical Monthly*, Vol. 96, No. 10, 1989, pp. 921–924.
- [4] Cheng, A. H.-D., and D. T. Cheng, "Heritage and Early History of the Boundary Element Method," *Engineering Analysis with Boundary Elements*, Vol. 29, No. 3, 2005, pp. 268–302.
- [5] Harms, P., R. Mittra, and W. Ko, "Implementation of the Periodic Boundary Condition in the Finite-Difference Time-Domain Algorithm for FSS Structures," *IEEE Trans. on Antennas and Propagation*, Vol. 42, No. 9, 1994, pp. 1317–1324.
- [6] Greenbaum, A., *Iterative Methods for Solving Linear Systems*: SIAM, 1997.
- [7] Saad, Y., *Iterative Methods for Sparse Linear Systems*: SIAM, 2003.

Finite-Difference Approximation of Maxwell's Equations

This chapter will explain how Maxwell's equations are converted into matrix equations using the finite-difference method. The chapter will show how to prepare Maxwell's equations for finite-difference frequency-domain (FDFD) by normalizing the fields and grid coordinates so they are all the same order of magnitude and as close to the numerical value of one as possible. The analytical functions in Maxwell's equations are made discrete according to the Yee grid scheme [1], allowing the derivatives to be approximated with simple and accurate finite differences. After this is done for full three-dimensional equations, the equations are reduced to two dimensions. The equations are then expressed in matrix form in preparation for being solved by the various FDFD methods described in following chapters. The derivative matrices used in FDFD are discussed in detail. A MATLAB function is presented to build the derivative matrices for the two-dimensional FDFD analyses described in Chapters 6 to 9. A second MATLAB function is presented to build the derivative matrices for the three-dimensional FDFD analyses described in Chapter 10.

4.1 Introduction to the Yee Grid Scheme

In Chapter 2, it was shown that it is possible to reduce the number of electromagnetic field components that have to be solved down to six. These were E_x , E_y , E_z , H_x , H_y , and H_z . In Chapter 3, it was shown how the finite-difference method can be used to solve differential equations. It was also shown that when multiple functions are to be solved, the discrete values should be staggered in order to make use of tighter finite differences. The manner in which the field components should be staggered can be determined by carefully inspecting the equations to be solved. From Chapter 2, Maxwell's curl equations for diagonally anisotropic media expand into the following set of six coupled partial differential equations.

$$\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} = -j\omega\mu_{xx}H_x \quad (4.1)$$

$$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = -j\omega\mu_{yy}H_y \quad (4.2)$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = -j\omega\mu_{zz}H_z \quad (4.3)$$

$$\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} = j\omega\epsilon_{xx}E_x \quad (4.4)$$

$$\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} = j\omega\epsilon_{yy}E_y \quad (4.5)$$

$$\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} = j\omega\epsilon_{zz}E_z \quad (4.6)$$

Examining (4.1), it will be best to define the positions of the discrete values of E_y and E_z to be immediately on either side of the discrete values of H_x . This will allow the partial derivatives to be estimated using the simplest and most accurate central finite-difference approximations. It will also be most convenient to define the discrete tensor element μ_{xx} to be at the same locations as H_x . A similar argument can be made for (4.1) to (4.6). E_x and E_z will be defined to be on either side of H_y . E_x and E_y will be defined to be on either side of H_z . H_y and H_z will be defined to be on either side of E_x . H_x and H_z will be defined to be on either side of E_y . Last, H_x and H_y will be defined to be on either side of E_z . Furthermore, ϵ_{yy} will be defined at the same points as E_y , ϵ_{zz} will be defined at the same points as E_z , μ_{xx} will be defined at the same points as H_x , μ_{yy} will be defined at the same points as H_y , and μ_{zz} will be defined at the same points as H_z . This will present some challenges when devices are built onto the grid because material interfaces that slice through the middle of Yee cells will place some field components inside of one medium and the other field components inside of another. Later in this chapter, the $2\times$ grid technique will be presented as an easy way to assign materials to the Yee grid that naturally handles the staggering.

The arrangement of field components that satisfies all of the above was first proposed by Yee [1] and is now called the Yee grid scheme. In this scheme, a Cartesian space is divided into many tiny cells and the six field components are staggered the same way within each cell. A single cell of the standard Yee grid is illustrated in Figure 4.1. The integers i , j , and k represent the array indices to access the fields stored in memory. Even though the six field components have the same array indices, they are at physically different locations within the cell. This will have important implications in how materials will be assigned to the grid, how sources will be injected, and how the fields will be postprocessed after the simulation is over. While E_x , E_y , and E_z are components of a single vector quantity, they will be out of phase because they are at different locations within the Yee cell. If the vector components ever need to be combined, they must be interpolated at a common point within the Yee cell before combining them. The origin of the Yee cell will be used in this book as the common point. Even more, a material interface that slices through the middle of the Yee cell may place the different field components within the same cell in different mediums, making the arrays storing the constitutive parameters different for each field component.

In addition to being the most efficient arrangement for finite-difference approximations, the Yee grid scheme offers additional benefits. These include the grid being divergence-free and physical boundary conditions being naturally satisfied [2]. By simply staggering the positions of the field components according to the

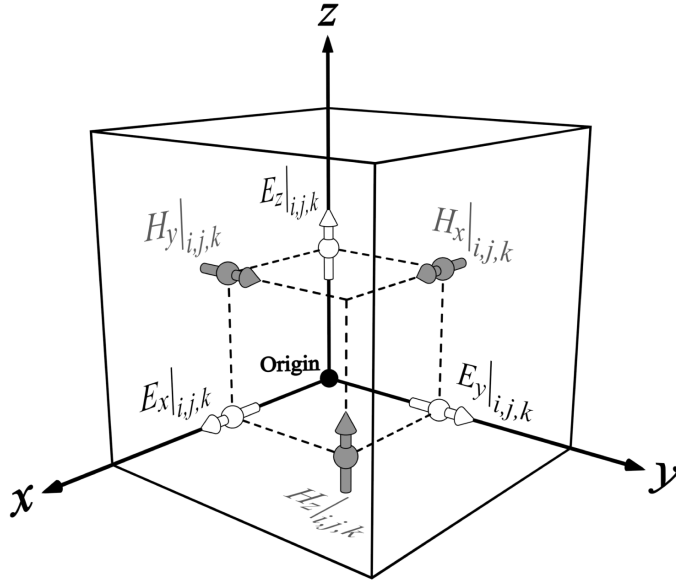


Figure 4.1 The three-dimensional Yee grid cell.

Yee grid scheme, Maxwell's divergence equations are satisfied so they do not need to be explicitly handled by the FDFD method. In addition, no special treatment at the interface between two materials is needed in order to obtain an accurate and stable simulation.

4.2 Preparing Maxwell's Equations for FDFD Analysis

From Chapter 2, the final form of Maxwell's equations for isotropic media was

$$\nabla \cdot (\epsilon \vec{E}) = 0 \quad (4.7)$$

$$\nabla \cdot (\mu \vec{H}) = 0 \quad (4.8)$$

$$\nabla \times \vec{E} = -j\omega\mu\vec{H} \quad (4.9)$$

$$\nabla \times \vec{H} = j\omega\epsilon\vec{E} \quad (4.10)$$

In these equations, it should be understood that both the permittivity ϵ and permeability μ can be complex quantities even though the tilde notation is dropped. When discussing material impedance η in Chapter 2, it was observed that the electric and magnetic fields are numerically around three orders of magnitude different. It is best practice in computation to normalize all of the functions and parameters so they are all of the same order of magnitude and are as close to the numerical value of 1 as possible. In this book, the magnetic field will be normalized according to

$$\tilde{\vec{H}} = -j\eta_0\vec{H} \quad (4.11)$$

The free space impedance η_0 is incorporated to make the normalized magnetic field \tilde{H} the same order of magnitude as \tilde{E} . The $-j$ is incorporated to simply give the curl equations a more convenient symmetry. Using the normalized magnetic field \tilde{H} , the free space permittivity ϵ_0 and free space permeability μ_0 are eliminated from (4.7) to (4.10) and become

$$\nabla \cdot (\epsilon_r \tilde{E}) = 0 \quad (4.12)$$

$$\nabla \cdot (\mu_r \tilde{H}) = 0 \quad (4.13)$$

$$\nabla \times \tilde{E} = k_0 \mu_r \tilde{H} \quad (4.14)$$

$$\nabla \times \tilde{H} = k_0 \epsilon_r \tilde{E} \quad (4.15)$$

Equation (4.7) was divided by the free space permittivity ϵ_0 to arrive at (4.12), and (4.8) was divided by the free space permeability μ_0 to arrive at (4.13). With this normalization applied to Maxwell's equations, there is no need to remember which of the curl equations has the negative sign! The curl equations with normalized magnetic field \tilde{H} expand into the following set of six coupled partial differential equations when diagonally anisotropic media is assumed.

$$\frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} = k_0 \mu_{xx} \tilde{H}_x \quad (4.16)$$

$$\frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} = k_0 \mu_{yy} \tilde{H}_y \quad (4.17)$$

$$\frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} = k_0 \mu_{zz} \tilde{H}_z \quad (4.18)$$

$$\frac{\partial \tilde{H}_z}{\partial y} - \frac{\partial \tilde{H}_y}{\partial z} = k_0 \epsilon_{xx} E_x \quad (4.19)$$

$$\frac{\partial \tilde{H}_x}{\partial z} - \frac{\partial \tilde{H}_z}{\partial x} = k_0 \epsilon_{yy} E_y \quad (4.20)$$

$$\frac{\partial \tilde{H}_y}{\partial x} - \frac{\partial \tilde{H}_x}{\partial y} = k_0 \epsilon_{zz} E_z \quad (4.21)$$

This will be the final form of the analytical equations for the FDFD method when the frequency is not known until the simulation calculates it. When the frequency is not known at the start of the simulation, the free space wavenumber k_0 must be retained as a variable to be calculated by the FDFD algorithm. This situation occurs in photonic band calculations and will be covered in Chapter 7. In other cases, the frequency is known at the start of the simulation and a numerical value can be assigned to k_0 . In this case, it is best to normalize the spatial coordinates x , y , and z by dividing them by the free space wavelength λ_0 . Since $k_0 = 2\pi/\lambda_0$, the grid

coordinates are divided by λ_0 by multiplying them by k_0 . With this normalization, all dimensions will be as close to the numerical value of 1 as possible and will not have any units. The normalized coordinates x' , y' , and z' are defined as

$$x' = k_0 x \quad y' = k_0 y \quad z' = k_0 z \quad (4.22)$$

While this normalization incorporates an unnecessary factor of 2π , it is convenient to include in order to perfectly cancel the k_0 term from the curl equations, making them simpler equations. Using normalized spatial coordinates in addition to normalized \tilde{H} , (4.16) to (4.21) become the following set of six coupled partial differential equations that are used when frequency is known at the start of the simulation. This will be the case for waveguide analysis covered in Chapter 6 and scattering analysis covered in Chapters 8–10.

$$\frac{\partial E_z}{\partial y'} - \frac{\partial E_y}{\partial z'} = \mu_{xx} \tilde{H}_x \quad (4.23)$$

$$\frac{\partial E_x}{\partial z'} - \frac{\partial E_z}{\partial x'} = \mu_{yy} \tilde{H}_y \quad (4.24)$$

$$\frac{\partial E_y}{\partial x'} - \frac{\partial E_x}{\partial y'} = \mu_{zz} \tilde{H}_z \quad (4.25)$$

$$\frac{\partial \tilde{H}_z}{\partial y'} - \frac{\partial \tilde{H}_y}{\partial z'} = \epsilon_{xx} E_x \quad (4.26)$$

$$\frac{\partial \tilde{H}_x}{\partial z'} - \frac{\partial \tilde{H}_z}{\partial x'} = \epsilon_{yy} E_y \quad (4.27)$$

$$\frac{\partial \tilde{H}_y}{\partial x'} - \frac{\partial \tilde{H}_x}{\partial y'} = \epsilon_{zz} E_z \quad (4.28)$$

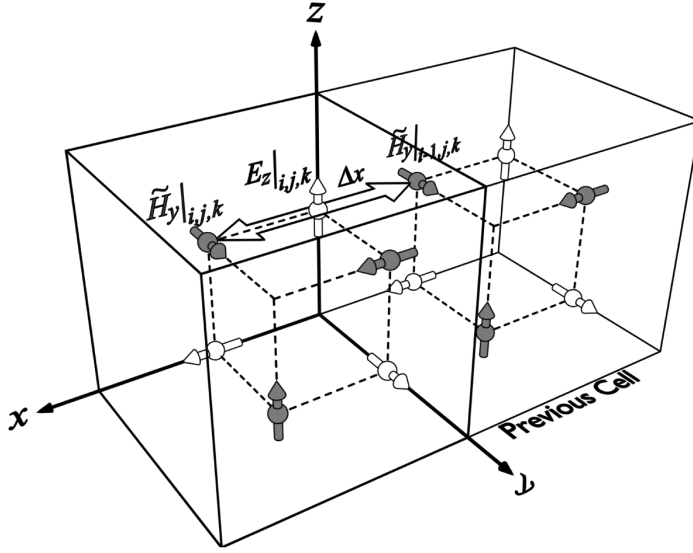
4.3 Finite-Difference Approximation of Maxwell's Curl Equations

Given the Yee grid scheme, (4.16) to (4.21) can be written in discrete form by estimating the partial derivatives with central finite differences. To demonstrate, start with (4.21) where the two partial derivatives to estimate with finite differences are $\partial \tilde{H}_y / \partial x$ and $\partial \tilde{H}_x / \partial y$. The manner in which this equation is made discrete is illustrated in Figure 4.2. All of the discrete terms will be defined at the same location as $E_z|_{i,j,k}$ in the Yee grid. The relative permittivity term $\epsilon_{zz}|_{i,j,k}$ should be defined at the same point as $E_z|_{i,j,k}$. For this reason, the discrete representation of the term on the right side of (4.28) is written as $k_0 \epsilon_{zz}|_{i,j,k} E_z|_{i,j,k}$.

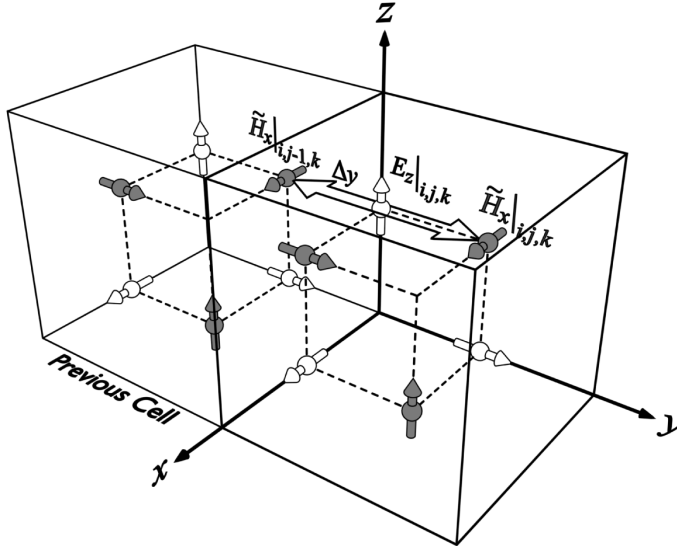
The first partial derivative in (4.21) is $\partial \tilde{H}_y / \partial x$ and its finite-difference approximation is illustrated in Figure 4.2(a). To ensure the central finite-difference estimates, the derivative at the same location as $E_z|_{i,j,k}$, magnetic field terms that lie

symmetrically on either side of $E_z|_{i,j,k}$ should be used. These terms are the magnetic field term $\tilde{H}_y|_{i,j,k}$ from the same cell as $E_z|_{i,j,k}$ and the magnetic field term $\tilde{H}_y|_{i-1,j,k}$ from the previous cell in the x -direction. The finite-difference approximation for $\partial\tilde{H}_y/\partial x$ is therefore

$$\frac{\partial\tilde{H}_y}{\partial x} \approx \frac{\tilde{H}_y|_{i,j,k} - \tilde{H}_y|_{i-1,j,k}}{\Delta x} \quad (4.29)$$



(a) Finite-difference approximation of $\partial\tilde{H}_y/\partial x$.



(b) Finite-difference approximation of $\partial\tilde{H}_x/\partial y$.

Figure 4.2 (a) Illustration of the finite-difference approximation of $\partial\tilde{H}_y/\partial x$. (b) Illustration of the finite-difference approximation of $\partial\tilde{H}_x/\partial y$.

The second partial derivative in (4.21) is $\partial \tilde{H}_x / \partial y$ and its finite-difference approximation is illustrated in Figure 4.2(b). The magnetic field terms that lie symmetrically on either side of $E_z|_{i,j,k}$ are $\tilde{H}_x|_{i,j,k}$ and $\tilde{H}_x|_{i,j-1,k}$. The finite-difference approximation for $\partial \tilde{H}_x / \partial y$ is therefore

$$\frac{\partial \tilde{H}_x}{\partial y} \approx \frac{\tilde{H}_x|_{i,j,k} - \tilde{H}_x|_{i,j-1,k}}{\Delta y} \quad (4.30)$$

All finite-difference approximations of derivatives of magnetic field terms will reach to a previous cell for the second magnetic field term. When applying this same reasoning to finite-difference approximations of derivatives of electric field terms, however, the finite-difference approximations will reach to a next cell in the positive direction along the axes. This difference is due to the staggered layout of the field components on the Yee grid. Applying these steps to all of (4.16) to (4.21), the discrete equations are

$$\frac{E_z|_{i,j+1,k} - E_z|_{i,j,k}}{\Delta y} - \frac{E_y|_{i,j,k+1} - E_y|_{i,j,k}}{\Delta z} = k_0 \mu_{xx}|_{i,j,k} \tilde{H}_x|_{i,j,k} \quad (4.31)$$

$$\frac{E_x|_{i,j,k+1} - E_x|_{i,j,k}}{\Delta z} - \frac{E_z|_{i+1,j,k} - E_z|_{i,j,k}}{\Delta x} = k_0 \mu_{yy}|_{i,j,k} \tilde{H}_y|_{i,j,k} \quad (4.32)$$

$$\frac{E_y|_{i+1,j,k} - E_y|_{i,j,k}}{\Delta x} - \frac{E_x|_{i,j+1,k} - E_x|_{i,j,k}}{\Delta y} = k_0 \mu_{zz}|_{i,j,k} \tilde{H}_z|_{i,j,k} \quad (4.33)$$

$$\frac{\tilde{H}_z|_{i,j,k} - \tilde{H}_z|_{i,j-1,k}}{\Delta y} - \frac{\tilde{H}_y|_{i,j,k} - \tilde{H}_y|_{i,j,k-1}}{\Delta z} = k_0 \epsilon_{xx}|_{i,j,k} E_x|_{i,j,k} \quad (4.34)$$

$$\frac{\tilde{H}_x|_{i,j,k} - \tilde{H}_x|_{i,j,k-1}}{\Delta z} - \frac{\tilde{H}_z|_{i,j,k} - \tilde{H}_z|_{i-1,j,k}}{\Delta x} = k_0 \epsilon_{yy}|_{i,j,k} E_y|_{i,j,k} \quad (4.35)$$

$$\frac{\tilde{H}_y|_{i,j,k} - \tilde{H}_y|_{i-1,j,k}}{\Delta x} - \frac{\tilde{H}_x|_{i,j,k} - \tilde{H}_x|_{i,j-1,k}}{\Delta y} = k_0 \epsilon_{zz}|_{i,j,k} E_z|_{i,j,k} \quad (4.36)$$

A strict rule for finite-difference equations is that every term in the equation must be defined at the same point in space. The Yee grid is staggered in a way that ensures this condition is met for all field components. Recall that for the relative permeability terms, $\mu_{xx}|_{i,j,k}$ is defined at the same points as $\tilde{H}_x|_{i,j,k}$, $\mu_{yy}|_{i,j,k}$ is defined at the same points as $\tilde{H}_y|_{i,j,k}$, and $\mu_{zz}|_{i,j,k}$ is defined at the same points as $\tilde{H}_z|_{i,j,k}$. Similarly for the relative permittivity terms, $\epsilon_{xx}|_{i,j,k}$ is defined at the same points as $E_x|_{i,j,k}$, $\epsilon_{yy}|_{i,j,k}$ is defined at the same points as $E_y|_{i,j,k}$, and $\epsilon_{zz}|_{i,j,k}$ must be defined at the same points as $E_z|_{i,j,k}$.

Focus for a moment on just (4.31). This equation is written once for every cell on the grid, leading to a large set of finite-difference equations. The set of equations can

be written as a single matrix equation following the concepts presented in Chapter 3 as $\mathbf{D}_y^e \mathbf{e}_z - \mathbf{D}_z^e \mathbf{e}_y = k_0 \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x$. The same can be said for each of (4.31) to (4.36). This gives the following set of six matrix equations that are used to formulate FDFD when frequency is not known, and therefore, k_0 is retained as a variable in the equations.

$$\mathbf{D}_y^e \mathbf{e}_z - \mathbf{D}_z^e \mathbf{e}_y = k_0 \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (4.37)$$

$$\mathbf{D}_z^e \mathbf{e}_x - \mathbf{D}_x^e \mathbf{e}_z = k_0 \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (4.38)$$

$$\mathbf{D}_x^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = k_0 \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (4.39)$$

$$\mathbf{D}_y^h \tilde{\mathbf{h}}_z - \mathbf{D}_z^h \tilde{\mathbf{h}}_y = k_0 \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (4.40)$$

$$\mathbf{D}_z^h \tilde{\mathbf{h}}_x - \mathbf{D}_x^h \tilde{\mathbf{h}}_z = k_0 \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (4.41)$$

$$\mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{D}_y^h \tilde{\mathbf{h}}_x = k_0 \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (4.42)$$

In these equations, \mathbf{D}_x^e , \mathbf{D}_y^e , \mathbf{D}_z^e , \mathbf{D}_x^h , \mathbf{D}_y^h , and \mathbf{D}_z^h are derivative matrices for calculating numerical derivatives of the field quantities across the Yee grid. Due to the staggered grid, the derivative matrices are different for electric and magnetic fields. The “e” and “h” superscripts indicate which field type the derivative matrix operates on. The terms \mathbf{e}_x , \mathbf{e}_y , and \mathbf{e}_z are column vectors containing the electric field components throughout the Yee grid reshaped into one-dimensional arrays. The terms $\tilde{\mathbf{h}}_x$, $\tilde{\mathbf{h}}_y$, and $\tilde{\mathbf{h}}_z$ are column vectors containing the normalized magnetic field components throughout the Yee grid reshaped into one-dimensional arrays. The terms $\boldsymbol{\epsilon}_{xx}$, $\boldsymbol{\epsilon}_{yy}$, $\boldsymbol{\epsilon}_{zz}$, $\boldsymbol{\mu}_{xx}$, $\boldsymbol{\mu}_{yy}$, and $\boldsymbol{\mu}_{zz}$ are diagonal matrices containing the relative permittivity and permeability, respectively, throughout the Yee grid. They are formed by reshaping the materials arrays into one-dimensional arrays and then placing those one-dimensional arrays along the center diagonal of sparse matrices. Even for isotropic materials, the materials matrices can be slightly different from each other due to the staggering of the field components on the grid.

When the frequency is known at the start of the simulation and a numerical value can be assigned to the free space wavenumber k_0 , (4.23) to (4.28) can immediately be written in matrix form as

$$\mathbf{D}_y^e \mathbf{e}_z - \mathbf{D}_z^e \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (4.43)$$

$$\mathbf{D}_z^e \mathbf{e}_x - \mathbf{D}_x^e \mathbf{e}_z = \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (4.44)$$

$$\mathbf{D}_x^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (4.45)$$

$$\mathbf{D}_y^h \tilde{\mathbf{h}}_z - \mathbf{D}_z^h \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (4.46)$$

$$\mathbf{D}_z^h \tilde{\mathbf{h}}_x - \mathbf{D}_x^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (4.47)$$

$$\mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{D}_y^h \tilde{\mathbf{h}}_x = \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (4.48)$$

In these equations, the terms have the same definitions as they did for (4.37) to (4.42). The only difference is that the free space wavenumber k_0 has been absorbed into the derivative matrices through the normalized grid coordinates defined in (4.22).

4.4 Finite-Difference Equations for Two-Dimensional FDFD

In Chapter 2, it was shown that it is possible to reduce the math of some three-dimensional problems to two dimensions when: (1) the device is uniform along z , (2) wave propagation is restricted to the xy plane, and (3) materials are isotropic or diagonally anisotropic. When these conditions are met, all derivatives with respect to z are zero and (4.31) to (4.36) reduce to

$$\frac{E_z|_{i,j+1} - E_z|_{i,j}}{\Delta y} = k_0 \mu_{xx}|_{i,j} \tilde{H}_x|_{i,j} \quad (4.49)$$

$$-\frac{E_z|_{i+1,j} - E_z|_{i,j}}{\Delta x} = k_0 \mu_{yy}|_{i,j} \tilde{H}_y|_{i,j} \quad (4.50)$$

$$\frac{E_y|_{i+1,j} - E_y|_{i,j}}{\Delta x} - \frac{E_x|_{i,j+1} - E_x|_{i,j}}{\Delta y} = k_0 \mu_{zz}|_{i,j} \tilde{H}_z|_{i,j} \quad (4.51)$$

$$\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i,j-1}}{\Delta y} = k_0 \epsilon_{xx}|_{i,j} E_x|_{i,j} \quad (4.52)$$

$$-\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i-1,j}}{\Delta x} = k_0 \epsilon_{yy}|_{i,j} E_y|_{i,j} \quad (4.53)$$

$$\frac{\tilde{H}_y|_{i,j} - \tilde{H}_y|_{i-1,j}}{\Delta x} - \frac{\tilde{H}_x|_{i,j} - \tilde{H}_x|_{i,j-1}}{\Delta y} = k_0 \epsilon_{zz}|_{i,j} E_z|_{i,j} \quad (4.54)$$

Observe the k array index was dropped because there is no longer a z dimension to be considered in the analysis. This is fortunate because the same symbol k is being used for the wavenumber!

When the frequency is known at the start of the simulation, the k_0 term will have a numerical value and is absorbed into the grid coordinates to normalize them according to (4.22). When this is done, (4.49) to (4.54) become

$$\frac{E_z|_{i,j+1} - E_z|_{i,j}}{\Delta y'} = \mu_{xx}|_{i,j} \tilde{H}_x|_{i,j} \quad (4.55)$$

$$-\frac{E_z|_{i+1,j} - E_z|_{i,j}}{\Delta x'} = \mu_{yy}|_{i,j} \tilde{H}_y|_{i,j} \quad (4.56)$$

$$\frac{E_y|_{i+1,j} - E_y|_{i,j}}{\Delta x'} - \frac{E_x|_{i,j+1} - E_x|_{i,j}}{\Delta y'} = \mu_{zz}|_{i,j} \tilde{H}_z|_{i,j} \quad (4.57)$$

$$\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i,j-1}}{\Delta y'} = \epsilon_{xx}|_{i,j} E_x|_{i,j} \quad (4.58)$$

$$-\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i-1,j}}{\Delta x'} = \epsilon_{yy}|_{i,j} E_y|_{i,j} \quad (4.59)$$

$$\frac{\tilde{H}_y|_{i,j} - \tilde{H}_y|_{i-1,j}}{\Delta x'} - \frac{\tilde{H}_x|_{i,j} - \tilde{H}_x|_{i,j-1}}{\Delta y'} = \epsilon_{zz}|_{i,j} E_z|_{i,j} \quad (4.60)$$

As was observed in Chapter 2, both sets of discrete equations have decoupled into two independent sets of three equations. Equations (4.49), (4.50), and (4.54) describe the E mode (TM polarization) when frequency is not known and (4.55), (4.56), and (4.60) describe the E mode when frequency is known. Equations (4.51) to (4.53) describe the H mode (TE polarization) when frequency is not known and (4.57) to (4.59) describe the H mode when frequency is known.

After inspecting these two groups of equations, it can be seen that one of the modes by itself does not utilize all of the field components within a three-dimensional Yee cell. For two-dimensional simulations, the three-dimensional Yee cell has separated into two two-dimensional Yee cells, one for each distinct mode. This is illustrated in Figure 4.3 where the Yee cell for the E mode is extracted from the top half of the three-dimensional Yee cell and the Yee cell for the H mode is extracted from the bottom half.

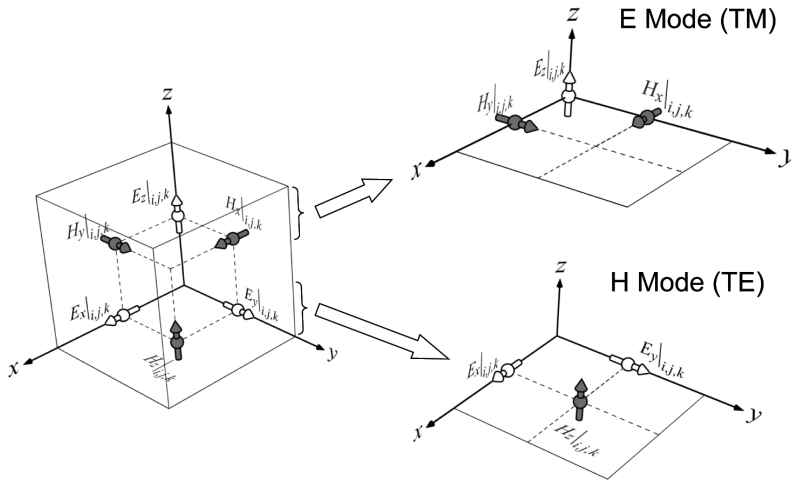


Figure 4.3 Extracting both two-dimensional Yee cells from the three-dimensional Yee cell.

4.4.1 Derivation of E Mode Equations When Frequency Is Not Known

The discrete equations for the E mode (TM polarization) when frequency is not known are taken from (4.49) to (4.54) to be

$$\frac{E_z|_{i,j+1} - E_z|_{i,j}}{\Delta y} = k_0 \mu_{xx}|_{i,j} \tilde{H}_x|_{i,j} \quad (4.61)$$

$$-\frac{E_z|_{i+1,j} - E_z|_{i,j}}{\Delta x} = k_0 \mu_{yy}|_{i,j} \tilde{H}_y|_{i,j} \quad (4.62)$$

$$\frac{\tilde{H}_y|_{i,j} - \tilde{H}_y|_{i-1,j}}{\Delta x} - \frac{\tilde{H}_x|_{i,j} - \tilde{H}_x|_{i,j-1}}{\Delta y} = k_0 \epsilon_{zz}|_{i,j} E_z|_{i,j} \quad (4.63)$$

Each of (4.61) to (4.63) is written once for every point on the grid, and each set of equations can be written as its own matrix equation. These are

$$\mathbf{D}_y^e \mathbf{e}_z = k_0 \mu_{xx} \tilde{\mathbf{h}}_x \quad (4.64)$$

$$-\mathbf{D}_x^e \mathbf{e}_z = k_0 \mu_{yy} \tilde{\mathbf{h}}_y \quad (4.65)$$

$$\mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{D}_y^h \tilde{\mathbf{h}}_x = k_0 \epsilon_{zz} \mathbf{e}_z \quad (4.66)$$

This is called the E mode because only a single electric field term \mathbf{e}_z remains in the equations and the final matrix equation that will be solved will contain only this term.

4.4.2 Derivation of H Mode Equations When Frequency Is Not Known

The discrete equations for the H mode (TE polarization) when the frequency is not known are taken from (4.49) to (4.54) to be

$$\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i,j-1}}{\Delta y} = k_0 \epsilon_{xx}|_{i,j} E_x|_{i,j} \quad (4.67)$$

$$-\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i-1,j}}{\Delta x} = k_0 \epsilon_{yy}|_{i,j} E_y|_{i,j} \quad (4.68)$$

$$\frac{E_y|_{i+1,j} - E_y|_{i,j}}{\Delta x} - \frac{E_x|_{i,j+1} - E_x|_{i,j}}{\Delta y} = k_0 \mu_{zz}|_{i,j} \tilde{H}_z|_{i,j} \quad (4.69)$$

Each of (4.67) to (4.69) is written once for every cell on the grid and each set of equations can be written as its own matrix equation. These are

$$\mathbf{D}_y^h \tilde{\mathbf{h}}_z = k_0 \epsilon_{xx} \mathbf{e}_x \quad (4.70)$$

$$-D_x^h \tilde{\mathbf{h}}_z = k_0 \epsilon_{yy} \mathbf{e}_y \quad (4.71)$$

$$D_x^e \mathbf{e}_y - D_y^e \mathbf{e}_x = k_0 \mu_{zz} \tilde{\mathbf{h}}_z \quad (4.72)$$

This is called the H mode because only a single magnetic field term $\tilde{\mathbf{h}}_z$ remains in the equations and the final matrix equation that will be solved will contain only this term.

4.4.3 Derivation of E Mode Equations When Frequency Is Known

The discrete equations for the E mode (TM polarization) when the frequency is known are taken from (4.55) to (4.60) to be

$$\frac{E_z|_{i,j+1} - E_z|_{i,j}}{\Delta y'} = \mu_{xx}|_{i,j} \tilde{H}_x|_{i,j} \quad (4.73)$$

$$-\frac{E_z|_{i+1,j} - E_z|_{i,j}}{\Delta x'} = \mu_{yy}|_{i,j} \tilde{H}_y|_{i,j} \quad (4.74)$$

$$\frac{\tilde{H}_y|_{i,j} - \tilde{H}_y|_{i-1,j}}{\Delta x'} - \frac{\tilde{H}_x|_{i,j} - \tilde{H}_x|_{i,j-1}}{\Delta y'} = \epsilon_{zz}|_{i,j} E_z|_{i,j} \quad (4.75)$$

Each of the above equations is written once for every cell on the grid and each set of equations can be written as its own matrix equation. These are

$$D_y^e \mathbf{e}_z = \mu_{xx} \tilde{\mathbf{h}}_x \quad (4.76)$$

$$-D_x^e \mathbf{e}_z = \mu_{yy} \tilde{\mathbf{h}}_y \quad (4.77)$$

$$D_x^h \tilde{\mathbf{h}}_y - D_y^h \tilde{\mathbf{h}}_x = \epsilon_{zz} \mathbf{e}_z \quad (4.78)$$

4.4.4 Derivation of H Mode Equations When Frequency Is Known

The discrete equations for the H mode (TE polarization) when the frequency is known taken from (4.55) to (4.60) to be

$$\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i,j-1}}{\Delta y'} = \epsilon_{xx}|_{i,j} E_x|_{i,j} \quad (4.79)$$

$$-\frac{\tilde{H}_z|_{i,j} - \tilde{H}_z|_{i-1,j}}{\Delta x'} = \epsilon_{yy}|_{i,j} E_y|_{i,j} \quad (4.80)$$

$$\frac{E_y|_{i+1,j} - E_y|_{i,j}}{\Delta x'} - \frac{E_x|_{i,j+1} - E_x|_{i,j}}{\Delta y'} = \mu_{zz}|_{i,j} \tilde{H}_z|_{i,j} \quad (4.81)$$

Each of the above equations is written once for every cell on the grid and each set of equations can be written as its own matrix equation. These are

$$D_y^h \tilde{\mathbf{h}}_z = \epsilon_{xx} \mathbf{e}_x \quad (4.82)$$

$$-D_x^h \tilde{\mathbf{h}}_z = \epsilon_{yy} \mathbf{e}_y \quad (4.83)$$

$$D_x^e \mathbf{e}_y - D_y^e \mathbf{e}_x = \mu_{zz} \tilde{\mathbf{h}}_z \quad (4.84)$$

4.5 Derivative Matrices for Two-Dimensional FDFD

It is now necessary to use what was discussed in Chapter 3 to build the derivative matrices for two-dimensional finite-difference analysis on the Yee grid. This will be accomplished by writing all four derivative matrices for a small 3×3 grid and identifying the patterns that will construct the derivative matrices for larger grids fast and simple. The grids for both the E mode and H mode are shown in Figure 4.4.

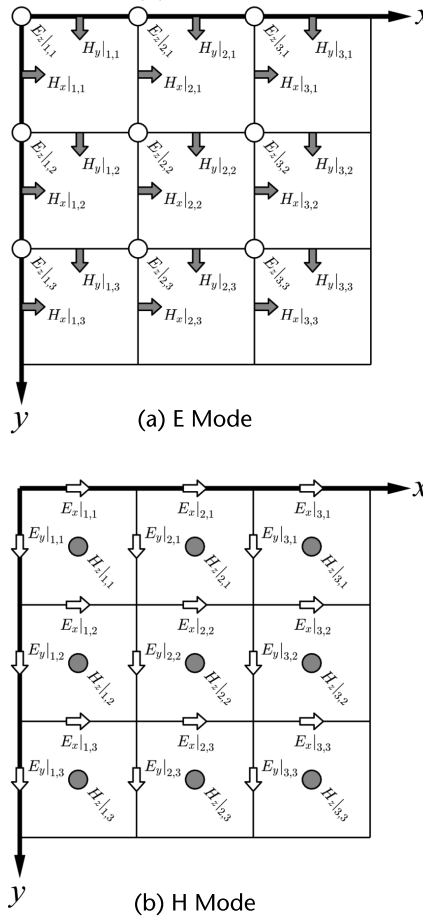


Figure 4.4 3×3 Grid for: (a) E mode and (b) H mode.

Similarly, the derivative matrix \mathbf{D}_y^e is determined by writing the finite-difference approximation in (4.61) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. Using Dirichlet boundary conditions, this large set of equations in matrix form is

$$\frac{1}{\Delta y} \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} E_z|_{1,1} \\ E_z|_{2,1} \\ E_z|_{3,1} \\ E_z|_{1,2} \\ E_z|_{2,2} \\ E_z|_{3,2} \\ E_z|_{1,3} \\ E_z|_{2,3} \\ E_z|_{3,3} \end{bmatrix} = \frac{1}{\Delta y} \begin{bmatrix} E_z|_{1,2} - E_z|_{1,1} \\ E_z|_{2,2} - E_z|_{2,1} \\ E_z|_{3,2} - E_z|_{3,1} \\ E_z|_{1,3} - E_z|_{1,2} \\ E_z|_{2,3} - E_z|_{2,2} \\ E_z|_{3,3} - E_z|_{3,2} \\ 0 - E_z|_{1,3} \\ 0 - E_z|_{2,3} \\ 0 - E_z|_{3,3} \end{bmatrix} \quad (4.87)$$

From (4.87), the derivative matrix \mathbf{D}_y^e is read off as the square matrix premultiplying the column vector of electric field components, including the division by Δy . Observe the form of the matrix for \mathbf{D}_y^e given in (4.88). It is composed of just two diagonals. The center diagonal contains all -1 's and the upper N_x diagonal contains all $+1$'s. For \mathbf{D}_y^e , the upper diagonal is not interrupted by any 0 's so it is an easier diagonal to calculate.

$$\mathbf{D}_y^e = \frac{1}{\Delta y} \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (4.88)$$

The derivative matrix \mathbf{D}_x^h is determined by writing the finite-difference approximation in (4.63) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. The matrix form of these equations is also a large square matrix premultiplying a column vector containing all of the magnetic field components $\tilde{H}_y|_{i,j}$ throughout the grid. The order of the magnetic field components in the column vector is exactly the same as the order of the electric field components used

previously. Using Dirichlet boundary conditions, this large set of equations in matrix form is

$$\frac{1}{\Delta x} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \tilde{H}_y|_{1,1} \\ \tilde{H}_y|_{2,1} \\ \tilde{H}_y|_{3,1} \\ \tilde{H}_y|_{1,2} \\ \tilde{H}_y|_{2,2} \\ \tilde{H}_y|_{3,2} \\ \tilde{H}_y|_{1,3} \\ \tilde{H}_y|_{2,3} \\ \tilde{H}_y|_{3,3} \end{bmatrix} = \frac{1}{\Delta x} \begin{bmatrix} \tilde{H}_y|_{1,1} - 0 \\ \tilde{H}_y|_{2,1} - \tilde{H}_y|_{1,1} \\ \tilde{H}_y|_{3,1} - \tilde{H}_y|_{2,1} \\ \tilde{H}_y|_{1,2} - 0 \\ \tilde{H}_y|_{2,2} - \tilde{H}_y|_{1,2} \\ \tilde{H}_y|_{3,2} - \tilde{H}_y|_{2,2} \\ \tilde{H}_y|_{1,3} - 0 \\ \tilde{H}_y|_{2,3} - \tilde{H}_y|_{1,3} \\ \tilde{H}_y|_{3,3} - \tilde{H}_y|_{2,3} \end{bmatrix} \quad (4.89)$$

From (4.89), the derivative matrix \mathbf{D}_x^h is read off as the square matrix premultiplying the column vector of magnetic field components, including the division by Δx . Observe the form of the matrix for \mathbf{D}_x^h given in (4.90). Like the derivative matrices for the electric fields, it is composed of just two diagonals. However, for the magnetic field the center diagonal contains all +1's and the first lower diagonal contains mostly all -1's but has some 0's inserted where Dirichlet boundary conditions were applied. The zeros occur every N_x rows, where N_x is the number of cells the grid is wide.

$$\mathbf{D}_x^h = \frac{1}{\Delta x} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \quad (4.90)$$

Similarly, the derivative matrix \mathbf{D}_y^h is determined by writing the finite-difference approximation in (4.63) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. Using Dirichlet boundary conditions, this large set of equations in matrix form is

$$\frac{1}{\Delta y} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{H}_x|_{1,1} \\ \tilde{H}_x|_{2,1} \\ \tilde{H}_x|_{3,1} \\ \tilde{H}_x|_{1,2} \\ \tilde{H}_x|_{2,2} \\ \tilde{H}_x|_{3,2} \\ \tilde{H}_x|_{1,3} \\ \tilde{H}_x|_{2,3} \\ \tilde{H}_x|_{3,3} \end{bmatrix} = \frac{1}{\Delta y} \begin{bmatrix} \tilde{H}_x|_{1,1} - 0 \\ \tilde{H}_x|_{2,1} - 0 \\ \tilde{H}_x|_{3,1} - 0 \\ \tilde{H}_x|_{1,2} - \tilde{H}_x|_{1,1} \\ \tilde{H}_x|_{2,2} - \tilde{H}_x|_{2,1} \\ \tilde{H}_x|_{3,2} - \tilde{H}_x|_{3,1} \\ \tilde{H}_x|_{1,3} - \tilde{H}_x|_{1,2} \\ \tilde{H}_x|_{2,3} - \tilde{H}_x|_{2,2} \\ \tilde{H}_x|_{3,3} - \tilde{H}_x|_{3,2} \end{bmatrix} \quad (4.91)$$

From (4.91), the derivative matrix \mathbf{D}_y^h is read off as the square matrix premultiplying the column vector of electric field components, including the division by Δy . Observe the form of the matrix for \mathbf{D}_y^h given in (4.88). It is composed of just two diagonals. The center diagonal contains all +1's and the lower $-N_x$ diagonal contains all -1's. The lower diagonal in \mathbf{D}_y^h is not interrupted by 0's so this diagonal is easier to construct.

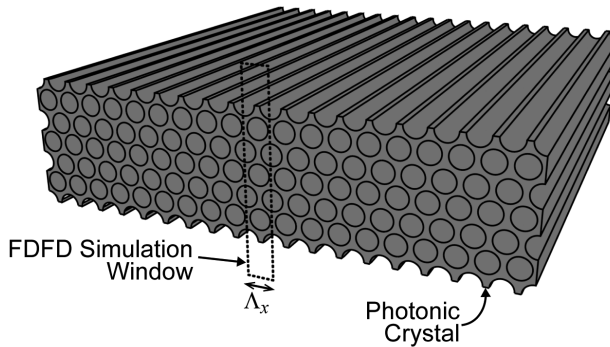
$$\mathbf{D}_y^h = \frac{1}{\Delta y} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \quad (4.92)$$

While only E_z , \tilde{H}_x , and \tilde{H}_y were considered for the above derivative matrices, the same derivative matrices for electric and magnetic fields are constructed no matter what combination of field component and finite-difference equation is considered. It is recommended to the reader to do these derivations as an exercise. Observe that the majority of the elements in the derivative matrices are equal to zero. This becomes even more pronounced when larger grids are considered. Storing all of the elements of a derivative matrix using double-precision floating-point numbers would consume a lot of memory. Instead, it is much more efficient to store the derivative matrices as sparse matrices where only the non-zero elements are stored. MATLAB makes working with sparse matrices very easy. Always store and work with derivative matrices as sparse matrices!

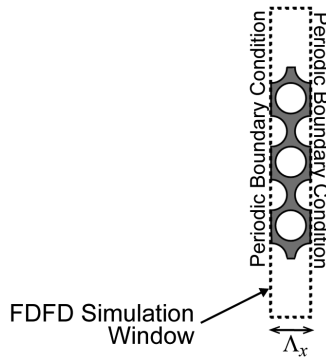
4.5.2 Periodic Boundary Conditions

Very often, periodic structures like photonic crystals and metamaterials can be very efficiently analyzed using periodic boundary conditions (PBCs). Figure 4.5 illustrates the concept where a large and three-dimensional photonic crystal can be represented in a much smaller two-dimensional simulation using PBCs at the left and right boundaries of the grid. In this manner, only a single unit cell of the lattice in the x -direction has to be stored in memory and processed by the simulation. As in Chapter 3, the finite-difference equations written for the cells at the left and right boundaries will require terms from outside of the grid. Since these are not stored in memory, something else must be done.

The concept of a PBC is this. When a field value is needed from outside of the grid, the field value from the opposite side of the grid is used in its place. For example, the finite-difference approximation in (4.62) has the term $E_z|_{i+1,j}$. At the far-right side of the grid where $i = N_x$, this term is $E_z|_{N_x+1,j}$ which is outside of the grid and cannot be used. The term at the opposite side of the grid that can be used in its place is $E_z|_{1,j}$. Summarizing this approach gives the finite-difference approximation in (4.93).



(a) Photonic crystal lattice with FDFD simulation window.



(b) Two-dimensional representation of photonic crystal.

Figure 4.5 (a) Large three-dimensional photonic crystal. (b) Application of periodic boundary conditions to efficiently analyze the three-dimensional photonic crystal with a two-dimensional simulation.

$$\frac{\partial E_z}{\partial x} \approx \begin{cases} \frac{E_z|_{i+1,j} - E_z|_{i,j}}{\Delta x} & i < N_x \\ \frac{E_z|_{1,j} - E_z|_{N_x,j}}{\Delta x} & i = N_x \end{cases} \quad (4.93)$$

A big limitation of the PBC in (4.93) is that it only works when the source wave is at normal incidence. When the source wave is applied at an oblique angle of incidence, the fields at the left and right sides of the grid are out of phase and the PBC in (4.93) fails. This concept is illustrated in Figure 4.6. The fields at the far left and far right sides of the grid are in phase for a wave at normal incidence, but they are out of phase for a wave at an oblique angle. To compensate for the phase difference across the unit cell, the PBC in (4.93) must be modified.

The fundamental states for electromagnetic waves inside of periodic structures are called *Bloch waves* and they obey Bloch's theorem [3]. For the E mode, Bloch's theorem requires the field component E_z to have the following form.

$$E_z(x, y) = A(x, y) \exp[-j(\beta_x x + \beta_y y)] \quad (4.94)$$

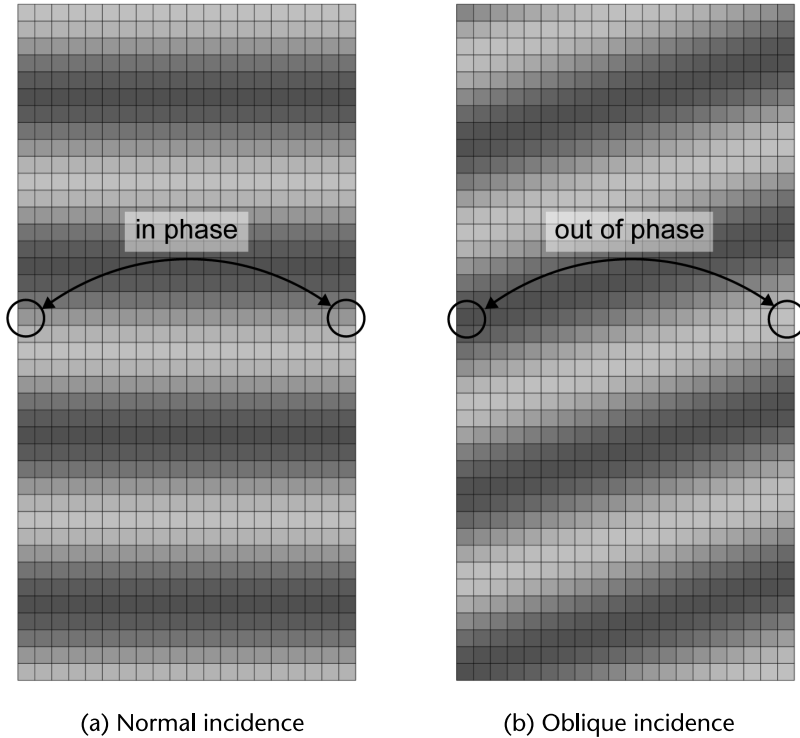


Figure 4.6 Comparison of phase at the far left and far right sides of a grid for a wave at normal incidence and a wave at an oblique angle of incidence.

The function $A(x,y)$ is the complex amplitude function of the Bloch wave, β_x and β_y are components of the Bloch wave vector $\vec{\beta}$, and \vec{r} is the position vector. The amplitude part of the Bloch wave $A(x,y)$ is periodic exactly like the structure the Bloch wave is propagating through. This means the PBC in (4.93) can be applied to $A(x,y)$, but it cannot be applied to the exponential term $\exp[-j(\beta_x x + \beta_y y)]$. From (4.94), the E_z component at the left boundary (x_{lo}) and right boundary (x_{hi}) of the grid can be written as

$$E_z(x_{lo}, y) = A(x_{lo}, y) \exp[-j(\beta_x x_{lo} + \beta_y y)] \quad \text{Left Boundary} \quad (4.95)$$

$$E_z(x_{hi}, y) = A(x_{hi}, y) \exp[-j(\beta_x x_{hi} + \beta_y y)] \quad \text{Right Boundary} \quad (4.96)$$

Since the amplitude function $A(x,y)$ of the Bloch wave is periodic, boundary conditions at the left and right boundaries of the grid require that $A(x_{hi}, y) = A(x_{lo}, y)$. This allows (4.96) to be written as

$$E_z(x_{hi}, y) = A(x_{lo}, y) \exp[-j(\beta_x x_{hi} + \beta_y y)] \quad (4.97)$$

Solving (4.95) for $A(x_{lo}, y)$ gives

$$A(x_{lo}, y) = E_z(x_{lo}, y) \exp[+j(\beta_x x_{lo} + \beta_y y)] \quad (4.98)$$

Substituting this expression for $A(x_{lo}, y)$ into (4.97) leads to an equation that relates the electric fields at the left and right boundaries of the grid.

$$E_z(x_{hi}, y) = E_z(x_{lo}, y) \exp[-j\beta_x(x_{hi} - x_{lo})] \quad (4.99)$$

The exponential term in (4.99) describes the phase that accumulates across the unit cell in the x -direction. Boundary conditions dictate that the tangential component of the wave vector be continuous. From Figure 4.5, the boundary conditions imply that $\beta_x = k_{x,inc}$. The expression $x_{hi} - x_{lo}$ is the physical width of the grid which is the period of the device Λ_x . It is convenient to define a term Φ_x that incorporates the transverse phase across one unit cell.

$$\Phi_x = \exp[-j\beta_x(x_{hi} - x_{lo})] = \exp(-jk_{x,inc}\Lambda_x) \quad (4.100)$$

Given all of this, the PBC for electric fields that accounts for the angle of incidence of the source wave is

$$\frac{\partial E_m}{\partial x} \approx \begin{cases} \frac{E_m|_{i+1,j} - E_m|_{i,j}}{\Delta x} & i < N_x \\ \frac{\Phi_x E_m|_{1,j} - E_m|_{N_x,j}}{\Delta x} & i = N_x \end{cases} \quad m = x, y, z \quad (4.101)$$

Similarly, the PBC for magnetic fields that accounts for the angle of incidence of the source wave is

$$\frac{\partial \tilde{H}_m}{\partial x} \approx \begin{cases} \frac{\tilde{H}_m|_{1,j} - \Phi_x^* \tilde{H}_m|_{N_x,j}}{\Delta x} & i = 1 \\ \frac{\tilde{H}_m|_{i,j} - \tilde{H}_m|_{i-1,j}}{\Delta x} & i > 1 \end{cases} \quad m = x, y, z \quad (4.102)$$

The * superscript in (4.102) indicates that a complex conjugate is applied to reverse the sign of the phase in the term Φ_x . For derivatives of electric fields, the boundary condition problem arose at the right side of the grid and terms from the left side were used. For derivatives of magnetic fields, the boundary condition problem arises at the left side of the grid and terms from the right side are used. This reverses the sign of the phase to be used in the PBC for magnetic fields.

In some cases, it may be desired to use PBCs at the top (y_{lo}) and bottom (y_{hi}) boundaries of the grid in addition to the left (x_{lo}) and right (x_{hi}) boundaries. This is how photonic bands will be calculated in Chapter 7. In these cases, the finite-difference approximations with PBCs are written as

$$\frac{\partial E_m}{\partial y} \approx \begin{cases} \frac{E_m|_{i,j+1} - E_m|_{i,j}}{\Delta y} & j < N_y \\ \frac{\Phi_y E_m|_{i,1} - E_m|_{i,N_y}}{\Delta y} & j = N_y \end{cases} \quad m = x, y, z \quad (4.103)$$

$$\frac{\partial \tilde{H}_m}{\partial y} \approx \begin{cases} \frac{\tilde{H}_m|_{i,1} - \Phi_y^* \tilde{H}_m|_{i,N_y}}{\Delta y} & j = 1 \\ \frac{\tilde{H}_m|_{i,j} - \tilde{H}_m|_{i,j-1}}{\Delta y} & j > 1 \end{cases} \quad m = x, y, z \quad (4.104)$$

$$\Phi_y = \exp(-jk_{y,\text{inc}} \Lambda_y) \quad (4.105)$$

4.5.3 Derivative Matrices Incorporating Periodic Boundary Conditions

The derivative matrices for electric and magnetic fields with the revised PBCs can be constructed following the same procedure used in Section 4.5.1. The derivative matrix \mathbf{D}_x^e with PBCs is determined by writing the finite-difference approximation in (4.62) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. The matrix form of these equations is a large square matrix premultiplying

a column vector containing all of the electric field components $E_z|_{i,j}$ throughout the grid. Using PBCs, this large set of equations in matrix form is

$$\frac{1}{\Delta x} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \Phi_x & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi_x & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \Phi_x & 0 & -1 \end{bmatrix} \begin{bmatrix} E_z|_{1,1} \\ E_z|_{2,1} \\ E_z|_{3,1} \\ E_z|_{1,2} \\ E_z|_{2,2} \\ E_z|_{3,2} \\ E_z|_{1,3} \\ E_z|_{2,3} \\ E_z|_{3,3} \end{bmatrix} = \frac{1}{\Delta x} \begin{bmatrix} E_z|_{2,1} - E_z|_{1,1} \\ E_z|_{3,1} - E_z|_{2,1} \\ \Phi_x E_z|_{1,1} - E_z|_{3,1} \\ E_z|_{2,2} - E_z|_{1,2} \\ E_z|_{3,2} - E_z|_{2,2} \\ \Phi_x E_z|_{1,2} - E_z|_{3,2} \\ E_z|_{2,3} - E_z|_{1,3} \\ E_z|_{3,3} - E_z|_{2,3} \\ \Phi_x E_z|_{1,3} - E_z|_{3,3} \end{bmatrix} \quad (4.106)$$

From (4.106), the derivative matrix \mathbf{D}_x^e is read off as the square matrix premultiplying the column vector of electric field components, including the division by Δx . Observe the form of the matrix for \mathbf{D}_x^e given in (4.107). It is essentially the derivative matrix with Dirichlet boundary conditions but with phase terms Φ_x added in the rows where PBCs are used. This suggests an easy way to build a derivative matrix. First, build the derivative matrix with Dirichlet boundary conditions. Second, incorporate the phase terms only if PBCs are required.

$$\mathbf{D}_x^e = \frac{1}{\Delta x} \begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \Phi_x & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & \Phi_x & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & \Phi_x & 0 & -1 \end{bmatrix} \quad (4.107)$$

The derivative matrix \mathbf{D}_y^e with PBCs is determined by writing the finite-difference approximation in (4.61) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. The results of this are

$$\frac{1}{\Delta y} \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} E_z|_{1,1} \\ E_z|_{2,1} \\ E_z|_{3,1} \\ E_z|_{1,2} \\ E_z|_{2,2} \\ E_z|_{3,2} \\ E_z|_{1,3} \\ E_z|_{2,3} \\ E_z|_{3,3} \end{bmatrix} = \frac{1}{\Delta y} \begin{bmatrix} E_z|_{1,2} - E_z|_{1,1} \\ E_z|_{2,2} - E_z|_{2,1} \\ E_z|_{3,2} - E_z|_{3,1} \\ E_z|_{1,3} - E_z|_{1,2} \\ E_z|_{2,3} - E_z|_{2,2} \\ E_z|_{3,3} - E_z|_{3,2} \\ \Phi_y E_z|_{1,1} - E_z|_{1,3} \\ \Phi_y E_z|_{2,1} - E_z|_{2,3} \\ \Phi_y E_z|_{3,1} - E_z|_{3,3} \end{bmatrix} \quad (4.108)$$

$$\mathbf{D}_y^e = \frac{1}{\Delta y} \begin{bmatrix} -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \\ \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & \Phi_y & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix} \quad (4.109)$$

The derivative matrix \mathbf{D}_x^h with PBCs is determined by writing the finite-difference approximation in (4.68) once for each point on the grid depicted in Figure 4.4,

writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. The results of this are

$$\frac{1}{\Delta x} \begin{bmatrix} 1 & 0 & -\Phi_x^* & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\Phi_x^* & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\Phi_x^* \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} \tilde{H}_y|_{1,1} \\ \tilde{H}_y|_{2,1} \\ \tilde{H}_y|_{3,1} \\ \tilde{H}_y|_{1,2} \\ \tilde{H}_y|_{2,2} \\ \tilde{H}_y|_{3,2} \\ \tilde{H}_y|_{1,3} \\ \tilde{H}_y|_{2,3} \\ \tilde{H}_y|_{3,3} \end{bmatrix} = \frac{1}{\Delta x} \begin{bmatrix} \tilde{H}_y|_{1,1} - \Phi_x^* \tilde{H}_y|_{3,1} \\ \tilde{H}_y|_{2,1} - \tilde{H}_y|_{1,1} \\ \tilde{H}_y|_{3,1} - \tilde{H}_y|_{2,1} \\ \tilde{H}_y|_{1,2} - \Phi_x^* \tilde{H}_y|_{3,2} \\ \tilde{H}_y|_{2,2} - \tilde{H}_y|_{1,2} \\ \tilde{H}_y|_{3,2} - \tilde{H}_y|_{2,2} \\ \tilde{H}_y|_{1,3} - \Phi_x^* \tilde{H}_y|_{3,3} \\ \tilde{H}_y|_{2,3} - \tilde{H}_y|_{1,3} \\ \tilde{H}_y|_{3,3} - \tilde{H}_y|_{2,3} \end{bmatrix} \quad (4.110)$$

$$\mathbf{D}_x^h = \frac{1}{\Delta x} \begin{bmatrix} 1 & 0 & -\Phi_x^* & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -\Phi_x^* & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & -\Phi_x^* \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix} \quad (4.111)$$

Last, the derivative matrix \mathbf{D}_y^h with PBCs is determined by writing the finite-difference approximation in (4.67) once for each point on the grid depicted in Figure 4.4, writing the large set of equations as a single matrix equation, and then reading off the derivative matrix. The results of this are

$$\frac{1}{\Delta y} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{H}_x|_{1,1} \\ \tilde{H}_x|_{2,1} \\ \tilde{H}_x|_{3,1} \\ \tilde{H}_x|_{1,2} \\ \tilde{H}_x|_{2,2} \\ \tilde{H}_x|_{3,2} \\ \tilde{H}_x|_{1,3} \\ \tilde{H}_x|_{2,3} \\ \tilde{H}_x|_{3,3} \end{bmatrix} = \frac{1}{\Delta y} \begin{bmatrix} \tilde{H}_x|_{1,1} - \Phi_y^* \tilde{H}_x|_{1,3} \\ \tilde{H}_x|_{2,1} - \Phi_y^* \tilde{H}_x|_{2,3} \\ \tilde{H}_x|_{3,1} - \Phi_y^* \tilde{H}_x|_{3,3} \\ \tilde{H}_x|_{1,2} - \tilde{H}_x|_{1,1} \\ \tilde{H}_x|_{2,2} - \tilde{H}_x|_{2,1} \\ \tilde{H}_x|_{3,2} - \tilde{H}_x|_{3,1} \\ \tilde{H}_x|_{1,3} - \tilde{H}_x|_{1,2} \\ \tilde{H}_x|_{2,3} - \tilde{H}_x|_{2,2} \\ \tilde{H}_x|_{3,3} - \tilde{H}_x|_{3,2} \end{bmatrix} \quad (4.112)$$

$$\mathbf{D}_y^h = \frac{1}{\Delta y} \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -\Phi_y^* \\ -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \quad (4.113)$$

4.5.4 Relationship Between the Derivative Matrices

For both Dirichlet boundary conditions and PBCs, the derivative matrices are related through

$$\mathbf{D}_x^h = -(\mathbf{D}_x^e)^H \quad \mathbf{D}_y^h = -(\mathbf{D}_y^e)^H \quad (4.114)$$

The superscript H indicates a Hermitian transpose. This is an ordinary transpose followed by calculating the complex conjugate of all the elements in the matrix. This is a very convenient relation because only the electric field derivative matrices have to be constructed and the magnetic field derivative matrices can be calculated directly from the electric field derivative matrices using (4.114). While it is possible to formulate and implement FDFD using only the electric field derivative matrices,

it is not recommended because this relation does not hold for all boundary conditions. Be cautious when applying the relation in (4.114)!

4.6 Derivative Matrices for Three-Dimensional FDFD

It is straightforward to generalize the discussion above to derivative matrices needed for three-dimensional FDFD analysis. For a $3 \times 3 \times 3$ grid, the derivative matrix D_x^c with Dirichlet boundary conditions is

$$D_x^c = \frac{1}{\Delta x} \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & -1 \end{bmatrix} \quad (4.115)$$

Observe the form of the matrix for D_x^c given in (4.115). Three-dimensional derivative matrices are still composed of just two diagonals. The center diagonal contains all -1 's and the first upper diagonal contains mostly all $+1$'s but has some 0 's inserted where Dirichlet boundary conditions were applied. The zeros occur every N_x rows, where N_x is the number of cells the grid is large in the x -direction.

The derivative matrix D_y^c with Dirichlet boundary conditions is

(4.116)

The derivative matrix \mathbf{D}_z^e with Dirichlet boundary conditions is

(4.117)

Observe the form of the matrix for D_z^c given in (4.117). The center diagonal contains all -1 's. The upper diagonal contains all $+1$'s and is placed along the $N_x N_y$ diagonal.

[illegible]

(4.118)

[illegible]

(4.119)

(4.120)

4.6.1 Relationship Between the Derivative Matrices

$$\mathbf{D}_x^h = -(\mathbf{D}_x^e)^H \quad \mathbf{D}_y^h = -(\mathbf{D}_y^e)^H \quad \mathbf{D}_z^h = -(\mathbf{D}_z^e)^H \quad (4.121)$$

This is the same relationship observed for two-dimensional grids with the addition of the relation between D_z^h and D_z^e . As mentioned previously, be cautious when applying this relationship when other boundary conditions are being used.

4.7 Programming the `yeeder2d()` Function in MATLAB

Almost all of the tediousness of the finite-difference method is wrapped up in the construction of the derivative matrices. The derivative matrices also give an intermediate step to test and verify for correctness. For these reasons, a function will be written in MATLAB to construct the derivative matrices, allowing the rest of the code for FDFD to be much cleaner and simpler. The MATLAB code for this very useful function can be downloaded at <https://empossible.net/fdfdbook/>. Line 1 declares the file as the MATLAB function `yeeder2d()` and defines four input arguments (`NS`, `RES`, `BC`, and `kinc`) and four output arguments (`DEX`, `DEY`, `DHX`, and `DHY`). When Dirichlet boundary conditions are used at all boundaries, only three input arguments are needed because it is not necessary to provide the incident wave vector `kinc`. When the grid is normalized, it is still necessary to multiply the resolution parameters `dx` and `dy` by `k0` even when `kinc` is not provided to the `yeeder2d()` function. The remainder of the header is commented so that when `help yeeder2d` is typed at the command prompt, the commented text in the header will be displayed in the command window. It is recommended to include the name of the file, an expanded title, the syntax for a typical call to the function, and a detailed list and description of all the input and output arguments of the function.

Lines 25 to 42 are a section of code that handles the input arguments. The first thing that is done is to extract the size of the grid `Nx` and `Ny` from the input array `NS` and to extract the resolution parameters `dx` and `dy` from the input array `RES`. It is certainly possible to use `NS(1)` in place of `Nx` and `NS(2)` in place of `Ny`, but this makes the code less readable. The same can be said for using `RES(1)` and `RES(2)` in place of `dx` and `dy`. The second thing that is done in this section is to define a default incident wave vector `kinc` if one is not specified as an input argument. While this step is not necessary either, it makes the task of initializing the derivative matrices in later sections a little easier. The third thing in this section of code is to calculate the size of the derivative matrices. The derivative matrices are square and contain one row and one column for every cell on the grid. If the grid is `Nx` cells wide by `Ny` cells tall, then the total number of cells on the grid is $M = Nx \times Ny$. The last thing accomplished in this section of code is to initialize a sparse matrix `Z` of all zeros. It is critical to never build or store a full matrix. All matrices will always be sparse and the zero matrix `Z` allows for easy and simple code to build the derivative matrices.

Lines 44 to 72 of the `yeeder2d()` function build the derivative matrix \mathbf{D}_x^e , which in MATLAB will be called `DEX`. It is good practice to write the `yeeder2d()` function in a way that will allow it to be used to build derivative matrices for one-dimensional grids in addition to two-dimensional grids. When the numerical size of the grid in the x -direction is $Nx = 1$, the derivative matrix should be set to all zeros unless PBCs are used. When PBCs are used, the derivative matrix should be a diagonal matrix with $-jk_{x,\text{inc}}$ placed down the entire center diagonal. By setting the input argument `kinc` to all zeros when this parameter is not provided, it allows the derivative matrix to be constructed for any case by placing $-jk_{x,\text{inc}}$ along the center diagonal. When the numerical size of the grid is greater than 1 in the x -direction, the `yeeder2d()` function moves on to build two one-dimensional arrays that will

be the numbers inserted into the two main diagonals of the derivative matrix `DEX`. The variable `d0` will be inserted into the center diagonal so it is initialized as a one-dimensional array of all -1 's. The variable `d1` will be inserted into the first upper diagonal. It is a one-dimensional array initialized with all $+1$'s, but with 0 's inserted into the positions where Dirichlet boundary conditions are needed every N_x number of cells. Given the one-dimensional arrays `d0` and `d1`, they are inserted into the derivative matrix at the same time using MATLAB's function `spdiags()`. The first input argument into the `spdiags()` function is a matrix, and its columns are the one-dimensional arrays to be inserted into the diagonals. It is convenient to divide these arrays by the grid resolution parameter `dx` at this step. The second input argument is an array containing the diagonal numbers that the previous one-dimensional arrays should be inserted into. In this case, 0 indicates the center diagonal and $+1$ indicates the first upper diagonal. The third input argument to `spdiags()` is the matrix into which the diagonals should be inserted. The zero matrix is given here so only the diagonals inserted by this single call to `spdiags()` will appear in the `DEX` matrix. At this point, the derivative matrix is correct for Dirichlet boundary conditions. The last part of this section of code is to insert a third diagonal containing the phase terms where PBCs are to be used instead of Dirichlet boundary conditions. The variable `d1` is overwritten here with the new diagonal containing the phase terms that are inserted into the `DEX` matrix using `spdiags()`.

Lines 74 to 100 of the `yeeDer2d()` function are very similar to the previous section of code. It builds the derivative matrix D_y^e , which in MATLAB will be called `DEY`. When the numerical size of the grid in the y -direction is $N_y = 1$, the derivative matrix is initialized by placing $-jk_{y,\text{inc}}$ along the entire center diagonal. When the numerical size of the grid is greater than 1, the `yeeDer2d()` function moves on to building two one-dimensional arrays that will be inserted into the two main diagonals of the derivative matrix. The variable `d0` will be inserted into the center diagonal so it is initialized as a one-dimensional array of all -1 's. The variable `d1` will be inserted into the N_x th upper diagonal. It is a one-dimensional array initialized with all $+1$'s. Given the one-dimensional arrays `d0` and `d1`, they are inserted into the derivative matrix at the same time using the built-in function `spdiags()`. At this point, the derivative matrix is correct for Dirichlet boundary conditions. The last part of this section of code inserts a third diagonal containing the phase terms where PBCs are to be used instead of Dirichlet boundary conditions. The variable `d1` is overwritten with the new diagonal containing the phase terms and then it is inserted into the `DEY` matrix using `spdiags()`.

Lines 102 to 107 of the `yeeDer2d()` function build the derivative matrices for the magnetic fields D_x^h and D_y^h . In MATLAB, these will be called `DHX` and `DHY`, respectively. This is a very easy step because the boundary conditions that this function incorporates allows D_x^h and D_y^h to be calculated directly from D_x^e and D_y^e using (4.114).

Adding and deleting elements from sparse matrices can be slow, especially for large matrices. To optimize the speed and efficiency of `yeeDer2d()`, the diagonals were calculated ahead of time and inserted into the derivative matrices with a minimum number of calls to `spdiags()`. It is possible to write `yeeDer2d()` a bit

differently that utilizes just a single call to `spdiags()`, but that approach was not taken here in order to make the MATLAB code easier to read and understand.

4.7.1 Using the `yeeDer2d()` Function

The `yeeDer2d()` function by itself is not executable. Instead, it is a function that must be called from a MATLAB program that defines the input arguments and passes them to the function so that `yeeDer2d()` can build the derivative matrices and return them to the program. The MATLAB program listed below demonstrates how to use the `yeeDer2d()` function and displays the derivative matrices that `yeeDer2d()` builds. The input arguments are defined on lines 8 to 11. NS is an array containing the size of the grid. In this case, the grid is three cells wide and four cells tall (i.e., $N_x = 3$ and $N_y = 4$). RES is an array containing the resolution parameters. For this case, $dx=0.2$ and $dy=0.1$. The boundary conditions in the array BC where set to Dirichlet for both x - and y -axis boundaries. The `yeeDer2d()` function is called on line 14 where the input arguments are given to the function, the derivative matrices are built, and the derivative matrices returned to the program below through the variables DEX , DEY , DHX , and DHY .

```

1  % Chapter4_yeeDer2d_demo.m
2
3  % INITIALIZE MATLAB
4  close all;
5  clc;
6  clear all;
7
8  % DEFINE INPUT ARGUMENTS FOR YEEDE2D
9  NS = [3 4];
10 RES = [0.2 0.1];
11 BC = [0 0];
12
13 % CALL YEEDE2D
14 [DEX,DEY,DHX,DHY] = yeeDer2d(NS,RES,BC);
15
16 % SHOW DERIVATIVE MATRICES
17 disp('DEX = ');
18 disp(full(DEX));
19
20 disp('DEY = ');
21 disp(full(DEY));
22
23 disp('DHX = ');
24 disp(full(DHX));
25
26 disp('DHY = ');
27 disp(full(DHY));

```

The derivative matrices are displayed to the command window from lines 16 to 27. They are sparse matrices so they are converted to full matrices before being displayed. Be careful to only attempt to display the derivative matrices for small grids this way. The output of the function is

```

DEX =
    -5     5     0     0     0     0     0     0     0     0     0     0
     0    -5     5     0     0     0     0     0     0     0     0     0
     0     0    -5     0     0     0     0     0     0     0     0     0
     0     0     0    -5     5     0     0     0     0     0     0     0
     0     0     0     0    -5     5     0     0     0     0     0     0
     0     0     0     0     0    -5     0     0     0     0     0     0
     0     0     0     0     0     0    -5     5     0     0     0     0
     0     0     0     0     0     0     0    -5     5     0     0     0
     0     0     0     0     0     0     0     0    -5     0     0     0
     0     0     0     0     0     0     0     0     0    -5     5     0
     0     0     0     0     0     0     0     0     0     0    -5     5
     0     0     0     0     0     0     0     0     0     0     0    -5

DEY =
   -10     0     0    10     0     0     0     0     0     0     0     0
     0    -10     0     0    10     0     0     0     0     0     0     0
     0     0   -10     0     0    10     0     0     0     0     0     0
     0     0     0   -10     0     0    10     0     0     0     0     0
     0     0     0     0   -10     0     0    10     0     0     0     0
     0     0     0     0     0   -10     0     0    10     0     0     0
     0     0     0     0     0     0   -10     0     0    10     0     0
     0     0     0     0     0     0     0   -10     0     0    10     0
     0     0     0     0     0     0     0     0   -10     0     0    10
     0     0     0     0     0     0     0     0     0   -10     0     0
     0     0     0     0     0     0     0     0     0     0   -10     0
     0     0     0     0     0     0     0     0     0     0     0   -10

DHX =
     5     0     0     0     0     0     0     0     0     0     0     0
    -5     5     0     0     0     0     0     0     0     0     0     0
     0    -5     5     0     0     0     0     0     0     0     0     0
     0     0     0     5     0     0     0     0     0     0     0     0
     0     0     0     0    -5     5     0     0     0     0     0     0
     0     0     0     0     0    -5     5     0     0     0     0     0
     0     0     0     0     0     0     5     0     0     0     0     0
     0     0     0     0     0     0     0    -5     5     0     0     0
     0     0     0     0     0     0     0     0   -5     5     0     0
     0     0     0     0     0     0     0     0     0     5     0     0
     0     0     0     0     0     0     0     0     0     0    -5     5
     0     0     0     0     0     0     0     0     0     0    -5     5

```

```

DHY =
10    0    0    0    0    0    0    0    0    0    0    0
 0   10    0    0    0    0    0    0    0    0    0    0
 0    0   10    0    0    0    0    0    0    0    0    0
-10   0    0   10    0    0    0    0    0    0    0    0
 0  -10   0    0   10    0    0    0    0    0    0    0
 0    0  -10   0    0   10    0    0    0    0    0    0
 0    0    0  -10   0    0   10    0    0    0    0    0
 0    0    0    0  -10   0    0   10    0    0    0    0
 0    0    0    0    0  -10   0    0   10    0    0    0
 0    0    0    0    0    0  -10   0    0   10    0    0
 0    0    0    0    0    0    0  -10   0    0   10    0
 0    0    0    0    0    0    0    0  -10   0    0   10

```

4.8 Programming the yeeder3d() Function in MATLAB

The MATLAB code for the `yeeder3d()` function can be downloaded at <https://empossible.net/fdfdbook/>. This function calculates the derivative matrices `DEX`, `DEY`, `DEZ`, `DHX`, `DHY`, and `DHZ` across a three-dimensional Yee grid. This function is only necessary for the FDFD codes presented in Chapter 10. Lines 2 to 25 are the header of the function and are what is displayed in the command window if `help yeeder3d` is entered at the command prompt. It is always a good practice to include a header in functions in order to remember how to use them.

Lines 27 to 45 handle the input arguments by extracting meaningful information and initializing some variables. The size of the grid and grid resolution parameters are extracted from the input arguments on lines 32 to 34. Lines 36 to 39 calculate a default `kinc` vector if one is not provided. The default vector is set to all zeros. The size of the derivative matrices is determined on line 42 by calculating the total number of points on the three-dimensional Yee grid. Line 45 initializes a sparse matrix `Z` of all zeros.

Lines 47 to 75 build the `DEX` derivative matrix and resembles the code given in `yeeder2d()` very closely. If the size of the grid in the x -direction is 1, lines 52 and 53 initialize the derivative matrix to a diagonal matrix with $-jk_{x,\text{inc}}$ along the entire center diagonal. If the size of the grid is larger than 1, lines 56 to 75 build the matrix. Line 59 calculates an array `d0` containing the numbers to be inserted into the center diagonal of `DEX`. Lines 62 and 63 calculate an array `d1` containing the numbers to be inserted into the +1 diagonal in `DEX`. Line 63 sets every `Nx` number of cells to zero to incorporate Dirichet boundary conditions. Line 66 divides the two arrays `d0` and `d1` by `dx` and inserts them into `DEX`. If periodic boundary conditions are called for, lines 69 to 73 incorporate these into `DEX`.

Lines 77 to 110 build the `DEY` derivative matrix. If the size of the grid in the y -direction is 1, lines 82 and 83 initialize the derivative matrix to a diagonal matrix with $-jk_{y,\text{inc}}$ along the entire center diagonal. If the size of the grid is larger than 1, lines 86 to 110 build the matrix. Line 89 calculates an array `d0` containing the numbers to be inserted into the center diagonal of `DEY`. Lines 91 and 94 calculate an array `d1` containing the numbers to be inserted into the `Nx` diagonal in `DEY`. Line 92 creates an array containing $(N_y - 1) \times N_x$ ones followed by `Nx` zeros. Line 93 repeats this array `Nz - 1` number of times to form a larger array. Line 94 adds `Nx` zeros to the

beginning of this larger array and $(N_y - 1) \times N_x$ ones to the end. While all of this may seem confusing, it is just done to create the correct pattern of zeros and ones that must be placed along the $N_x \times (N_y - 1)$ diagonal of DEY to construct the derivative matrix. Line 97 divides the two arrays d0 and d1 by dy and inserts them into DEY. If periodic boundary conditions are called for, lines 100 to 108 incorporate these into DEY.

Lines 112 to 135 build the DEZ derivative matrix. If the size of the grid in the z -direction is 1, lines 117 and 118 initialize the derivative matrix to a diagonal matrix with $-jk_{z,\text{inc}}$ along the entire center diagonal. If the size the grid is larger than one, lines 120 to 135 build the matrix. Line 124 calculates an array d0 of all ones. There is no need for a second array d1 to be calculated because the second diagonal is not broken up in any way to enforce boundary conditions. Line 127 divides this array by dz and inserts it twice into DEZ, but one diagonal is inserted with opposite sign. If periodic boundary conditions are called for, lines 130 to 133 incorporate these into DEZ. Last, lines 137 to 143 calculate the magnetic field derivative matrices DHX, DHY, and DHZ directly from the electric field derivative matrices. This is possible because only Dirichlet and periodic boundary conditions are used.

4.8.1 Using the yeeder3d() Function

The `yeeder3d()` function by itself is not executable. Instead, it is a function that must be called from a MATLAB program that defines the input arguments and passes them to the function so that `yeeder3d()` can build the derivative matrices and return them to the program. The MATLAB program listed below demonstrates how to use the `yeeder3d()` function and displays the derivative matrices that `yeeder3d()` builds. The input arguments are defined on lines 8 to 11. NS is an array containing the size of the grid. In this case, the grid has $N_x = N_y = N_z = 2$. RES is an array containing the resolution parameters. For this case, $dx=0.3$, $dy=0.2$, and $dz=0.1$. The boundary conditions in the array BC were set to Dirichlet for all boundaries. The `yeeder3d()` function is called on line 14 where the input arguments are given to the function, the derivative matrices are built, and the derivative matrices returned to the program below through the variables DEX, DEY, DEZ DHX, DHY, and DHZ.

```

1  % Chapter4_yeeder3d_demo.m
2
3  % INITIALIZE MATLAB
4  close all;
5  clc;
6  clear all;
7
8  % DEFINE INPUT ARGUMENTS FOR YEEDER2D
9  NS = [2 2 2];
10 RES = [0.3 0.2 0.1];
11 BC = [0 0 0];
12
13 % CALL YEEDER3D
14 [DEX,DEY,DEZ,DHX,DHY,DHZ] = yeeder3d(NS,RES,BC);
15
16 % SHOW DERIVATIVE MATRICES
17 disp('DEX = ');
```

```

18 disp(full(DEX));
19
20 disp('DEY = ');
21 disp(full(DEY));
22
23 disp('DEZ = ');
24 disp(full(DEZ));
25
26 disp('DHX = ');
27 disp(full(DHX));
28
29 disp('DHY = ');
30 disp(full(DHY));
31
32 disp('DHZ = ');
33 disp(full(DHZ));

```

The derivative matrices are displayed to the command window from lines 16 to 33. They are sparse matrices so they are converted to full matrices before being displayed. Be careful to only attempt to display the derivative matrices for small grids this way. The output of the function is

```

DEX =
-3.3333    3.3333         0         0         0         0         0         0
         0   -3.3333         0         0         0         0         0         0
         0         0   -3.3333    3.3333         0         0         0         0
         0         0         0   -3.3333         0         0         0         0
         0         0         0         0   -3.3333    3.3333         0         0
         0         0         0         0         0   -3.3333         0         0
         0         0         0         0         0         0   -3.3333    3.3333
         0         0         0         0         0         0         0   -3.3333

```

```

DEY =
-5         0         5         0         0         0         0         0
         0        -5         0         5         0         0         0         0
         0         0        -5         0         0         0         0         0
         0         0         0        -5         0         0         0         0
         0         0         0         0        -5         0         5         0
         0         0         0         0         0        -5         0         5
         0         0         0         0         0         0        -5         0
         0         0         0         0         0         0         0        -5

```

```

DEZ =
-10         0         0         0        10         0         0         0
         0        -10         0         0         0        10         0         0
         0         0        -10         0         0         0        10         0
         0         0         0        -10         0         0         0        10
         0         0         0         0        -10         0         0         0
         0         0         0         0         0        -10         0         0
         0         0         0         0         0         0        -10         0
         0         0         0         0         0         0         0        -10

```

$$\begin{array}{cccccccc}
 \text{DHX} = & & & & & & & \\
 & 3.3333 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & -3.3333 & 3.3333 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 3.3333 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & -3.3333 & 3.3333 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 3.3333 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & -3.3333 & 3.3333 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & 3.3333 \\
 & 0 & 0 & 0 & 0 & 0 & 0 & -3.3333 & 3.3333
 \end{array}$$

$$\begin{array}{cccccccc}
 \text{DHY} = & & & & & & & \\
 & 5 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 5 & 0 & 0 & 0 & 0 & 0 \\
 & -5 & 0 & 5 & 0 & 0 & 0 & 0 \\
 & 0 & -5 & 0 & 5 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 5 & 0 & 0 \\
 & 0 & 0 & 0 & 0 & 0 & 5 & 0 \\
 & 0 & 0 & 0 & 0 & -5 & 0 & 5 \\
 & 0 & 0 & 0 & 0 & 0 & -5 & 0 & 5
 \end{array}$$

$$\begin{array}{cccccccc}
 \text{DHZ} = & & & & & & & \\
 & 10 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 10 & 0 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 10 & 0 & 0 & 0 & 0 \\
 & 0 & 0 & 0 & 10 & 0 & 0 & 0 \\
 & -10 & 0 & 0 & 0 & 10 & 0 & 0 \\
 & 0 & -10 & 0 & 0 & 0 & 10 & 0 \\
 & 0 & 0 & -10 & 0 & 0 & 0 & 10 \\
 & 0 & 0 & 0 & -10 & 0 & 0 & 0 & 10
 \end{array}$$

4.9 The 2× Grid Technique

In FDFD, the field components are positioned throughout space following the Yee grid scheme. While the Yee grid offers many numerical advantages, it makes it more difficult to build devices onto the grid, especially when material boundaries slice through the middle of cells on the grid. Sorting out the assignment of material values to the different permittivity and permeability arrays across the Yee grid can be tedious and difficult. The 2× grid technique was developed to make this process simple and fast and is particularly well suited for dielectric devices with curved features [4].

From Figures 4.4 and 4.7, the field components are staggered across the Yee grid in a way that suggests that somehow a simulation is getting twice the resolution as the standard Yee grid. While this is not entirely true, it is the inspiration for the 2× grid technique. Figure 4.7 illustrates the same grids as Figure 4.4 but the grids are overlaid with a second grid drawn with dashed lines. The second grid has the same physical size as the original Yee grid, but has twice as many cells at half the size as the original Yee grid. This second grid that has twice as many cells is called the 2× grid. The field components from the Yee grid overlay perfectly onto unique cells in the 2× grid.

Calculating the tensor arrays ER_{xx} , ER_{yy} , ER_{zz} , UR_{xx} , UR_{yy} , and UR_{zz} using the $2\times$ grid technique is a four-step process. First, the grid parameters for the standard Yee grid are calculated. This includes the number of cells wide N_x , number of cells tall N_y , and the resolution parameters dx and dy . Second, the grid parameters for the $2\times$ grid are calculated directly from the Yee grid parameters so that the same physical amount of space is represented with twice as many cells.

```
Nx2 = 2*Nx;
Ny2 = 2*Ny;
dx2 = dx/2;
dy2 = dy/2;
```

Third, the device to be simulated is built onto the $2\times$ grid without having to consider anything about the Yee grid or staggering. This is easily accomplished using any of the techniques presented in Chapter 1. When this is done, two arrays are produced. $ER2$ is the array containing the relative permittivity across the $2\times$ grid and $UR2$ is the array containing the relative permeability across the $2\times$ grid. Fourth, the tensor elements ER_{xx} , ER_{yy} , and ER_{zz} across the Yee grid are extracted from the $2\times$ grid array $ER2$ using the three lines of code below.

```
ERxx = ER2(2:2:Nx2,1:2:Ny2);
ERyy = ER2(1:2:Nx2,2:2:Ny2);
ERzz = ER2(1:2:Nx2,1:2:Ny2);
```

Each of these lines of code pulls numbers from every other cell in $ER2$. The difference is the starting index being a 1 or a 2, and these are determined by examining Figure 4.7. The constitutive values for $\epsilon_{xx}|_{i,j}$ reside at the same points as $E_x|_{i,j}$, and the first occurrence of $E_x|_{i,j}$ on the $2\times$ grid is at $i = 2$ and $j = 1$. Therefore, 2 and 1 are used as the starting indices for where the array ER_{xx} is extracted from the array $ER2$. A similar examination will show that ER_{yy} is extracted from $ER2$ with starting indices $i = 1$ and $j = 2$, and ER_{zz} is extracted from $ER2$ with starting indices $i = 1$ and $j = 1$.

Last, the tensor elements UR_{xx} , UR_{yy} , and UR_{zz} across the Yee grid are extracted from the $2\times$ grid array $UR2$ using the three lines of code below.

```
URxx = UR2(1:2:Nx2,2:2:Ny2);
URyy = UR2(2:2:Nx2,1:2:Ny2);
URzz = UR2(2:2:Nx2,2:2:Ny2);
```

Each of these lines of code pulls numbers from every other cell of $UR2$. The constitutive values for $\mu_{xx}|_{i,j}$ reside at the same points as $\tilde{H}_x|_{i,j}$, and the first occurrence of $\tilde{H}_x|_{i,j}$ on the $2\times$ grid is at $i = 1$ and $j = 2$. Therefore, 1 and 2 are used as the starting indices for where the array UR_{xx} is extracted from the array $UR2$. A similar examination will show that UR_{yy} is extracted from $UR2$ with starting indices $i = 2$ and $j = 1$, and UR_{zz} is extracted from $UR2$ with starting indices $i = 2$ and $j = 2$.

It is best to illustrate the $2\times$ grid technique by example. Suppose it is desired to simulate scattering from a cylinder. In this case, a circle has to be constructed onto a two-dimensional Yee grid and values for permittivity and permeability assigned

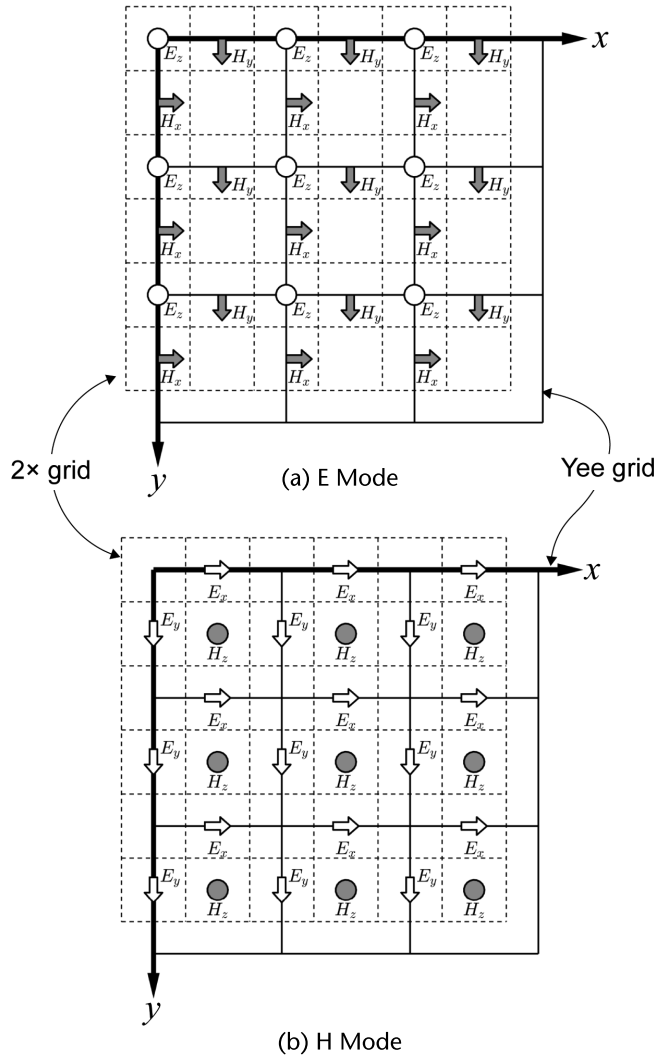


Figure 4.7 2× grid overlaid onto the Yee grid for both E and H modes.

correctly to the six tensor arrays. Figure 4.8(a) shows a 5×5 Yee grid indicated by solid lines, the field components for both E and H modes distributed throughout the grid, and the 2× grid indicated by dashed lines. The highlighted cells in the background convey the device that was constructed onto the 2× grid. In this example, the 2× grid arrays ER2 and UR2 have the same pattern, but this does not have to be the case and it does not mean they will ultimately have the same values stored in them. The column of diagrams in Figure 4.8(c) shows the arrays ERxx, ERyy, and ERzz which are extracted from the array ER2. The column of diagrams in Figure 4.8(b) shows the arrays URxx, URyy, and URzz which are extracted from the array UR2. Observe the arrays do not look similar. In fact, they are different for the cells where the edge of the circle slices through the middle of the Yee cells. Constructing the tensor arrays individually for the Yee grid would be a more complicated and tedious task, but the 2× grid technique makes it simple and intuitive.

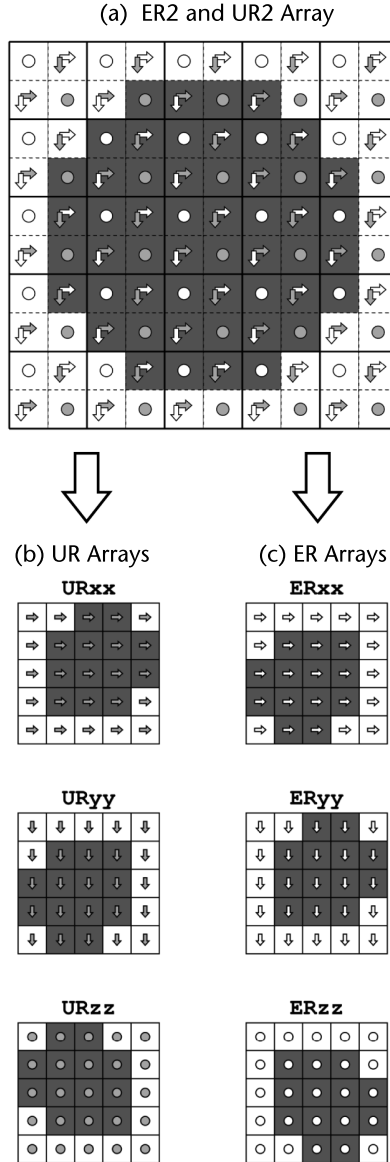


Figure 4.8 Extracting the tensor element arrays from the $2\times$ grid arrays. (a) Original array on $2\times$ grid. (b) Relative permeability tensor arrays UR_{xx} , UR_{yy} , and UR_{zz} extracted from the $2\times$ grid. (c) Relative permittivity tensor arrays ER_{xx} , ER_{yy} , and ER_{zz} extracted from the $2\times$ grid.

When FDFD simulations are performed for three-dimensional simulations, the equations to extract the tensor arrays from the three-dimensional $2\times$ grid are

$$\begin{aligned} ER_{xx} &= ER2(2:2:Nx2, 1:2:Ny2, 1:2:Nz2); \\ ER_{yy} &= ER2(1:2:Nx2, 2:2:Ny2, 1:2:Nz2); \\ ER_{zz} &= ER2(1:2:Nx2, 1:2:Ny2, 2:2:Nz2); \end{aligned}$$

$$\begin{aligned} UR_{xx} &= UR2(1:2:Nx2, 2:2:Ny2, 2:2:Nz2); \\ UR_{yy} &= UR2(2:2:Nx2, 1:2:Ny2, 2:2:Nz2); \\ UR_{zz} &= UR2(2:2:Nx2, 2:2:Ny2, 1:2:Nz2); \end{aligned}$$

Furthermore, when full nine-element tensors are used, the equations to extract the tensor arrays from the three-dimensional $2 \times$ grid are

```
ERxx = ER2xx(2:2:Nx2,1:2:Ny2,1:2:Nz2);
ERxy = ER2xy(1:2:Nx2,2:2:Ny2,1:2:Nz2);
ERxz = ER2xz(1:2:Nx2,1:2:Ny2,2:2:Nz2);
ERyx = ER2yx(2:2:Nx2,1:2:Ny2,1:2:Nz2);
ERyy = ER2yy(1:2:Nx2,2:2:Ny2,1:2:Nz2);
ERyz = ER2yz(1:2:Nx2,1:2:Ny2,2:2:Nz2);
ERzx = ER2zx(2:2:Nx2,1:2:Ny2,1:2:Nz2);
ERzy = ER2zy(1:2:Nx2,2:2:Ny2,1:2:Nz2);
ERzz = ER2zz(1:2:Nx2,1:2:Ny2,2:2:Nz2);
```

```
URxx = UR2xx(1:2:Nx2,2:2:Ny2,2:2:Nz2);
URxy = UR2xy(2:2:Nx2,1:2:Ny2,2:2:Nz2);
URxz = UR2xz(2:2:Nx2,2:2:Ny2,1:2:Nz2);
URyx = UR2yx(1:2:Nx2,2:2:Ny2,2:2:Nz2);
URyy = UR2yy(2:2:Nx2,1:2:Ny2,2:2:Nz2);
URyz = UR2yz(2:2:Nx2,2:2:Ny2,1:2:Nz2);
URzx = UR2zx(1:2:Nx2,2:2:Ny2,2:2:Nz2);
URzy = UR2zy(2:2:Nx2,1:2:Ny2,2:2:Nz2);
URzz = UR2zz(2:2:Nx2,2:2:Ny2,1:2:Nz2);
```

4.10 Numerical Dispersion

A simulated wave in FDFD propagates at a slightly slower speed than a physical wave due to *numerical dispersion*. Numerical dispersion is a nonphysical dispersion exhibited by the Yee grid [2] and is illustrated in Figure 4.9. In this figure, the analytical wave characterized by the wave vector \vec{k} is compared to the numerical wave at the same frequency characterized by the wave vector \vec{k}' . The numerical wave is slower and so it exhibits a compressed wavelength. The severity of the numerical dispersion depends on frequency, direction of the wave, and grid resolution. The general tendency of numerical dispersion is to shift the frequency response of devices to slightly lower frequencies. When simulating very large structures, numerical dispersion produces particularly severe errors when extracting phase information from a simulation.

In Chapter 2, the dispersion relation was derived for an analytical wave by substituting the expression for a plane wave into Maxwell's equations for a linear, homogeneous, and isotropic (LHI) medium. This gave

$$\left(\frac{\omega n}{c_0}\right)^2 = k_x^2 + k_y^2 + k_z^2 \quad (4.122)$$

A similar numerical dispersion relation is derived by substituting the expression for a numerical plane wave into the discrete form of Maxwell's equations in

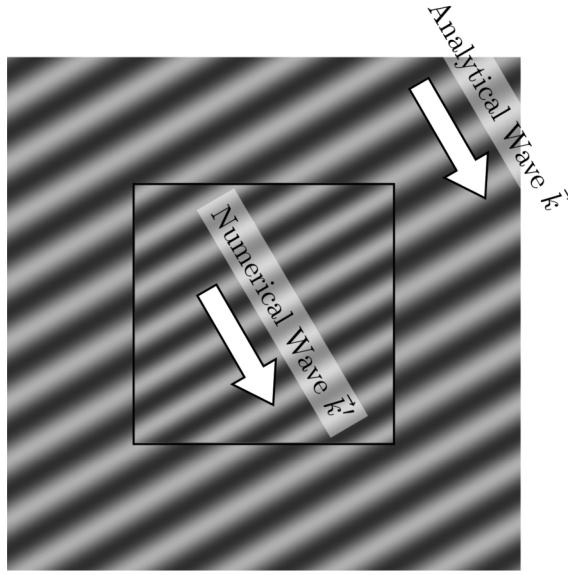


Figure 4.9 Illustration of the concept of numerical dispersion caused by the Yee grid. Despite having the same frequency, the analytical and numerical waves propagate at different velocities characterized by the wave vectors \vec{k} and \vec{k}' . The difference has been exaggerated in this figure for illustration purposes.

an LHI medium. The following equations give the expression for a numerical plane wave for the E mode.

$$E_z|_{I,J} = E_{z0} e^{-j[k'_x I \Delta x + k'_y J \Delta y]} \quad (4.123)$$

$$\tilde{H}_x|_{I,J} = \tilde{H}_{x0} e^{-j[k'_x I \Delta x + k'_y (J+0.5) \Delta y]} \quad (4.124)$$

$$\tilde{H}_y|_{I,J} = \tilde{H}_{y0} e^{-j[k'_x (I+0.5) \Delta x + k'_y J \Delta y]} \quad (4.125)$$

In these expressions, the wave vector components are written with a prime superscript to indicate that they are the numerical wave vector components for the numerical wave that propagates at a slightly slower velocity than the physical wave. Substituting these expressions into (4.61) to (4.63) written for an LHI medium described by relative permeability μ_r and relative permittivity ϵ_r gives

$$\frac{E_{z0} e^{-j[k'_x I \Delta x + k'_y (J+1) \Delta y]} - E_{z0} e^{-j[k'_x I \Delta x + k'_y J \Delta y]}}{\Delta y} = k_0 \mu_r \tilde{H}_{x0} e^{-j[k'_x I \Delta x + k'_y (J+0.5) \Delta y]} \quad (4.126)$$

$$-\frac{E_{z0}e^{-j[k'_x(I+1)\Delta x+k'_yJ\Delta y]} - E_{z0}e^{-j[k'_xI\Delta x+k'_yJ\Delta y]}}{\Delta x} = k_0\mu_r\tilde{H}_{y0}e^{-j[k'_x(I+0.5)\Delta x+k'_yJ\Delta y]} \quad (4.127)$$

$$\frac{\tilde{H}_{y0}e^{-j[k'_x(I+0.5)\Delta x+k'_yJ\Delta y]} - \tilde{H}_{y0}e^{-j[k'_x(I-0.5)\Delta x+k'_yJ\Delta y]}}{\Delta x} \quad (4.128)$$

$$-\frac{\tilde{H}_{x0}e^{-j[k'_xI\Delta x+k'_y(J+0.5)\Delta y]} - \tilde{H}_{x0}e^{-j[k'_xI\Delta x+k'_y(J-0.5)\Delta y]}}{\Delta y} = k_0\varepsilon_rE_{z0}e^{-j[k'_xI\Delta x+k'_yJ\Delta y]}$$

Next, (4.126) is solved for \tilde{H}_{x0} , (4.127) is solved for \tilde{H}_{y0} , (4.128) is solved for E_{z0} , and each of the new expressions is simplified to get

$$\tilde{H}_{x0} = \frac{E_{z0}}{k_0\mu_r\Delta y} 2j \sin\left(\frac{k'_y\Delta y}{2}\right) \quad (4.129)$$

$$\tilde{H}_{y0} = -\frac{E_{z0}}{k_0\mu_r\Delta x} 2j \sin\left(\frac{k'_x\Delta x}{2}\right) \quad (4.130)$$

$$E_{z0} = \frac{\tilde{H}_{y0}}{k_0\varepsilon_r\Delta x} 2j \sin\left(\frac{k'_x\Delta x}{2}\right) - \frac{\tilde{H}_{x0}}{k_0\varepsilon_r\Delta y} 2j \sin\left(\frac{k'_y\Delta y}{2}\right) \quad (4.131)$$

Last, (4.129) and (4.130) are substituted into (4.131) to arrive at the numerical dispersion relation for the E mode.

$$k_0^2\mu_r\varepsilon_r = \left[\frac{2}{\Delta x} \sin\left(\frac{k'_x\Delta x}{2}\right) \right]^2 + \left[\frac{2}{\Delta y} \sin\left(\frac{k'_y\Delta y}{2}\right) \right]^2 \quad (4.132)$$

Recognizing that $k_0 = \omega/c_0$ and $n = \mu_r\varepsilon_r$, (4.132) can be written in a form that looks more like the analytical dispersion relation in (4.122).

$$\left(\frac{\omega n}{c_0} \right)^2 = \left[\frac{2}{\Delta x} \sin\left(\frac{k'_x\Delta x}{2}\right) \right]^2 + \left[\frac{2}{\Delta y} \sin\left(\frac{k'_y\Delta y}{2}\right) \right]^2 \quad (4.133)$$

Figure 4.10 shows the comparison of the numerical dispersion described by (4.133) and the analytical dispersion described by (4.122) for a Yee grid where the grid resolution is $\Delta x = 0.35\lambda$ and $\Delta y = 0.15\lambda$, and the background refractive index is $n = 1.0$. In this figure, the analytical and numerical wave vectors are plotted as a function of direction. Larger values of k correspond to slower waves. Numerical error in a simulation arises because these lines are different. Observe that numerical dispersion is greatest in the directions with poorest grid resolution. In this case, the grid resolution in the x -direction is around three times lower than the y -direction, so the greatest discrepancy between the analytical and numerical waves are experienced in the x -direction.

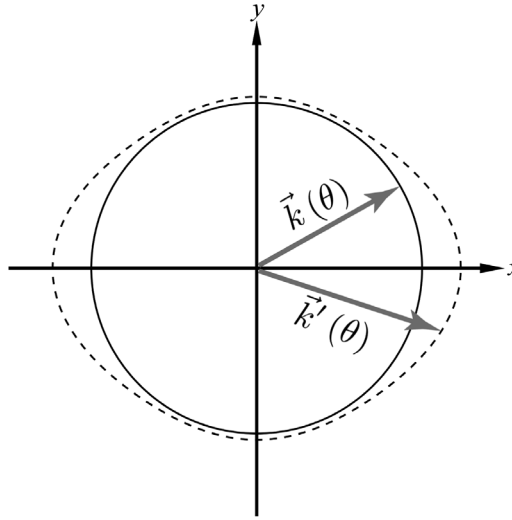


Figure 4.10 Plot of analytical and numerical wave vectors as a function of direction on a two-dimensional Yee grid where $\Delta x = 0.35\lambda$, $\Delta y = 0.15\lambda$, and $n = 1$.

The same numerical dispersion relation as (4.133) would be derived if the H mode were analyzed. This is recommended as an exercise to the reader. It is straightforward to generalize the numerical dispersion relation to full three dimensions. This is

$$\left(\frac{\omega n}{c_0}\right)^2 = \left[\frac{2}{\Delta x} \sin\left(\frac{k'_x \Delta x}{2}\right)\right]^2 + \left[\frac{2}{\Delta y} \sin\left(\frac{k'_y \Delta y}{2}\right)\right]^2 + \left[\frac{2}{\Delta z} \sin\left(\frac{k'_z \Delta z}{2}\right)\right]^2 \quad (4.134)$$

Observe that in the limit as Δx , Δy , and Δz approach zero, the numerical dispersion relation becomes exactly the analytical dispersion relation. This shows that numerical dispersion can be reduced by decreasing the values of Δx , Δy , and Δz . However, this would require more points on the grid, greater memory requirements, and slower simulations. Other solutions include using finite-difference approximations with higher-order accuracy [5] or using a hexagonal grid [6, 7]. In Chapter 8, an alternative technique will be presented to compensate for numerical dispersion that does not require any of this.

To get a feel for the numbers, let $n = 1$, $\Delta x = 0.1\lambda_0$, and $k'_y = 0$. Under these conditions, (4.133) reduces to

$$1 = \frac{10}{\pi} \sin\left(\frac{k'_x \lambda_0}{20}\right) \quad (4.135)$$

The numerical wave vector component k'_x differs from the analytical wave vector component k_x by a constant ψ according to $k_x = \psi k'_x$. In addition, since $n = 1$ the analytical wave vector component is simply $k_x = k_0 = 2\pi/\lambda_0$. Applying these realizations to (4.135) gives

$$1 = \frac{10}{\pi} \sin\left(\frac{\pi}{10\psi}\right) \quad (4.136)$$

Solving (4.136) for ψ gives the numerical value for ψ .

$$\psi = \frac{\pi/10}{\sin^{-1}(\pi/10)} = 0.9831 \quad (4.137)$$

This value for ψ means the wave on the Yee grid is propagating around 1.7% slower than a physical wave. At a minimum, this would shift the spectrum of a device to a 1.7% lower frequency. Resonance and other wave phenomena occurring in real devices tend to amplify the effects caused by numerical dispersion and produce artifacts more serious than just shifting the spectral response of the device.

References

- [1] Yee, K., "Numerical Solution of Initial Boundary Value Problems Involving Maxwell's Equations in Isotropic Media," *IEEE Trans. on Antennas and Propagation*, Vol. 14, No. 3, 1966, pp. 302–307.
- [2] Taflov, A., and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Norwood, MA: Artech House, 2005.
- [3] Rumpf, R. C., "Engineering the Dispersion and Anisotropy of Periodic Electromagnetic Structures," *Solid State Physics*, Vol. 66, 2015, pp. 213–300.
- [4] Rumpf, R. C., "Simple Implementation of Arbitrarily Shaped Total-Field/Scattered-Field Regions in Finite-Difference Frequency-Domain," *Progress in Electromagnetics Research*, Vol. 36, 2012, pp. 221–248.
- [5] Kuzu, L., "A Fourth-Order Accurate Compact 2-D FDFD Method for Waveguide Problems," *Turkish Journal of Electrical Engineering & Computer Sciences*, Vol. 23, No. 4, 2015, pp. 1001–1008.
- [6] Aghaie, K. Z., S. Fan, and M. J. Digonnet, "Birefringence Analysis of Photonic-Bandgap Fibers Using the Hexagonal Yee's Cell," *IEEE J. of Quantum Electronics*, Vol. 46, No. 6, 2010, pp. 920–930.
- [7] Guo, S., *et al.*, "Photonic Band Gap Analysis Using Finite-Difference Frequency-Domain Method," *Optics Express*, Vol. 12, No. 8, 2004, pp. 1741–1746.

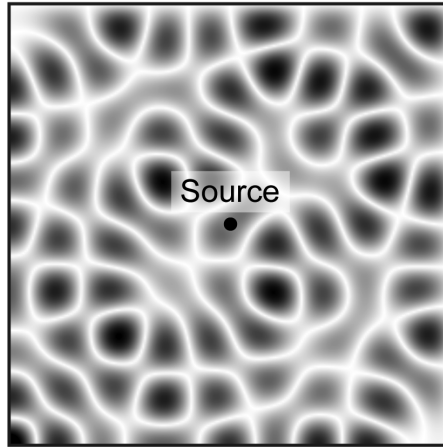
The Perfectly Matched Layer Absorbing Boundary

This chapter will derive and describe how to implement a perfectly matched layer (PML) absorbing boundary [1] in the finite-difference frequency-domain (FDFD) method. The uniaxial PML (UPML) [2, 3] will be derived and explained how it physically absorbs waves. The UPML will be incorporated into Maxwell's equations in a way that only entails modifying the permittivity and permeability arrays. This simplifies things considerably because FDFD can be formulated and implemented with minimal consideration of the UPML. Calculating the various PML parameters will be covered afterward along with a MATLAB function that calculates the PML terms and incorporates a UPML into the permittivity and permeability arrays for FDFD. Last, the stretched-coordinate PML (SCPML) [4, 5] will be described that is more complicated to implement, but improves the conditioning of the matrices, allowing large and three-dimensional FDFD problems to be solved by iteration. The SCPML will only be used in Chapter 10.

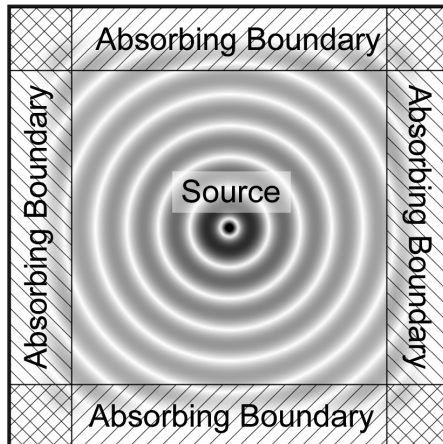
5.1 The Absorbing Boundary

During the course of a simulation, waves will propagate outward until they reach the boundaries of the grid. In most cases, it is desired to prevent the waves from reflecting from the boundaries of the grid and reentering the simulation domain. If reflections from the boundaries are not eliminated, it becomes difficult or impossible to distinguish between what was reflected from a device being simulated from what was reflected from the boundaries. The purpose of the absorbing boundary is to make it appear as if the outward propagating waves keep propagating out to infinity. Waves will be absorbed by incorporating loss into the outer cells of the grid in a special way that does not produce additional reflections. Figure 5.1 shows the field calculated from two FDFD simulations of a point source centered on the grid. The only difference between the two simulations is the use of an absorbing boundary. The field from the simulation without an absorbing boundary is completely crazy due to all of the reflections. Imagine if you were trying to calculate something meaningful about a device from this simulation! When the absorbing boundary is added, the cylindrical wave emitting from the point source is clear and obvious.

An enormous amount of research has been carried out over the years to develop an efficient absorbing boundary [6, 7]. Today, the PML [1, 8] has emerged as the



(a) Point source without an absorbing boundary.



(b) Point source with an absorbing boundary.

Figure 5.1 (a) FDFD simulation of a point source without an absorbing boundary. (b) Same simulation with an absorbing boundary.

state-of-the-art absorbing boundary. When compared to other absorbing boundaries, PMLs are simple to implement, numerically efficient, can be applied to any mesh or geometry, work with a wide variety of wave problems, and their performance can be controlled. Other absorbing boundaries can sometimes be made to work better or more efficiently than a PML for special cases, but the PML is considered superior because it can work well over a wide variety of conditions. In most parts of this book, the UPML will be used because it is the easiest to implement in FDFD and the most intuitive to understand. An arguably better PML for FDFD is the SCPML. When FDFD is solved iteratively instead of by direct matrix division, the SCPML offers better matrix conditioning than the UPML [9]. The SCPML, however, is more complicated to implement in FDFD and is more difficult to understand physically than the UPML.

5.2 Derivation of the UPML Absorbing Boundary

To absorb outgoing waves, the UPML will incorporate loss into the outer regions of the simulation domain. However, this must be done in a special manner that does not introduce reflections from the lossy regions themselves. The ideal absorbing boundary will prevent reflections from itself and absorb waves for all polarizations and all angles of incidence.

To see if this is even possible, examine the Fresnel equations in (5.1) that calculate the reflection of a wave incident from the simulation domain (medium 1) onto the absorbing layer (medium 2). In a simulation, the impedance of the simulation domain η_1 is determined by what is being simulated, so η_1 is not a variable that can be modified to implement the absorbing layer. In addition, the absorbing layer must prevent reflections for all angles of incidence θ_1 , so neither θ_1 nor θ_2 are variables that the absorbing layer can modify. The only degree of freedom that remains in these equations to implement an absorbing boundary is η_2 . After a careful examination of (5.1), it becomes apparent that there is no choice for η_2 that will prevent reflections for both polarizations and all angles of incidence at the same time.

$$r_{\text{TE}} = \frac{\eta_2 \cos \theta_1 - \eta_1 \cos \theta_2}{\eta_2 \cos \theta_1 + \eta_1 \cos \theta_2} \quad r_{\text{TM}} = \frac{\eta_2 \cos \theta_2 - \eta_1 \cos \theta_1}{\eta_2 \cos \theta_2 + \eta_1 \cos \theta_1} \quad (5.1)$$

More degrees of freedom are needed to achieve the ideal absorbing boundary. It will turn out to be sufficient if the absorbing boundary is made *doubly diagonally anisotropic*. This means that both the permittivity and permeability will have to be made diagonally anisotropic as discussed in Chapter 2. It will then be shown that the absorbing medium should actually have uniaxial anisotropy, leading to the absorbing medium being called a UPML. It will be convenient to set the impedance η_2 of the UPML equal to the vacuum impedance. The most straightforward way to best match η_2 to vacuum for a general anisotropic medium is to set the permittivity and permeability tensors equal to each other. Equation (5.2) defines a general tensor for the UPML. The only things that remain to be determined are the tensor elements a , b , and c . At this moment, the tensor is biaxial and not yet uniaxial because a , b , and c may all be different values.

$$[\epsilon_2] = [\mu_2] = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \quad (5.2)$$

The standard Fresnel equations for reflection can no longer be used because those were only valid for isotropic media. Building on Figure 2.4(b), let the interface between the simulation domain and the absorbing medium lie in the xy plane. When a wave is incident from vacuum onto this doubly-anisotropic medium, Snell's law of refraction and the Fresnel equations for reflection become [2]

$$\sin \theta_1 = \sqrt{bc} \sin \theta_2 \quad (5.3)$$

$$r_{\text{TE}} = \frac{\sqrt{a} \cos \theta_1 - \sqrt{b} \cos \theta_2}{\sqrt{a} \cos \theta_1 + \sqrt{b} \cos \theta_2} \quad r_{\text{TM}} = \frac{\sqrt{b} \cos \theta_2 - \sqrt{a} \cos \theta_1}{\sqrt{b} \cos \theta_2 + \sqrt{a} \cos \theta_1} \quad (5.4)$$

From (5.3), refraction and reflection can be made independent of the angle of incidence θ_1 if b and c are chosen such that $bc = 1$. With this choice, (5.3) and (5.4) reduce to

$$\theta_1 = \theta_2 \quad (5.5)$$

$$r_{\text{TE}} = \frac{\sqrt{a} - \sqrt{b}}{\sqrt{a} + \sqrt{b}} \quad r_{\text{TM}} = \frac{\sqrt{b} - \sqrt{a}}{\sqrt{b} + \sqrt{a}} \quad (5.6)$$

Observe that angles θ_1 and θ_2 have been completely eliminated from the modified Fresnel equations indicating that they are angle independent when $bc = 1$. From (5.6), reflection for both polarizations can be made perfectly zero if $a = b$. Given that the parameters a , b , and c are now related through $a = b = 1/c$, the tensor from (5.2) can be written in terms of a single parameter. Any of the three variables can be used, but (5.7) expresses the tensor using a .

$$[\epsilon_2] = [\mu_2] = \begin{bmatrix} a & 0 & 0 \\ 0 & a & 0 \\ 0 & 0 & \frac{1}{a} \end{bmatrix} \quad (5.7)$$

Recall that this tensor was derived for a wave propagating primarily in the z -direction. Instead of expressing the tensor using the variable a , it will be expressed using the variable s_z , where the z subscript indicates this is for a wave propagating primarily in the z -direction incident onto a z -axis boundary.

$$[\epsilon_2] = [\mu_2] = \begin{bmatrix} s_z & 0 & 0 \\ 0 & s_z & 0 \\ 0 & 0 & \frac{1}{s_z} \end{bmatrix} \quad \text{For waves propagating in } z\text{-direction} \quad (5.8)$$

A similar procedure can be repeated for a wave propagating primarily in the x -direction that is incident onto an x -axis boundary. This would lead to the tensor elements being related through $1/a = b = c$. This tensor will be expressed using the variable s_x .

$$[\epsilon_2] = [\mu_2] = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & s_x \end{bmatrix} \quad \text{For waves propagating in } x\text{-direction} \quad (5.9)$$

The procedure can be repeated one last time for a wave propagating primarily in the y -direction that is incident onto a y -axis boundary. This would lead to the tensor elements being related through $a = 1/b = c$. This tensor will be expressed using the variable s_y .

$$[\epsilon_2] = [\mu_2] = \begin{bmatrix} s_y & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & s_y \end{bmatrix} \quad \text{For waves propagating in } y\text{-direction} \quad (5.10)$$

Compare the elements in the tensors in (5.8) to (5.10). Each is uniaxial but the position of the inverse PML term $1/s_i$ is different for each case along with the subscripts on the PML terms. In order to absorb all waves at all boundaries, all of the above tensors are multiplied to form a single tensor describing the UPML.

$$[\epsilon_{r,\text{UPML}}] = [\mu_{r,\text{UPML}}] = [S] = \begin{bmatrix} \frac{1}{s_x} & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & s_x \end{bmatrix} \begin{bmatrix} s_y & 0 & 0 \\ 0 & \frac{1}{s_y} & 0 \\ 0 & 0 & s_y \end{bmatrix} \begin{bmatrix} s_z & 0 & 0 \\ 0 & s_z & 0 \\ 0 & 0 & \frac{1}{s_z} \end{bmatrix} \quad (5.11)$$

$$= \begin{bmatrix} \frac{s_y s_z}{s_x} & 0 & 0 \\ 0 & \frac{s_x s_z}{s_y} & 0 \\ 0 & 0 & \frac{s_x s_y}{s_z} \end{bmatrix}$$

From this derivation, the UPML parameters s_x , s_y , and s_z are interpreted as constitutive parameters like permittivity and permeability. In Chapter 2, the loss was incorporated into the permittivity through the electrical conductivity σ as $\tilde{\epsilon} = \epsilon_r +$

$\sigma/j\omega\epsilon_0$. This gives the start of a recipe for how to make the UPML parameters have a loss to absorb outgoing waves. This will be discussed in more detail in Section 5.4.

5.3 Incorporating the UPML into Maxwell's Equations

In the previous section, the UPML parameters were derived as constitutive parameters like permittivity and permeability. This implies the UPML parameters should be incorporated into Maxwell's equations the same way as the actual constitutive parameters. With a UPML tensor $[S]$, Maxwell's curl equations are modified according to

$$\nabla \times \vec{E} = -j\omega[S][\mu]\vec{H} \quad (5.12)$$

$$\nabla \times \vec{H} = j\omega[S][\epsilon]\vec{E} \quad (5.13)$$

When the tensor quantities are multiplied to combine them, (5.12) and (5.13) are written as

$$\nabla \times \vec{E} = -j\omega[\mu']\vec{H} \quad (5.14)$$

$$\nabla \times \vec{H} = j\omega[\epsilon']\vec{E} \quad (5.15)$$

where $[\mu'] = [S][\mu]$ and $[\epsilon'] = [S][\epsilon]$. Observe that (5.14) and (5.15) have the same form as the ordinary curl equations, as if there were no UPML incorporated. This means that FDFD can be formulated and implemented with hardly considering the UPML at all. This is the primary advantage of the UPML. Incorporating the UPML will involve one simple step in the algorithm where the PML terms are incorporated into the permittivity and permeability tensors according to

$$[\mu'] = [S][\mu] = \begin{bmatrix} \frac{s_y s_z}{s_x} \mu_{xx} & 0 & 0 \\ 0 & \frac{s_x s_z}{s_z} \mu_{yy} & 0 \\ 0 & 0 & \frac{s_x s_y}{s_z} \mu_{zz} \end{bmatrix} \quad (5.16)$$

$$[\epsilon'] = [S][\epsilon] = \begin{bmatrix} \frac{s_y s_z}{s_x} \epsilon_{xx} & 0 & 0 \\ 0 & \frac{s_x s_z}{s_y} \epsilon_{yy} & 0 \\ 0 & 0 & \frac{s_x s_y}{s_z} \epsilon_{zz} \end{bmatrix} \quad (5.17)$$

For two-dimension simulations, a UPML at the z -axis boundaries is not needed. For this case, $s_z = 1$ and the permittivity and permeability tensors reduce to

$$[\mu'] = [S][\mu] = \begin{bmatrix} \frac{s_y}{s_x} \mu_{xx} & 0 & 0 \\ 0 & \frac{s_x}{s_y} \mu_{yy} & 0 \\ 0 & 0 & s_x s_y \mu_{zz} \end{bmatrix} \quad (5.18)$$

$$[\epsilon'] = [S][\epsilon] = \begin{bmatrix} \frac{s_y}{s_x} \epsilon_{xx} & 0 & 0 \\ 0 & \frac{s_x}{s_y} \epsilon_{yy} & 0 \\ 0 & 0 & s_x s_y \epsilon_{zz} \end{bmatrix} \quad (5.19)$$

5.4 Calculating the UPML Parameters

Up to this point, it has only been mentioned that the UPML parameters are complex numbers to incorporate loss just like a complex permittivity. When incorporating the loss, it is best to gradually increase the loss into the PML [10]. Numerical problems can arise at the entrance of an abrupt PML that cause reflections. In addition, profiling the PML reduces reflections of waves that are incident onto the PML at larger angles. With this in mind, there has been much research to determine the best values and optimum profiles for the PML parameters [11–16]. If the UPML terms are interpreted as constitutive parameters, they should be defined as such. This implies that for the negative sign convention the imaginary part should be negative, $s = \text{Re}[s] - j\text{Im}[s]$. Recognizing that $k_0 = \omega\sqrt{\mu_0\epsilon_0}$ and $\eta_0 = \sqrt{\mu_0/\epsilon_0}$, the UPML term can be written as $s = 1 - j\eta_0\sigma/k_0$. If the FDFD analysis is normalizing its parameters properly, then $\lambda_0 \approx 1$ and $k_0 \approx 2\pi$. Now the UPML term reduces to $s \approx 1 - j60\sigma$. In order to taper this basic definition of s , the following equations will be used for calculating the UPML terms.

$$\begin{aligned} s_x(x) &= a_x(x)[1 - j60\sigma_x(x)] \\ s_y(y) &= a_y(y)[1 - j60\sigma_y(y)] \end{aligned} \quad (5.20)$$

The functions $\sigma_x(x)$ and $\sigma_y(y)$ quantify the artificial conductivity associated with the UMPL and are not intended to represent physical conductivities associated with devices in the simulation. Observe the UPML conductivities are functions of x and y . This is because the UPML conductivities will be tapered from zero where a wave

enters the UPML up to some maximum value σ_{\max} at the far side of the UPML. For this book, the following profile for the UPML conductivity functions will be used.

$$\begin{aligned}\sigma_x(x) &= \sigma_{\max} \sin^2\left(\frac{\pi x}{2L_x}\right) \\ \sigma_y(y) &= \sigma_{\max} \sin^2\left(\frac{\pi y}{2L_y}\right)\end{aligned}\tag{5.21}$$

In these equations, L_x is the physical size of the UPML extending in the x -direction and L_y is the physical size of the UPML extending in the y -direction. The ratios x/L_x and y/L_y are quantities that range from 0 to 1 from the start to the end of the UPML. Interpreting the ratios this way will be very useful in the computer code because the same ratios can be calculated directly from array indices without having to calculate physical distance or size. The conductivity profiles $\sigma_x(x)$ and $\sigma_y(y)$ calculated from (5.21) are shown in Figure 5.2. They are equal to zero through most of the simulation domain and only become nonzero inside of the UPML regions.

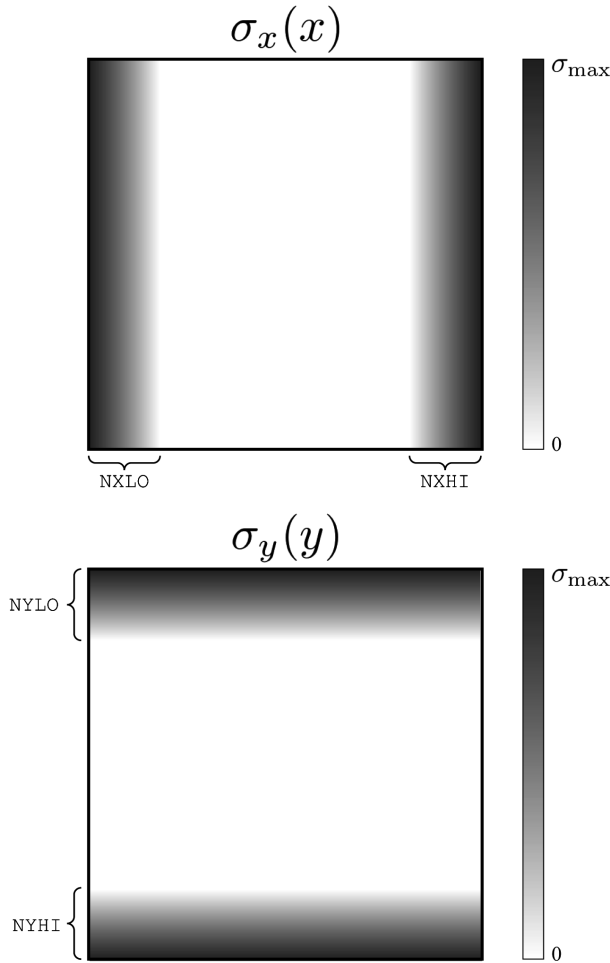


Figure 5.2 Conductivity profiles for the UPML.

The functions $a_x(x)$ and $a_y(y)$ in (5.20) scale the overall magnitude of the PML functions $s_x(x)$ and $s_y(y)$. Even the amplitude functions are tapered following a profile given in (5.22) which is controlled by the parameter p . The functions $a_x(x)$ and $a_y(y)$ range from 1 where the wave enters the UPML up to some maximum value a_{\max} at the end of the UPML. If visualized, they would look similar to that shown in Figure 5.2, but would have a value of 1 throughout most of the simulation domain and only being larger than 1 inside of the UPML regions. Observe that (5.22) makes use of the same ratios x/L_x and y/L_y that will be calculated from array indices in the computer code.

$$\begin{aligned} a_x(x) &= 1 + (a_{\max} - 1) \left(\frac{x}{L_x} \right)^p \\ a_y(y) &= 1 + (a_{\max} - 1) \left(\frac{y}{L_y} \right)^p \end{aligned} \quad (5.22)$$

From (5.21) and (5.22), the UPML is defined from the parameters σ_{\max} , a_{\max} , and p . In this book, the following values will be used: $\sigma_{\max} = 1$, $a_{\max} = 4$, and $p = 3$. The size of the UPML will not be defined in terms of physical size L_x and L_y . Instead, to be consistent with calculating the ratios from array indices, the size of the UPML will be defined in terms of the number of cells on the grid that the UPMLs will occupy. For two-dimensional simulations, up to four UPML regions will be defined. This is because a UPML will be needed at both sides of the grid for both x - and y -axis boundaries. `NXL0` will define the number of cells large the UPML is at the lower x -axis boundary, `NXHI` will define the number of cells large the UPML is at the upper x -axis boundary, `NYL0` will define the number of cells large the UPML is at the lower y -axis boundary, and `NYHI` will define the number of cells large the UPML is at the upper y -axis boundary. The standard size for a UPML is 10 to 20 cells. The larger the UPML regions, the better the UPMLs will perform, but simulations will be larger, require more memory, and take more time to calculate. The different UPML regions do not have to be the same size, but are typically made to be the same size. The size of each UPML can be chosen based on apriori knowledge of where more or less wave power will enter the UPMLs or to more efficiently handle other special cases.

5.5 Implementation of the UPML in MATLAB

To incorporate a UPML into the constitutive parameters, a function will be written in MATLAB called `addupml2d()`. The code for this function is divided into four major sections and can be downloaded at <https://empossible.net/fdfdbook/>. The header for this function extends from lines 1 to 19. The commented text below the function declaration at the top is the text that will be displayed in the command window after typing `help addupml2d` at the command prompt. It will remind you later what the function does and how to use it. It lists and defines both the input and output arguments of the function. The input arguments are the relative permittivity

array ER2 on the $2\times$ grid, the relative permeability array UR2 on the $2\times$ grid, and an array NPML defining the number of cells large for each of the four different UPML regions on the Yee grid. The output arguments are the permittivity and permeability diagonal tensor element arrays ERxx, Eryy, ERzz, URxx, URyy, and URzz on the Yee grid with the UPML incorporated.

The `addupml2d()` function is initialized from lines 21 to 37. This section starts by defining the parameters a_{\max} and σ_{\max} and profile parameter p that controls the tapering of the UPML. In MATLAB, these parameters are named `amax`, `cmax`, and `p`, respectively. Next, the size of the $2\times$ grid is determined from the size of the array ER2. Last, the size of the four different UPML regions is extracted individually from the array NPML. Since the values in the array NPML are for the standard Yee grid and not the $2\times$ grid, each of the elements in NPML is multiplied by two to get the size of the UPML regions on the $2\times$ grid.

The real work in this function happens from lines 39 to 73 where the UPML functions $s_x(x)$ and $s_y(y)$ are calculated as arrays `sx` and `sy`, respectively, on the $2\times$ grid. First, they are initialized to all ones, which are the correct values if no UPML is incorporated. Next, each UPML region is built into `sx` and `sy` separately because each region could potentially be of a different size. The boundary at $n_x=1$ is called the `xlo` UPML. To build the UPML here, a loop is a setup that iterates from $n_x=1$ to $n_x=NXLO$. If the UPML at the `xlo` side is 20 cells large, the loop will iterate from 1 to 20. The first two lines of code inside of the loop calculate the values of $a_x(x)$ and $\sigma_x(x)$ according to (5.21) and (5.22). In MATLAB, the variables are called `ax` and `cx`. The third, and last, line of code in the loop calculates $s_x(x)$ at the current position in the UPML using (5.20). While the loop iterates from 1 to 20, the UPML is filled in by the loop from right to left, or from 20 down to 1. This is the purpose of the array index `NXLO-nx+1` where the UPML function value is written at each iteration of the loop. The next loop does essentially the same, but assigns values to the array `sx` for the `xhi` UPML region. The last two loops repeat the first two loops, but instead, assign values to the array `sy` since they are incorporating the `ylo` and `yhi` UPML regions.

Lines 75 to 95 of the `addupml2d()` function calculate the tensor elements $\epsilon_{xx}(x,y)$, $\epsilon_{yy}(x,y)$, $\epsilon_{zz}(x,y)$, $\mu_{xx}(x,y)$, $\mu_{yy}(x,y)$, and $\mu_{zz}(x,y)$. In MATLAB, the arrays are called ERxx, Eryy, ERzz, URxx, URyy, and URzz, respectively. First, the tensor elements are calculated on the $2\times$ grid with the UPML parameters incorporated following (5.18) and (5.19). Second, the tensor elements on the standard Yee grid are extracted from the tensor elements on the $2\times$ grid.

5.5.1 Using the `addupml2d()` Function

The `addupml2d()` function by itself is not executable. Instead, it is a function that must be called from a MATLAB program that defines the input arguments and passes them to the function so that the UPML can be implemented. The MATLAB program listed below demonstrates how to use the `addupml2d()` function and displays the resulting material tensor arrays with the UPML incorporated. The size of the Yee grid is defined on lines 8 to 10 as $N_x = N_y = 5$. The size of the UPML at

each axis boundary is defined on line 13. Due to the small size of the grid, the PML was set to two cells large at each boundary. The materials arrays `ER2` and `UR2` are set to all 1's corresponding to vacuum on lines 15 to 17. These arrays are defined on the $2\times$ grid so the grid size parameters `Nx` and `Ny` were multiplied by two. The `addupml2d()` function is called on lines 20 and 21 where the input arguments are passed to the function, the UPML is incorporated, and the tensor elements are returned in the arrays `ERxx`, `ERyy`, `ERzz`, `URxx`, `URyy`, and `URzz`.

```

1  % Chapter5_addupml2d_demo.m
2
3  % INITIALIZE MATLAB
4  close all;
5  clc;
6  clear all;
7
8  % DEFINE GRID
9  Nx = 5;
10 Ny = 5;
11
12 % DEFINE PML
13 NPML = [2 2 2 2];
14
15 % BUILD ER2 AND UR2 ARRAYS
16 ER2 = ones(2*Nx,2*Ny);
17 UR2 = ones(2*Nx,2*Ny);
18
19 % CALL ADDUPML2D
20 [ERxx,ERyy,ERzz,URxx,URyy,URzz] ...
21     = addupml2d(ER2,UR2,NPML);
22
23 % DISPLAY THE RESULTS
24 disp('ERxx =');
25 disp(ERxx);
26
27 disp('ERyy =');
28 disp(ERyy);
29
30 disp('ERzz =');
31 disp(ERzz);
32
33 disp('URxx =');
34 disp(URxx);
35
36 disp('URyy =');
37 disp(URyy);
38
39 disp('URzz =');
40 disp(URzz);

```

The material tensors are displayed to the command window from lines 23 to 40. These are arrays and not matrices. The output of this program is

ERxx =

1.0-e+02 *

0.0207 - 0.0001i	0.0036 + 0.0000i	0.0000 + 0.0001i	0.0008 + 0.0001i	0.0100 + 0.0000i
0.2581 - 0.0250i	0.0444 - 0.0036i	0.0001 + 0.0011i	0.0100 + 0.0000i	0.1248 - 0.0117i
0.0400 - 2.4000i	0.0138 - 0.4125i	0.0100 + 0.0000i	0.0105 - 0.0920i	0.0227 - 1.1603i
0.0581 - 0.0010i	0.0100 + 0.0000i	0.0000 + 0.0002i	0.0022 + 0.0002i	0.0281 - 0.0004i
0.0100 + 0.0000i	0.0017 + 0.0000i	0.0000 + 0.0000i	0.0004 + 0.0000i	0.0048 + 0.0000i

ERyy =

1.0e+02 *

0.0207 - 0.0001i	0.2581 - 0.0250i	0.0400 - 2.4000i	0.0581 - 0.0010i	0.0100 + 0.0000i
0.0036 + 0.0000i	0.0444 - 0.0036i	0.0138 - 0.4125i	0.0100 + 0.0000i	0.0017 + 0.0000i
0.0000 + 0.0001i	0.0001 + 0.0011i	0.0100 + 0.0000i	0.0000 + 0.0002i	0.0000 + 0.0000i
0.0008 + 0.0001i	0.0100 + 0.0000i	0.0105 - 0.0920i	0.0022 + 0.0002i	0.0004 + 0.0000i
0.0100 + 0.0000i	0.1248 - 0.0117i	0.0227 - 1.1603i	0.0281 - 0.0004i	0.0048 + 0.0000i

ERzz =

1.0e+04 *

-5.7584 - 0.1920i	-0.9894 - 0.0495i	0.0004 - 0.0240i	-0.2203 - 0.0288i	-2.7838 - 0.1008i
-0.9894 - 0.0495i	-0.1700 - 0.0113i	0.0001 - 0.0041i	-0.0378 - 0.0056i	-0.4783 - 0.0253i
0.0004 - 0.0240i	0.0001 - 0.0041i	0.0001 + 0.0000i	0.0001 - 0.0009i	0.0002 - 0.0116i
-0.2203 - 0.0288i	-0.0378 - 0.0056i	0.0001 - 0.0009i	-0.0084 - 0.0019i	-0.1065 - 0.0142i
-2.7838 - 0.1008i	-0.4783 - 0.0253i	0.0002 - 0.0116i	-0.1065 - 0.0142i	-1.3458 - 0.0526i

URxx =

1.0e+02 *

0.0048 + 0.0000i	0.0004 + 0.0000i	0.0000 + 0.0000i	0.0017 + 0.0000i	0.0100 + 0.0000i
0.0281 - 0.0004i	0.0022 + 0.0002i	0.0000 + 0.0002i	0.0100 + 0.0000i	0.0581 - 0.0010i
0.0227 - 1.1603i	0.0105 - 0.0920i	0.0100 + 0.0000i	0.0138 - 0.4125i	0.0400 - 2.4000i
0.1248 - 0.0117i	0.0100 + 0.0000i	0.0001 + 0.0011i	0.0444 - 0.0036i	0.2581 - 0.0250i
0.0100 + 0.0000i	0.0008 + 0.0001i	0.0000 + 0.0001i	0.0036 + 0.0000i	0.0207 - 0.0001i

URyy =

1.0e+02 *

0.0048 + 0.0000i	0.0281 - 0.0004i	0.0227 - 1.1603i	0.1248 - 0.0117i	0.0100 + 0.0000i
0.0004 + 0.0000i	0.0022 + 0.0002i	0.0105 - 0.0920i	0.0100 + 0.0000i	0.0008 + 0.0001i
0.0000 + 0.0000i	0.0000 + 0.0002i	0.0100 + 0.0000i	0.0001 + 0.0011i	0.0000 + 0.0001i
0.0017 + 0.0000i	0.0100 + 0.0000i	0.0138 - 0.4125i	0.0444 - 0.0036i	0.0036 + 0.0000i
0.0100 + 0.0000i	0.0581 - 0.0010i	0.0400 - 2.4000i	0.2581 - 0.0250i	0.0207 - 0.0001i

URzz =

1.0e+04 *

```

-1.3458 - 0.0526i -0.1065 - 0.0142i 0.0002 - 0.0116i -0.4783 - 0.0253i -2.7838 - 0.1008i
-0.1065 - 0.0142i -0.0084 - 0.0019i 0.0001 - 0.0009i -0.0378 - 0.0056i -0.2203 - 0.0288i
0.0002 - 0.0116i 0.0001 - 0.0009i 0.0001 + 0.0000i 0.0001 - 0.0041i 0.0004 - 0.0240i
-0.4783 - 0.0253i -0.0378 - 0.0056i 0.0001 - 0.0041i -0.1700 - 0.0113i -0.9894 - 0.0495i
-2.7838 - 0.1008i -0.2203 - 0.0288i 0.0004 - 0.0240i -0.9894 - 0.0495i -5.7584 - 0.1920i

```

5.6 The SCPML Absorbing Boundary

The SCPML is an absorbing boundary that will be needed for the FDFD codes presented in Chapter 10. The SCPML will be presented by rearranging the terms of the UPML and interpreting the terms differently. This conversion is not mathematically rigorous, but will help to understand the SCPML more intuitively. Equations (5.12) and (5.13) show how the UPML is incorporated into Maxwell's equations. The SCPML is derived by moving the UPML tensor $[S]$ to the left-hand side of the equations and associating them with the curl operations.

$$[S]^{-1} \nabla \times \vec{E} = -j\omega[\mu]\vec{H} \quad (5.23)$$

$$[S]^{-1} \nabla \times \vec{\tilde{H}} = j\omega[\varepsilon]\vec{E} \quad (5.24)$$

Equations (5.23) and (5.24) are expanded in Cartesian coordinates in order to see all of the terms and vector components explicitly.

$$\begin{bmatrix} \frac{s_x}{s_y s_z} & 0 & 0 \\ 0 & \frac{s_y}{s_x s_z} & 0 \\ 0 & 0 & \frac{s_z}{s_x s_y} \end{bmatrix} \begin{bmatrix} 0 & -\frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & -\frac{\partial}{\partial x} \\ -\frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = -j\omega \begin{bmatrix} \mu_{xx} & \mu_{xy} & \mu_{xz} \\ \mu_{yx} & \mu_{yy} & \mu_{yz} \\ \mu_{zx} & \mu_{zy} & \mu_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \quad (5.25)$$

$$\begin{bmatrix} \frac{s_x}{s_y s_z} & 0 & 0 \\ 0 & \frac{s_y}{s_x s_z} & 0 \\ 0 & 0 & \frac{s_z}{s_x s_y} \end{bmatrix} \begin{bmatrix} 0 & -\frac{\partial}{\partial z} & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial z} & 0 & -\frac{\partial}{\partial x} \\ -\frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = j\omega \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \quad (5.26)$$

Both of these equations have the same matrix multiplication on the left-hand side. Multiplying these together incorporates the PML terms into the curl operation. After reorganizing the terms, the result of the multiplication is

$$\begin{bmatrix} 0 & -\frac{s_x}{s_y} \left(\frac{1}{s_z} \frac{\partial}{\partial z} \right) & \frac{s_x}{s_z} \left(\frac{1}{s_y} \frac{\partial}{\partial y} \right) \\ \frac{s_y}{s_x} \left(\frac{1}{s_z} \frac{\partial}{\partial z} \right) & 0 & -\frac{s_y}{s_z} \left(\frac{1}{s_x} \frac{\partial}{\partial x} \right) \\ -\frac{s_z}{s_x} \left(\frac{1}{s_y} \frac{\partial}{\partial y} \right) & \frac{s_z}{s_y} \left(\frac{1}{s_x} \frac{\partial}{\partial x} \right) & 0 \end{bmatrix} \quad (5.27)$$

Observe the PML terms that have been associated with the partial derivatives in (5.27). ∂x is always multiplied by s_x , ∂y is always multiplied by s_y , and ∂z is always multiplied by s_z . Instead of interpreting the PML terms as lossy constitutive parameters, they are interpreted as factors scaling the coordinates. It is hard to visualize this because the coordinates are being scaled by complex numbers. At an x -axis boundary, the parameter s_x is scaling the x coordinates to absorb outgoing waves. The parameters s_y and s_z are not contributing at x -axis boundaries because they are just equal to 1 except in the small regions where the PMLs may overlap. In this sense, s_x by itself is sufficient to absorb outgoing waves at x -axis boundaries. Similarly, only s_y is needed to absorb outgoing waves at y -axis boundaries and only s_z is needed to absorb outgoing waves at z -axis boundaries. For these reasons, the PML terms that are not scaling coordinates are dropped from the tensor. This is a conceptual step and not a mathematically rigorous step that converts a UPML into an SCPML. Equations (5.25) and (5.26) can now be written in their final form for the SCPML.

$$\begin{bmatrix} 0 & -\frac{1}{s_z} \frac{\partial}{\partial z} & \frac{1}{s_y} \frac{\partial}{\partial y} \\ \frac{1}{s_z} \frac{\partial}{\partial z} & 0 & -\frac{1}{s_x} \frac{\partial}{\partial x} \\ -\frac{1}{s_y} \frac{\partial}{\partial y} & \frac{1}{s_x} \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = -j\omega \begin{bmatrix} \mu_{xx} & \mu_{xy} & \mu_{xz} \\ \mu_{yx} & \mu_{yy} & \mu_{yz} \\ \mu_{zx} & \mu_{zy} & \mu_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} \quad (5.28)$$

$$\begin{bmatrix} 0 & -\frac{1}{s_z} \frac{\partial}{\partial z} & \frac{1}{s_y} \frac{\partial}{\partial y} \\ \frac{1}{s_z} \frac{\partial}{\partial z} & 0 & -\frac{1}{s_x} \frac{\partial}{\partial x} \\ -\frac{1}{s_y} \frac{\partial}{\partial y} & \frac{1}{s_x} \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = j\omega \begin{bmatrix} \epsilon_{xx} & \epsilon_{xy} & \epsilon_{xz} \\ \epsilon_{yx} & \epsilon_{yy} & \epsilon_{yz} \\ \epsilon_{zx} & \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} \quad (5.29)$$

While the SCPML improves the conditioning of the FDFD matrices, the PML terms cannot be absorbed into the permittivity and permeability. They must remain distinct terms in the formulation of FDFD and so the formulation and implementation are more complicated. Fortunately, the calculation of the PML terms for the SCPML remains the same as they were for the UPML.

5.6.1 MATLAB Implementation of `calcpml3d()`

Implementation of the SCPML in Chapter 10 will require a function to calculate the PML terms. The MATLAB code for the `calcpml3d()` function can be downloaded at <https://empossible.net/fdfdbook/>. The header of the function extends from lines 2 to 9 and is what is displayed in the command window if ‘`help calcpml3d`’ is entered at the command prompt. Lines 11 to 31 define the parameters controlling the PML profile and extract the grid size and PML size from the input arguments.

Lines 33 to 81 calculate the PML arrays s_x , s_y , and s_z . This function does not know or care if these are being constructed onto the Yee grid, $2\times$ grid, or something else. It is up to the program calling `calcpml3d()` to handle this aspect. This is virtually identical code to that used in the `addupml2d()` function except that a z dimension is added requiring calculation of the new array s_z and all the PML terms are now three-dimensional arrays.

5.6.2 Using the `calcpml3d()` Function

The `calcpml3d()` function by itself is not executable. Instead, it is a function that must be called from a MATLAB program that defines the input arguments and passes them to the function so that the SCPML can be calculated. The MATLAB program listed below demonstrates how to use the `calcpml3d()` function and displays the resulting SCPML arrays s_x , s_y , and s_z . The array `NGRID` is defined on line 9 to be the size of the $2\times$ grid. The grid was set to $N_x2 = N_y2 = N_z2 = 6$. The size of the SCPML at each axis boundary is defined on line 10 in the array `NPML`. Due to the small size of the grid, the SCPML was set to two cells large at each boundary. The `calcpml3d()` function is called on line 13 where the input arguments are passed to the function, the SCPML arrays are calculated and returned in the arrays s_x , s_y , and s_z .

```

1  % Chapter5_calcpml3d_demo.m
2
3  % INITIALIZE MATLAB
4  close all;
5  clc;
6  clear all;
7
8  % DEFINE INPUT ARGUMENTS
9  NGRID = [6 6 6];
10 NPML = [2 2 2 2 2 2];
11
```



```

12 % CALL CALCPML3D
13 [sx,sy,sz] = calcpml3d(NGRID,NPML);
14
15 % DISPLAY THE RESULTS
16 disp('sx =');
17 disp(sx);
18
19 disp('sy =');
20 disp(sy);
21
22 disp('sz =');
23 disp(sz);

```

The SCPML arrays are displayed to the command window from lines 15 to 23. These are arrays and not matrices. The output of this program is

```

sx =
(:, :, 1) =
    1.0e+02 *
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i

(:, :, 2) =
    1.0e+02 *
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i

(:, :, 3) =
    1.0e+02 *
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i

(:, :, 4) =
    1.0e+02 *
    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i    0.0400 - 2.4000i
    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i    0.0138 - 0.4125i
    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i    0.0100 + 0.0000i

```

[illegible]

[illegible]

(:,:,4) =

1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i
1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i	1.0000 + 0.0000i

(:,:,5) =

1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i
1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i
1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i
1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i
1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i
1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i	1.3750 -41.2500i

(:,:,6) =

1.0e+02 *					
0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i
0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i
0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i
0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i
0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i	0.0400 - 2.4000i

References

- [1] Bérenger, J.-P., "Perfectly Matched Layer (PML) for Computational Electromagnetics," *Synthesis Lectures on Computational Electromagnetics*, Vol. 2, No. 1, 2007, pp. 1–117.
- [2] Sacks, Z. S., *et al.*, "A Perfectly Matched Anisotropic Absorber for Use as an Absorbing Boundary Condition," *IEEE Trans. on Antennas and Propagation*, Vol. 43, No. 12, 1995, pp. 1460–1463.
- [3] Teixeira, F., and W. C. Chew, "Systematic Derivation of Anisotropic PML Absorbing Media in Cylindrical and Spherical Coordinates," *IEEE Microwave and Guided Wave Letters*, Vol. 7, No. 11, 1997, pp. 371–373.
- [4] Li, J., and J. Dai, "An Efficient Implementation of the Stretched Coordinate Perfectly Matched Layer," *IEEE Microwave and Wireless Components Letters*, Vol. 17, No. 5, 2007, pp. 322–324.
- [5] Roden, J. A., and S. D. Gedney, "Convolution PML (CPML): An Efficient FDTD Implementation of the CFS–PML for Arbitrary Media," *Microwave and Optical Technology Letters*, Vol. 27, No. 5, 2000, pp. 334–339.
- [6] Antoine, X., E. Lorin, and Q. Tang, "A Friendly Review of Absorbing Boundary Conditions and Perfectly Matched Layers for Classical and Relativistic Quantum Waves Equations," *Molecular Physics*, Vol. 115, No. 15–16, 2017, pp. 1861–1879.
- [7] Mittra, R., *et al.*, "A Review of Absorbing Boundary Conditions for Two and Three-Dimensional Electromagnetic Scattering Problems," *IEEE Trans. on Magnetics*, Vol. 25, No. 4, 1989, pp. 3034–3039.
- [8] Pled, F., and C. Desceliers, "Review and Recent Developments on the Perfectly Matched Layer (PML) Method for the Numerical Modeling and Simulation of Elastic Wave Propagation in Unbounded Domains," *Archives of Computational Methods in Engineering*, 2021, pp. 1–48.

- [9] Shin, W., *3D Finite-Difference Frequency-Domain Method for Plasmonics and Nanophotonics*: Stanford University, 2013.
- [10] Margengo, E., C. M. Rappaport, and E. L. Miller, "Optimum PML ABC Conductivity Profile in FDFD," *IEEE Trans. on Magnetics*, Vol. 35, No. 3, 1999, pp. 1506–1509.
- [11] Cao, J., *et al.*, "Determination of Optimum Conductivity Profile for PML and PML-D Using Multiple-Variables Pade Approximation" *2000 IEEE MTT-S International Microwave Symposium Digest*, Boston, MA, June 11–16, 2000, pp. 1133–1136.
- [12] Chew, W., and J. Jin, "Perfectly Matched Layers in the Discretized Space: An Analysis and Optimization," *Electromagnetics*, Vol. 16, No. 4, pp. 325–340, 1996.
- [13] Collino, F., and P. B. Monk, "Optimizing the Perfectly Matched Layer," *Computer Methods in Applied Mechanics and Engineering*, Vol. 164, No. 1–2, 1998, pp. 157–171.
- [14] Dedek, L., J. Dedkova, and J. Valsa, "Optimization of Perfectly Matched Layer for Laplace's Equation," *IEEE Trans. on Magnetics*, Vol. 38, No. 2, 2002, pp. 501–504.
- [15] Modave, A., "Optimizing the PML in the Discrete Context," 2012.
- [16] Movahhedi, M., *et al.*, "Optimization of the Perfectly Matched Layer for the Finite-Element Time-Domain Method," *IEEE Microwave and Wireless Components Letters*, Vol. 17, No. 1, 2007, pp. 10–12.

FDFD for Calculating Guided Modes

Sometimes exact or approximate analytical solutions exist for waveguides such as rectangular metal waveguides and optical fibers. For waveguides like integrated optical waveguides and photonic crystal waveguides, no analytical solutions exist and numerical solutions are the only option for a designer. When analytical solutions do exist for simple waveguides, usually little can be changed about the waveguide for the solution to still be valid. However, virtually any type of waveguide can be analyzed with numerical techniques. Finite-difference frequency-domain (FDFD) makes calculating guided modes very easy so it is the first implementation to be discussed in this book. Formulation of the method is covered for both rigorous hybrid mode analysis and slab waveguides. Waveguides will be analyzed as an eigenvalue problem so no source will be needed. The effective index method (EIM) is described as an application of slab waveguide analysis to reduce some three-dimensional devices to a two-dimensional model. Last, implementation in MATLAB is discussed and several examples are given for benchmarking codes including a rib waveguide, a slab waveguide, a surface plasmon polariton (SPP), and a microstrip transmission line. The critical concept of convergence is discussed where the resolution of the simulation is increased until numerical error falls below an acceptable threshold.

6.1 Formulation for Rigorous Hybrid Mode Calculation

Let the geometry for analyzing a channel waveguide be as shown in Figure 6.1. The cross section of the channel waveguide is in the xy plane while the guided mode propagates in the z -direction. For hybrid mode analysis, Maxwell's equations will not be separated into two sets of equations. This means that all modes supported by the waveguide will be calculated from the same eigenvalue equation, regardless of how the mode is polarized.

The starting point in FDFD for all waveguide mode calculations is Maxwell's curl equations. The frequency will be known at the start so the free space wavenumber k_0 can be used to normalize the grid coordinates. From Chapter 4, Maxwell's curl equations expanded to

$$\frac{\partial E_z}{\partial y'} - \frac{\partial E_y}{\partial z'} = \mu_{xx} \tilde{H}_x \quad (6.1)$$

$$\frac{\partial E_x}{\partial z'} - \frac{\partial E_z}{\partial x'} = \mu_{yy} \tilde{H}_y \quad (6.2)$$

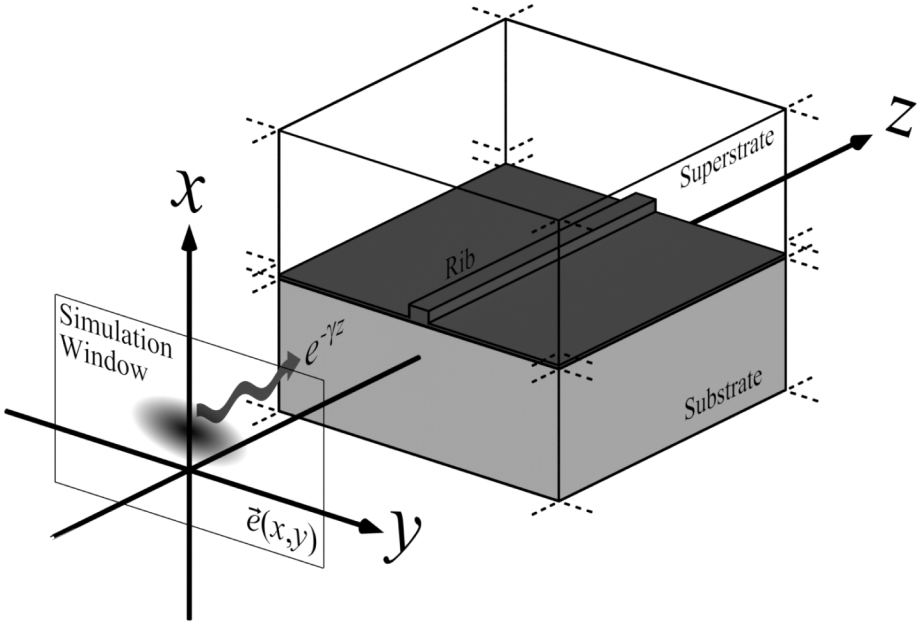


Figure 6.1 Geometry of a channel waveguide.

$$\frac{\partial E_y}{\partial x'} - \frac{\partial E_x}{\partial y'} = \mu_{zz} \tilde{H}_z \quad (6.3)$$

$$\frac{\partial \tilde{H}_z}{\partial y'} - \frac{\partial \tilde{H}_y}{\partial z'} = \epsilon_{xx} E_x \quad (6.4)$$

$$\frac{\partial \tilde{H}_x}{\partial z'} - \frac{\partial \tilde{H}_z}{\partial x'} = \epsilon_{yy} E_y \quad (6.5)$$

$$\frac{\partial \tilde{H}_y}{\partial x'} - \frac{\partial \tilde{H}_x}{\partial y'} = \epsilon_{zz} E_z \quad (6.6)$$

where the magnetic field was normalized according to $\vec{\tilde{H}} = -j\eta_0 \vec{H}$ and the grid coordinates were normalized according to $x' = k_0 x$, $y' = k_0 y$, and $z' = k_0 z$. As discussed in Chapter 2, all guided modes will have the following form, but here the coordinates and the magnetic field are normalized to be consistent with the formulation.

$$\begin{aligned} \vec{\tilde{E}}(x', y', z') &= \vec{e}(x', y') e^{-\gamma z'/k_0} \\ \vec{\tilde{H}}(x', y', z') &= \vec{h}(x', y') e^{-\gamma z'/k_0} \end{aligned} \quad (6.7)$$

Substituting the form of the solution in (6.7) into (6.1) to (6.6) and simplifying gives

$$\frac{\partial e_z}{\partial y'} + \frac{\gamma}{k_0} e_y = \mu_{xx} \tilde{h}_x \quad (6.8)$$

$$-\frac{\gamma}{k_0}e_x - \frac{\partial e_z}{\partial x'} = \mu_{yy}\tilde{h}_y \quad (6.9)$$

$$\frac{\partial e_y}{\partial x'} - \frac{\partial e_x}{\partial y'} = \mu_{zz}\tilde{h}_z \quad (6.10)$$

$$\frac{\partial \tilde{h}_z}{\partial y'} + \frac{\gamma}{k_0}\tilde{h}_y = \epsilon_{xx}e_x \quad (6.11)$$

$$-\frac{\gamma}{k_0}\tilde{h}_x - \frac{\partial \tilde{h}_z}{\partial x'} = \epsilon_{yy}e_y \quad (6.12)$$

$$\frac{\partial \tilde{h}_y}{\partial x'} - \frac{\partial \tilde{h}_x}{\partial y'} = \epsilon_{zz}e_z \quad (6.13)$$

In these equations, the analysis has reduced to just two dimensions because z' was eliminated from the analysis. When the field components are made discrete in the xy plane following the Yee grid scheme, the partial derivatives in each of the above equations can be approximated using finite differences. In addition, observe the ratio γ/k_0 that appears in four of the above equations. Let this ratio define the *normalized complex propagation constant* according to $\tilde{\gamma} = \gamma/k_0$. The resulting set of discrete equations with the normalized propagation constant is

$$\frac{e_z|_{i,j+1} - e_z|_{i,j}}{\Delta y'} + \tilde{\gamma}e_y|_{i,j} = \mu_{xx}|_{i,j}\tilde{h}_x|_{i,j} \quad (6.14)$$

$$-\tilde{\gamma}e_x|_{i,j} - \frac{e_z|_{i+1,j} - e_z|_{i,j}}{\Delta x'} = \mu_{yy}|_{i,j}\tilde{h}_y|_{i,j} \quad (6.15)$$

$$\frac{e_y|_{i+1,j} - e_y|_{i,j}}{\Delta x'} - \frac{e_x|_{i,j+1} - e_x|_{i,j}}{\Delta y'} = \mu_{zz}|_{i,j}\tilde{h}_z|_{i,j} \quad (6.16)$$

$$\frac{\tilde{h}_z|_{i,j} - \tilde{h}_z|_{i,j-1}}{\Delta y'} + \tilde{\gamma}\tilde{h}_y|_{i,j} = \epsilon_{xx}|_{i,j}e_x|_{i,j} \quad (6.17)$$

$$-\tilde{\gamma}\tilde{h}_x|_{i,j} - \frac{\tilde{h}_z|_{i,j} - \tilde{h}_z|_{i-1,j}}{\Delta x'} = \epsilon_{yy}|_{i,j}e_y|_{i,j} \quad (6.18)$$

$$\frac{\tilde{h}_y|_{i,j} - \tilde{h}_y|_{i-1,j}}{\Delta x'} - \frac{\tilde{h}_x|_{i,j} - \tilde{h}_x|_{i,j-1}}{\Delta y'} = \epsilon_{zz}|_{i,j}e_z|_{i,j} \quad (6.19)$$

Each of these discrete equations is written once for every cell in the grid, and each set of equations is written in matrix form following the procedures described in Chapters 3 and 4. These equations are

$$\mathbf{D}_{y'z}^e \mathbf{e}_z + \tilde{\gamma} \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (6.20)$$

$$-\tilde{\gamma} \mathbf{e}_x - \mathbf{D}_{x'z}^e \mathbf{e}_z = \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (6.21)$$

$$\mathbf{D}_{x'y}^e \mathbf{e}_y - \mathbf{D}_{y'x}^e \mathbf{e}_x = \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (6.22)$$

$$\mathbf{D}_{y'}^h \tilde{\mathbf{h}}_z + \tilde{\gamma} \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (6.23)$$

$$-\tilde{\gamma} \tilde{\mathbf{h}}_x - \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (6.24)$$

$$\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y - \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_x = \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (6.25)$$

Next, (6.22) is solved for $\tilde{\mathbf{h}}_z$ and (6.25) is solved for \mathbf{e}_z .

$$\tilde{\mathbf{h}}_z = \boldsymbol{\mu}_{zz}^{-1} \left(\mathbf{D}_{x'y}^e \mathbf{e}_y - \mathbf{D}_{y'x}^e \mathbf{e}_x \right) \quad (6.26)$$

$$\mathbf{e}_z = \boldsymbol{\epsilon}_{zz}^{-1} \left(\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y - \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_x \right) \quad (6.27)$$

It is possible to eliminate $\tilde{\mathbf{h}}_z$ and \mathbf{e}_z from the matrix equations by substituting (6.26) into (6.23) and (6.24) and substituting (6.27) into (6.20) and (6.21). This gives the following set of four coupled matrix equations containing only \mathbf{e}_x , \mathbf{e}_y , $\tilde{\mathbf{h}}_x$, and $\tilde{\mathbf{h}}_y$.

$$\mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \left(\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y - \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_x \right) + \tilde{\gamma} \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (6.28)$$

$$-\tilde{\gamma} \mathbf{e}_x - \mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{zz}^{-1} \left(\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y - \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_x \right) = \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (6.29)$$

$$\mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \left(\mathbf{D}_{x'y}^e \mathbf{e}_y - \mathbf{D}_{y'x}^e \mathbf{e}_x \right) + \tilde{\gamma} \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (6.30)$$

$$-\tilde{\gamma} \tilde{\mathbf{h}}_x - \mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \left(\mathbf{D}_{x'y}^e \mathbf{e}_y - \mathbf{D}_{y'x}^e \mathbf{e}_x \right) = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (6.31)$$

It is very important to realize that the longitudinal components $\tilde{\mathbf{h}}_z$ and \mathbf{e}_z were not set to zero and are not necessarily equal to zero. Instead, they were just algebraically eliminated from the formulation. Next, (6.28) to (6.31) are simplified, terms are rearranged, and the equations are expressed in a different order.

$$\mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_x - \left(\mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h + \boldsymbol{\mu}_{yy} \right) \tilde{\mathbf{h}}_y = \tilde{\gamma} \mathbf{e}_x \quad (6.32)$$

$$\left(\mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h + \boldsymbol{\mu}_{xx} \right) \tilde{\mathbf{h}}_x - \mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y = \tilde{\gamma} \mathbf{e}_y \quad (6.33)$$

$$\mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e \mathbf{e}_x - \left(\mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e + \boldsymbol{\epsilon}_{yy} \right) \mathbf{e}_y = \tilde{\gamma} \tilde{\mathbf{h}}_x \quad (6.34)$$

$$\left(\mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e + \boldsymbol{\epsilon}_{xx} \right) \mathbf{e}_x - \mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e \mathbf{e}_y = \tilde{\gamma} \tilde{\mathbf{h}}_y \quad (6.35)$$

Equations (6.32) and (6.33) can be combined into a single block matrix equation as well as (6.34) and (6.35). These two block matrix equations are

$$\mathbf{P} \begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \end{bmatrix} = \tilde{\gamma} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} \quad (6.36)$$

$$\mathbf{Q} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} = \tilde{\gamma} \begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \end{bmatrix} \quad (6.37)$$

where

$$\mathbf{P} = \begin{bmatrix} \mathbf{D}_x^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h & -(\mathbf{D}_x^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h + \boldsymbol{\mu}_{yy}) \\ \mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h + \boldsymbol{\mu}_{xx} & -\mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h \end{bmatrix} \quad (6.38)$$

$$\mathbf{Q} = \begin{bmatrix} \mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e & -(\mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e + \boldsymbol{\epsilon}_{yy}) \\ \mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e + \boldsymbol{\epsilon}_{xx} & -\mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e \end{bmatrix} \quad (6.39)$$

To derive an eigenvalue problem in terms of just the electric field terms \mathbf{e}_x and \mathbf{e}_y , first (6.37) is solved for the magnetic field terms $\tilde{\mathbf{h}}_x$ and $\tilde{\mathbf{h}}_y$.

$$\begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \end{bmatrix} = \frac{1}{\tilde{\gamma}} \mathbf{Q} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} \quad (6.40)$$

Second, (6.40) is substituted into (6.36) to eliminate the magnetic field terms $\tilde{\mathbf{h}}_x$ and $\tilde{\mathbf{h}}_y$. This gives a matrix wave equation in the form of a standard eigenvalue problem $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$.

$$\boldsymbol{\Omega}^2 \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} = \tilde{\gamma}^2 \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} \quad (6.41)$$

$$\boldsymbol{\Omega}^2 = \mathbf{PQ} \quad (6.42)$$

This general “PQ” form of the eigenvalue problem arises in other methods like the method of lines and rigorous coupled-wave analysis [1]. Solving eigenvalue problems is a huge and involved topic. Fortunately, MATLAB makes this very simple, and (6.41) is solved as simple as $[\mathbf{V}, \mathbf{D}] = \text{eigs}(\text{OMEGASQ})$ where OMEGASQ is a sparse matrix defined in (6.42). When (6.41) is solved as an eigenvalue problem, two matrices are calculated. The eigenvector matrix \mathbf{V} contains the electric field components of the modes along its columns. That is, the m th column of \mathbf{V} is $\vec{e}_m(x, y)$ that contains $e_{x,m}(x, y)$ and $e_{y,m}(x, y)$. The eigenvalue matrix \mathbf{D} contains the eigenvalues $\tilde{\gamma}^2$ along its diagonal. In this case, the eigenvalues are the squares of the normalized complex propagation constants for the guided modes because (6.41) is arranged such that $\tilde{\gamma}^2$

is the eigenvalue. Eigenvectors and eigenvalues always come in pairs. The m th column in the eigenvector matrix must always be kept with the m th eigenvalue. Given the eigenvalue $\tilde{\gamma}_m^2$ of the m th guided mode, the complex propagation constant γ_m , attenuation coefficient α_m , phase constant β_m , and effective refractive index $n_{m,\text{eff}}$ are calculated as follows.

$$\gamma_m = k_0 \sqrt{\tilde{\gamma}_m^2} \quad (6.43)$$

$$\alpha_m = \text{Re}[\gamma_m] \quad (6.44)$$

$$\beta_m = \text{Im}[\gamma_m] \quad (6.45)$$

$$n_{m,\text{eff}} = \frac{\gamma_m}{jk_0} \quad (6.46)$$

If needed, the magnetic field components $\tilde{h}_x(x, y)$ and $\tilde{h}_y(x, y)$ can be calculated from the solution using (6.40). The longitudinal components $\tilde{h}_z(x, y)$ and $e_z(x, y)$ can be calculated using (6.26) and (6.27), respectively.

6.2 Formulation for Rigorous Slab Waveguide Mode Calculation

Let the geometry for analyzing a slab waveguide be as shown in Figure 6.2. Let the cross section of the slab waveguide be in the x -direction while the guided mode propagates in the z -direction. In this configuration, the slab waveguide and the

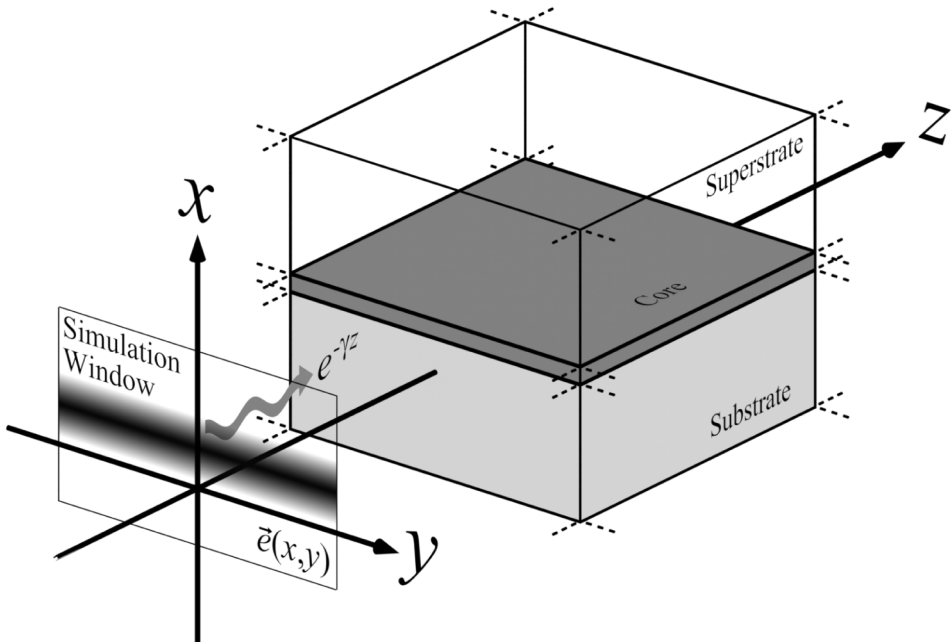


Figure 6.2 Geometry for a slab waveguide.

guided mode itself will be uniform and unchanging in the y -direction. This means that any derivative with respect to y must be equal to zero because nothing changes in the y -direction. Under this condition, $\mathbf{D}_{y'}^e = \mathbf{D}_{y'}^h = \mathbf{0}$ and (6.20) to (6.25) reduce to

$$\tilde{\gamma} \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (6.47)$$

$$-\tilde{\gamma} \mathbf{e}_x - \mathbf{D}_{x'}^e \mathbf{e}_z = \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (6.48)$$

$$\mathbf{D}_{x'}^e \mathbf{e}_y = \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (6.49)$$

$$\tilde{\gamma} \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (6.50)$$

$$-\tilde{\gamma} \tilde{\mathbf{h}}_x - \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (6.51)$$

$$\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (6.52)$$

6.2.1 Formulation of E Mode Slab Waveguide Analysis

Observe that (6.47) to (6.52) have decoupled into two independent sets of three equations. The first set will be called the E mode because its analysis will be reduced to one equation in terms of the single electric field quantity \mathbf{e}_y . The equations for the E mode are given by (6.47), (6.49), and (6.51) and contain only \mathbf{e}_y , $\tilde{\mathbf{h}}_x$, and $\tilde{\mathbf{h}}_z$. The missing components are $\mathbf{e}_x = \mathbf{e}_z = \tilde{\mathbf{h}}_y = 0$. For convenience, these three equations are repeated below.

$$-\tilde{\gamma} \tilde{\mathbf{h}}_x - \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (6.53)$$

$$\tilde{\gamma} \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (6.54)$$

$$\mathbf{D}_{x'}^e \mathbf{e}_y = \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (6.55)$$

The first step to derive the matrix wave equation in the form of an eigenvalue problem is to solve (6.54) for $\tilde{\mathbf{h}}_x$ and solve (6.55) for $\tilde{\mathbf{h}}_z$. This gives

$$\tilde{\mathbf{h}}_x = \tilde{\gamma} \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_y \quad (6.56)$$

$$\tilde{\mathbf{h}}_z = \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e \mathbf{e}_y \quad (6.57)$$

The second step is to substitute the above expressions into (6.53) to eliminate the terms $\tilde{\mathbf{h}}_x$ and $\tilde{\mathbf{h}}_z$ and then rearrange the equation into the form of a generalized eigenvalue problem $\mathbf{A}\mathbf{v} = \lambda\mathbf{B}\mathbf{v}$. This is solved in the same manner as for hybrid mode analysis and is given by

$$-\left(\mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e + \boldsymbol{\epsilon}_{yy}\right) \mathbf{e}_y = \tilde{\gamma}^2 \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_y \quad (6.58)$$

When the permeability is close to vacuum, $\mu_{xx} \approx \mathbf{I}$ and (6.58) reduces to a standard eigenvalue problem $\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$. After the eigenvalue problem is solved, the guided mode parameters are calculated the same as for hybrid mode analysis using (6.43) to (6.46). That is because the eigenvalue problem for slab waveguide analysis has the same term for the eigenvalue.

6.2.2 Formulation of H Mode Slab Waveguide Analysis

The second set of equations that comes from (6.47) to (6.52) will be called the H mode because its analysis will be reduced to one equation in terms of the single magnetic field quantity $\tilde{\mathbf{h}}_y$. The equations for the H mode are given by (6.48), (6.50), and (6.52). The equations contain only $\tilde{\mathbf{h}}_y$, \mathbf{e}_x , and \mathbf{e}_z so for the H mode $\tilde{\mathbf{h}}_x = \tilde{\mathbf{h}}_z = \mathbf{e}_y = 0$. For convenience, these three equations are repeated below.

$$-\tilde{\gamma}\mathbf{e}_x - \mathbf{D}_{x'}^e \mathbf{e}_z = \mu_{yy} \tilde{\mathbf{h}}_y \quad (6.59)$$

$$\tilde{\gamma} \tilde{\mathbf{h}}_y = \epsilon_{xx} \mathbf{e}_x \quad (6.60)$$

$$\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y = \epsilon_{zz} \mathbf{e}_z \quad (6.61)$$

The first step to derive the matrix wave equation in the form of an eigenvalue problem is to solve (6.60) for \mathbf{e}_x and solve (6.61) for \mathbf{e}_z .

$$\mathbf{e}_x = \tilde{\gamma} \epsilon_{xx}^{-1} \tilde{\mathbf{h}}_y \quad (6.62)$$

$$\mathbf{e}_z = \epsilon_{zz}^{-1} \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_y \quad (6.63)$$

The second step is to substitute the above expressions into (6.59) to eliminate the terms \mathbf{e}_x and \mathbf{e}_z and then rearrange the equation into the form of a generalized eigenvalue problem $\mathbf{A}\mathbf{v} = \lambda\mathbf{B}\mathbf{v}$. This is solved in the same manner as for hybrid mode analysis and is given by

$$-\left(\mathbf{D}_{x'}^e \epsilon_{zz}^{-1} \mathbf{D}_{x'}^h + \mu_{yy}\right) \tilde{\mathbf{h}}_y = \tilde{\gamma}^2 \epsilon_{xx}^{-1} \tilde{\mathbf{h}}_y \quad (6.64)$$

Unlike E mode analysis, the \mathbf{B} matrix is permittivity which will likely never be equal to that of a vacuum so (6.64) will rarely reduce to a standard eigenvalue problem. After the eigenvalue problem is solved, the guided mode parameters are calculated the same as for hybrid mode analysis using (6.43) to (6.46). That is because the eigenvalue problem for slab waveguide analysis has the same eigenvalue.

6.2.3 Formulations for Slab Waveguides in Other Orientations

It is very useful to derive the matrix eigenvalue problems where the slab waveguide is oriented along different axes. This is particularly useful when the slab waveguide analysis is being used to calculate sources for waveguide simulations or when the EIM is being used to reduce three-dimensional problems down to two dimensions. For slab modes propagating in the $+x$ -direction in a slab waveguide, the uniform

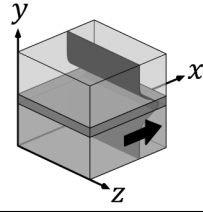
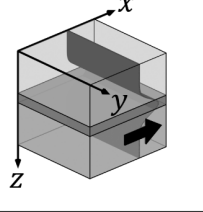
	E Mode $-(\mathbf{D}_{yy}^h \mu_{xx}^{-1} \mathbf{D}_{yy}^e + \epsilon_{zz}) \mathbf{e}_z = \tilde{\gamma}^2 \mu_{yy}^{-1} \mathbf{e}_z$ $\tilde{\mathbf{h}}_x = \mu_{xx}^{-1} \mathbf{D}_{yy}^e \mathbf{e}_z$ $\tilde{\mathbf{h}}_y = \tilde{\gamma} \mu_{yy}^{-1} \mathbf{e}_z$	H Mode $-(\mathbf{D}_{yy}^e \epsilon_{xx}^{-1} \mathbf{D}_{yy}^h + \mu_{zz}) \tilde{\mathbf{h}}_z = \tilde{\gamma}^2 \epsilon_{yy}^{-1} \tilde{\mathbf{h}}_z$ $\mathbf{e}_x = \epsilon_{xx}^{-1} \mathbf{D}_{yy}^h \tilde{\mathbf{h}}_z$ $\mathbf{e}_y = \tilde{\gamma} \epsilon_{yy}^{-1} \tilde{\mathbf{h}}_z$
	E Mode $-(\mathbf{D}_{zz}^h \mu_{xx}^{-1} \mathbf{D}_{zz}^e + \epsilon_{yy}) \mathbf{e}_y = \tilde{\gamma}^2 \mu_{zz}^{-1} \mathbf{e}_y$ $\tilde{\mathbf{h}}_x = -\mu_{xx}^{-1} \mathbf{D}_{zz}^e \mathbf{e}_y$ $\tilde{\mathbf{h}}_z = -\tilde{\gamma} \mu_{zz}^{-1} \mathbf{e}_y$	H Mode $-(\mathbf{D}_{zz}^e \epsilon_{xx}^{-1} \mathbf{D}_{zz}^h + \mu_{yy}) \tilde{\mathbf{h}}_y = \tilde{\gamma}^2 \epsilon_{zz}^{-1} \tilde{\mathbf{h}}_y$ $\mathbf{e}_x = -\epsilon_{xx}^{-1} \mathbf{D}_{zz}^h \tilde{\mathbf{h}}_y$ $\mathbf{e}_z = -\tilde{\gamma} \epsilon_{zz}^{-1} \tilde{\mathbf{h}}_y$

Figure 6.3 Slab waveguide analysis for modes propagating in the +x-direction.

direction can be either in the y-direction or the z-direction. Each choice leads to a different formulation for both the E and H modes and these are summarized in Figure 6.3.

For slab modes propagating in the +y-direction in a slab waveguide, the uniform direction can be either in the x-direction or the z-direction. The formulations for both the E and H modes for this case are summarized in Figure 6.4.

For slab modes propagating in the +z-direction in a slab waveguide, the uniform direction can be either in the x-direction or the y-direction. The formulations for both the E and H modes for this case are summarized in Figure 6.5.

6.2.4 The Effective Index Method

Many times, large and complicated three-dimensional simulations can be reduced to much simpler and more numerically efficient two-dimensional simulations quite accurately. An excellent example is the optical integrated circuit (OIC) for a ring

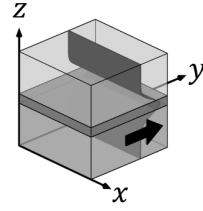
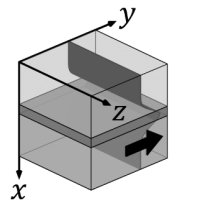
	E Mode $-(\mathbf{D}_{zz}^h \mu_{yy}^{-1} \mathbf{D}_{zz}^e + \epsilon_{xx}) \mathbf{e}_x = \tilde{\gamma}^2 \mu_{zz}^{-1} \mathbf{e}_x$ $\tilde{\mathbf{h}}_y = \mu_{yy}^{-1} \mathbf{D}_{zz}^e \mathbf{e}_x$ $\tilde{\mathbf{h}}_z = \tilde{\gamma} \mu_{zz}^{-1} \mathbf{e}_x$	H Mode $-(\mathbf{D}_{zz}^e \epsilon_{yy}^{-1} \mathbf{D}_{zz}^h + \mu_{xx}) \tilde{\mathbf{h}}_x = \tilde{\gamma}^2 \epsilon_{zz}^{-1} \tilde{\mathbf{h}}_x$ $\mathbf{e}_y = \epsilon_{yy}^{-1} \mathbf{D}_{zz}^h \tilde{\mathbf{h}}_x$ $\mathbf{e}_z = \tilde{\gamma} \epsilon_{zz}^{-1} \tilde{\mathbf{h}}_x$
	E Mode $-(\mathbf{D}_{xx}^h \mu_{yy}^{-1} \mathbf{D}_{xx}^e + \epsilon_{zz}) \mathbf{e}_z = \tilde{\gamma}^2 \mu_{xx}^{-1} \mathbf{e}_z$ $\tilde{\mathbf{h}}_x = -\tilde{\gamma} \mu_{xx}^{-1} \mathbf{e}_z$ $\tilde{\mathbf{h}}_y = -\mu_{yy}^{-1} \mathbf{D}_{xx}^e \mathbf{e}_z$	H Mode $-(\mathbf{D}_{xx}^e \epsilon_{yy}^{-1} \mathbf{D}_{xx}^h + \mu_{zz}) \tilde{\mathbf{h}}_z = \tilde{\gamma}^2 \epsilon_{xx}^{-1} \tilde{\mathbf{h}}_z$ $\mathbf{e}_x = -\tilde{\gamma} \epsilon_{xx}^{-1} \tilde{\mathbf{h}}_z$ $\mathbf{e}_y = -\epsilon_{yy}^{-1} \mathbf{D}_{xx}^h \tilde{\mathbf{h}}_z$

Figure 6.4 Slab waveguide analysis for modes propagating in the +y-direction.

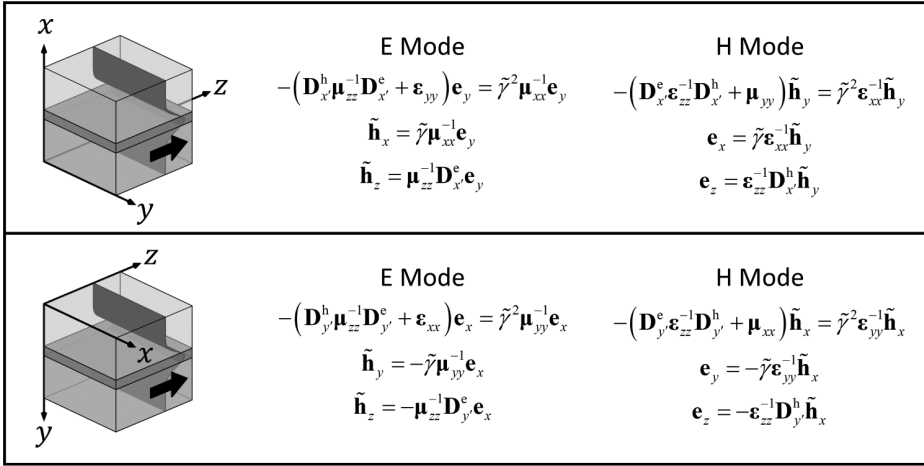


Figure 6.5 Slab waveguide analysis for modes propagating in the +z-direction.

resonator depicted in Figure 6.6. This OIC leads to a very large and three-dimensional simulation that would be very computationally intensive to perform rigorously. Instead, the OIC can be reduced to a two-dimensional representation using the EIM [2]. To do this, the circuit is interpreted as being composed of two regions. The first region is away from the rib waveguide and is composed of the substrate, a thin high-index film on the surface, and the air above. The second region comes from on the rib waveguide and is composed of the substrate, the thin high-index film, a high-index rib on top of the film, and the air above. Both regions are analyzed as a slab waveguide to determine the effective refractive index of the fundamental mode. Recall from Chapter 2 that the fundamental mode is the guided mode with the lowest cutoff frequency. The effective refractive index of the off-waveguide region is n_{eff1} and the effective refractive index of the on-waveguide region is n_{eff2} . These analyses and the resulting fundamental mode calculated for each region are illustrated on the right of Figure 6.6.

Given the two effective refractive indices, a model of the circuit can be constructed in two dimensions as illustrated at the bottom left of Figure 6.6. This two-dimensional representation can be thought of as sort of a top view of the three-dimensional OIC and is constructed from the two effective refractive indices. While this two-dimensional model is not a rigorous representation of the three-dimensional OIC, it is orders of magnitude less computationally intensive while maintaining very good accuracy [2]. In some cases, the regions chosen may not support a guided mode or not even be a slab waveguide at all. In this case, any reasonable estimate of the average refractive index can still provide good results that match well with a three-dimensional simulation.

For best accuracy, it is important to carefully consider whether E or H mode analysis should be performed to ensure the correct polarization is used throughout the entire analysis. For example, the OIC depicted in Figure 6.6 shows the rib waveguide is to be illuminated with a vertically polarized mode. To be consistent, the slab waveguide analysis should be H mode to allow the electric field to be in the z-direction. The E mode would place the electric field solely in the x-direction, so the

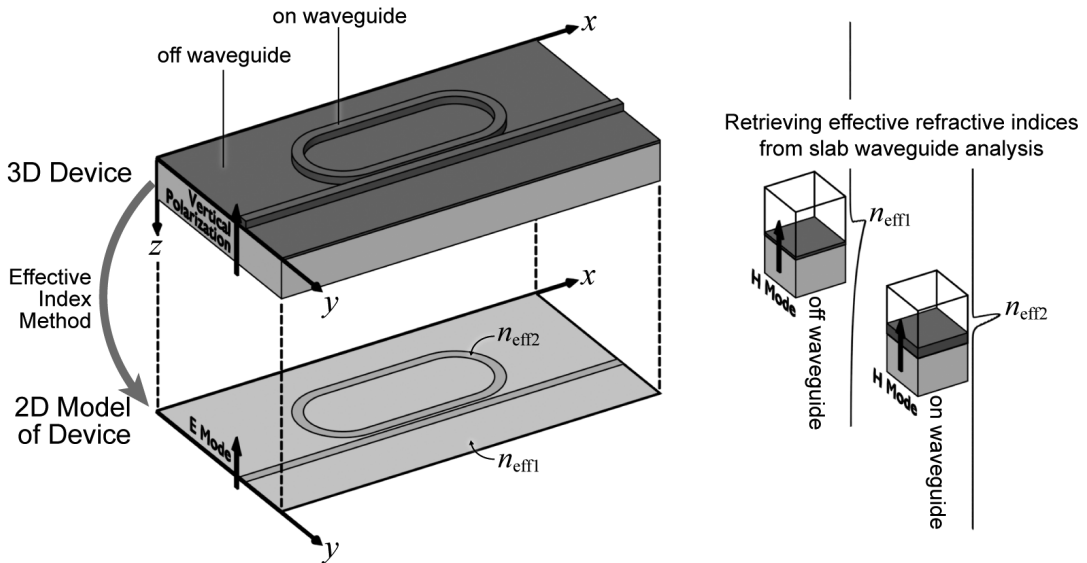


Figure 6.6 EIM used to reduce a three-dimensional OIC to a two-dimensional representation.

E mode is not a correct choice. The two-dimensional simulation of the OIC should be E mode to place the electric field perpendicular to the plane of the OIC. Following a similar line of reasoning, if the rib waveguide mode were horizontally polarized, the slab waveguide analyses would need to be E mode while the two-dimensional simulation of the OIC would need to be H mode. Be careful when applying the EIM because identifying the correct polarizations can be tricky!

6.3 Implementation of Waveguide Mode Calculations

Calculating guided modes using FDFD consists of three major steps, as illustrated in Figure 6.7. Step 1 calculates everything that is needed for the FDFD analysis and starts by initializing MATLAB. It is followed by the dashboard where all of the parameters are defined that control the simulation. The program then moves on to calculating the grid, which includes the number of cells and the resolution. With the grid calculated, the waveguide is built onto the grid producing the tensor arrays ER_{xx} , ER_{yy} , ER_{zz} , UR_{xx} , UR_{yy} , and UR_{zz} used to build the eigenvalue problem in the FDFD method.

With the simulation setup, the code goes on to perform the actual FDFD analysis. Everything in this section of code works toward solving the eigenvalue problem. It starts by reshaping the materials arrays into diagonal matrices and then building the derivative matrices. From here, the eigenvalue problem is constructed. For hybrid mode analysis, this entails calculating \mathbf{P} , \mathbf{Q} , and then $\mathbf{\Omega}^2$ using (6.38), (6.39), and (6.42), respectively. For slab waveguide analysis, this entails calculating the \mathbf{A} and \mathbf{B} matrices in (6.58) for the E mode or (6.64) for the H mode. When the eigenvalue problem is constructed, it is solved using the MATLAB function `eigs()` to calculate the eigenvectors and eigenvalues, referred to collectively as the eigenmodes. It is

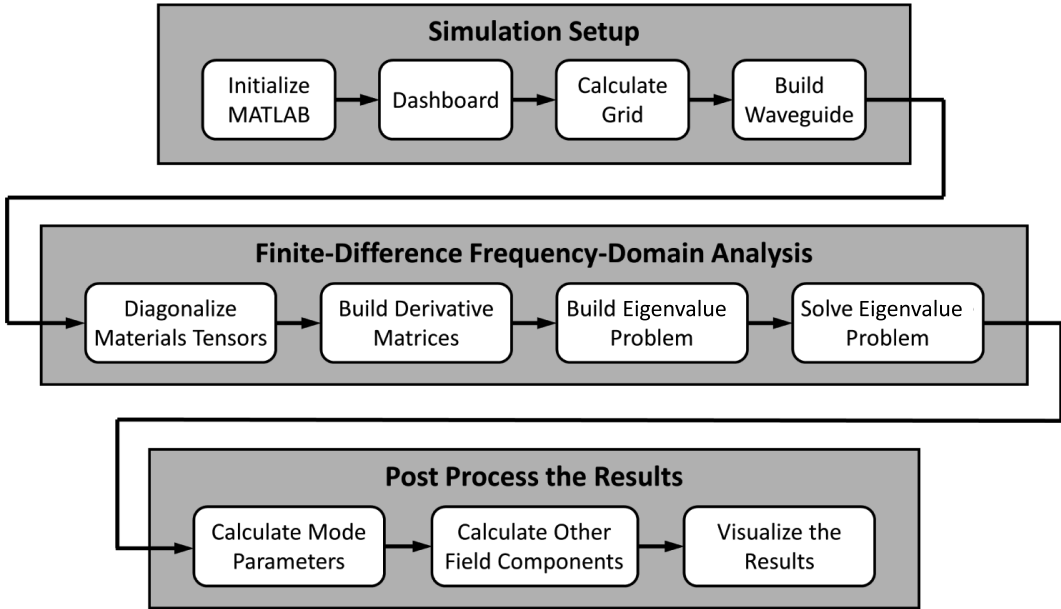


Figure 6.7 Block diagram of FDFD analysis of waveguides.

less efficient and usually unnecessary to calculate all possible eigenmodes. Instead, it is best to calculate only a small set of eigenmodes. If a good guess can be made for the eigenvalue $\tilde{\gamma}_0^2$ of the fundamental mode, it is possible to calculate only a small set of eigenmodes with eigenvalues closest to $\tilde{\gamma}_0^2$. The small set will be the lowest-order eigenmodes. For dielectric waveguides, the majority of power in the fundamental mode typically resides in the core. For this reason, a good guess for the eigenvalue of the fundamental mode is $\tilde{\gamma}_0^2 = -n_{\text{core}}^2$, where n_{core} is the refractive index of the core. For other waveguides, some experimentation may be needed to determine a good guess for the eigenvalue of the fundamental mode.

Last, the eigenmodes are analyzed and postprocessed to extract meaning from the analysis. Oftentimes, this starts by calculating the various mode parameters like the complex propagation constant γ , attenuation coefficient α , phase constant β , and/or the effective refractive index n_{eff} . Each guided mode will have its own set of parameters and they can be calculated from the eigenvalue $\tilde{\gamma}_m^2$ using (6.43) to (6.46). The field terms are extracted from the columns of the eigenvector matrix. If needed, the other field components can be calculated. A typical guided mode calculation will end by visualizing the fields of the guided modes.

6.3.1 MATLAB Implementation of Rib Waveguide Analysis

An excellent application for rigorous hybrid mode analysis for waveguides is that of a silicon-on-insulator (SoI) rib waveguide that is commonly used in integrated optics [3–6]. The waveguide has an inhomogeneous dielectric so it does not strictly support TE and TM modes. The geometry and coordinate setup for a typical SoI rib waveguide is shown in Figure 6.8. Let propagation be in the z -direction and the cross section of the waveguide fall in the xy plane. The figure shows the waveguide,

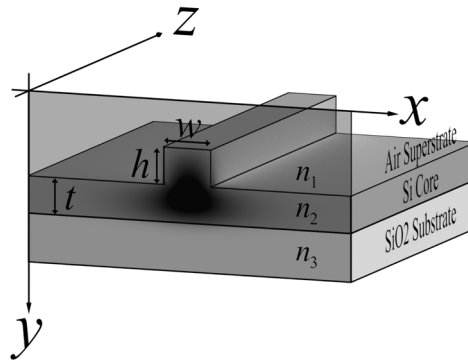


Figure 6.8 Geometry and coordinate setup for Sol rib waveguide. The dimensions are $w = 0.8 \mu\text{m}$, $h = 0.6 \mu\text{m}$, and $t = 0.6 \mu\text{m}$. The refractive indices are $n_1 = 1.0$, $n_2 = 3.5$, and $n_3 = 1.5$. The waveguide is analyzed at free space wavelength $\lambda_0 = 1.55 \mu\text{m}$.

the coordinate axes, dimensions, refractive indices, and the amplitude profile of the fundamental mode supported by the rib waveguide.

The analysis will be performed at the free space wavelength $\lambda_0 = 1.55 \mu\text{m}$, as this is a standard wavelength for telecommunications [7]. The region above the waveguide is called the superstrate and is air in this example. Air will be assigned a refractive index of $n_1 = 1.0$. The middle layer and rib are made of silicon (Si) which has a refractive index close to $n_2 = 3.5$. The region below the waveguide is called the substrate and is silicon dioxide (SiO_2) in this example. The substrate will be assigned a refractive index of $n_3 = 1.5$. The width of the rib is $w = 0.8 \mu\text{m}$, the height of the rib is $h = 0.6 \mu\text{m}$, and the thickness of the silicon layer away from the rib is $t = 0.6 \mu\text{m}$.

The MATLAB code to analyze this rib waveguide can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter6_ribwaveguide.m`. The code begins with the header from lines 1 to 11. The first line is the name of the MATLAB file. In this case, the file is saved as “Chapter6_ribwaveguide.m.” The first task performed is initializing MATLAB. This consists of `close all` to close any figure windows that may happen to be open, `clc` to clear the command window, and the most important `clear all` that clears all variables from memory. Next, the units used in the simulation are defined. For photonic waveguide analysis, `micrometers` is set to 1. Any other units, such as `nanometers`, should be defined relative to `micrometers`.

The next section of the code from lines 12 to 33 is the dashboard. The dashboard contains all of the parameters that define and control the simulation. Parameters to be defined inside of the dashboard include the free space wavelength, waveguide parameters such as dimensions and refractive indices, and grid parameters that control the size and resolution of the grid.

The FDFD analysis here calculates the set of modes that the waveguide can support so the analysis does not require a source. However, the free space wavelength `lam0` for the analysis must still be defined. The parameter `lam0` is the first thing defined in this dashboard on line 17. The second set of parameters defines everything that is needed about the waveguide in order to calculate the modes it supports. This includes the refractive index of the superstrate `rib_n1`, refractive index of the core `rib_n2`, refractive index of the substrate `rib_n3`, height of the rib

`rib_h`, thickness of the silicon layer away from the rib `rib_t`, and width of the rib `rib_w`. These parameters are as consistent as possible with those shown in Figure 6.8. The reason for the `rib_` prefix is to avoid using common variable names that may accidentally get overwritten with different information later in the code. The third set of parameters defines everything that is needed to calculate the grid. The first parameter `nmax` is the largest refractive index that will be assigned to any cells on the grid. This is needed in order to determine the smallest wavelength the analysis will be required to resolve, $\lambda_{\min} = \lambda_0/n_{\max}$. The second parameter is `NRES`. This is the number of points that will be used to resolve the shortest wavelength λ_{\min} . Higher values will improve accuracy by resolving the problem with more points, but the calculations will require more memory and take more time to run. Values in the range of 10 to 40 are typical, but it is necessary to test for convergence to be sure that sufficient resolution is being used. Last, it is necessary to put some space around the rib waveguide so that the guided mode decays to zero before reaching the boundary. This allows simple Dirichlet boundary conditions to be used. Periodic boundary conditions cannot be used since the guided mode is not periodic. `SPACER` is an array containing four numbers that define how much space there will be from the edge of the waveguide to the four grid boundaries. The first number in `SPACER` is the amount of space on the left of the waveguide, the second number is the amount of space on the right, the third number is the amount of space above, and the fourth number is the amount of space below. In the code, the air superstrate will occupy the entire spacer region above the waveguide. The substrate with refractive index n_3 will occupy the entire spacer region below the waveguide. The last parameter defined in the dashboard is the number of modes to calculate `NMODES`. If the grid has 200 by 100 points, 20,000 modes will be calculated unless specified otherwise. Calculation time is greatly improved when only a subset is calculated. In this case, three modes will be calculated. If the number of supported modes is to be determined, `NMODES` will need to be increased until the additional modes are clearly not guided modes. Identifying which modes are guided and which are not will be discussed near the end of this section.

The next section of code from lines 35 to 66 calculates the grid for the waveguide analysis. This includes the number of cells on the grid, `Nx` and `Ny`, as well as the grid resolution parameters, `dx` and `dy`. The first thing in this section is to calculate preliminary values for the grid resolution parameters. The parameters `dx` and `dy` will be adjusted in the second step. Lines 40 and 41 set the preliminary value for both `dx` and `dy` equal to the minimum wavelength λ_{\min} divided by `NRES`. This calculation does not consider the dimensions of the waveguide at all so it is highly likely that the dimensions will not match the grid perfectly. In fact, at this point in the code, the width of the rib is 36.13 cells wide. Since it is not possible to fill in a fraction of a cell, the simulation will not accurately represent the width of the rib. Lines 43 to 47 adjust `dx` and `dy` so that the most critical dimensions of the waveguide are represented exactly by an integer number of cells on the grid. This practice greatly improves the convergence rate of the simulation so it will be possible to achieve high accuracy with a minimum number of cells on the grid. To do this, a critical dimension is chosen for each axis. For the x -direction, the critical dimension is chosen to be `rib_w` because it was the only dimension to choose. Given

the critical dimension, the number of cells representing that dimension is calculated and rounded up to the nearest integer according to $n_x = \text{ceil}(\text{rib_w}/dx)$. The resolution parameter is then adjusted by recalculating it as the critical dimension divided by the number of cells just calculated according to $dx = \text{rib_w}/n_x$. When done correctly, rib_w will be exactly equal to $n_x * dx$. For the y -direction, it is not clear whether rib_t or rib_h is the critical dimension so rib_t was chosen. The number of cells representing this dimension is calculated and rounded up to the nearest integer according to $n_y = \text{ceil}(\text{rib_t}/dy)$. The resolution parameter is then adjusted by recalculating it as the critical dimension divided by the number of cells just calculated according to $dy = \text{rib_t}/n_y$. When done correctly, rib_t will exactly equal $n_y * dy$. Some experimentation with the analysis can give clues about what may be the most important dimension to resolve exactly. After this step, both dx and dy will usually be slightly smaller than they were initially calculated. Next, the size of the grid is calculated on lines 49 to 56. First, the physical width of the grid S_x is calculated on line 50 as the leftmost SPACER region plus the width of the rib plus the rightmost SPACER region. From this, the numerical size of the grid, or the number of grid cells wide, is calculated on line 51 by dividing the physical width by the cell size and rounding up to the nearest integer. Due to the rounding operation, the physical size of the grid may be slightly off so it is recalculated on line 52 as the number of cells wide multiplied by the cell size. The same actions are performed for the vertical size of the grid from lines 54 to 56, but the size of the waveguide is the height of the rib plus the thickness of the silicon layer. At this point, the grid is calculated. If needed, the $2\times$ grid parameters are calculated followed by the axis vectors to be used for graphics and visualization.

The next section of code from lines 68 to 98 builds the rib waveguide onto the Yee grid. This is done by building the rib waveguide onto the $2\times$ grid and then extracting the permittivity and permeability tensor arrays on the Yee grid from the $2\times$ grid. Before anything is added to the $2\times$ grid, the permittivity and permeability arrays ER2 and UR2 on the $2\times$ grid are initialized to air on lines 73 and 74. Observe that the refractive index is squared because it is relative permittivity that must be assigned to points on the grid, not the refractive index. The two are related through $\epsilon_r = n^2$. Forgetting to square the refractive index is one of the most common mistakes made in photonic FDTD simulations.

The second task is to calculate the array indices of where the structures of the waveguide begin and end on the $2\times$ grid. This is implemented on lines 76 to 84. The overall grid strategy for this is illustrated in Figure 6.9 showing the spacer regions around the rib waveguide, the refractive indices, and the array indices. It is highly recommended to draw a figure like this before building anything onto a grid. Observe that n_{x1} and n_{x2} are the array indices for where the rib part of the waveguide begins and ends in the x -direction. The array index n_{x1} is calculated as one plus the number the cells wide of the leftmost spacer region. The $+1$ is included so that the left side of the device is outside of the left spacer region. There is no need to get n_{x1} exact to anything. What is important is that n_{x2} is located correctly relative to n_{x1} . The array index n_{x2} is calculated as n_{x1} plus the number of cells wide for the rib $\text{round}(\text{rib_w}/dx)$ rounded to the nearest integer minus one. As described in Chapter 1, the minus one is required so that the dimension is not one cell greater

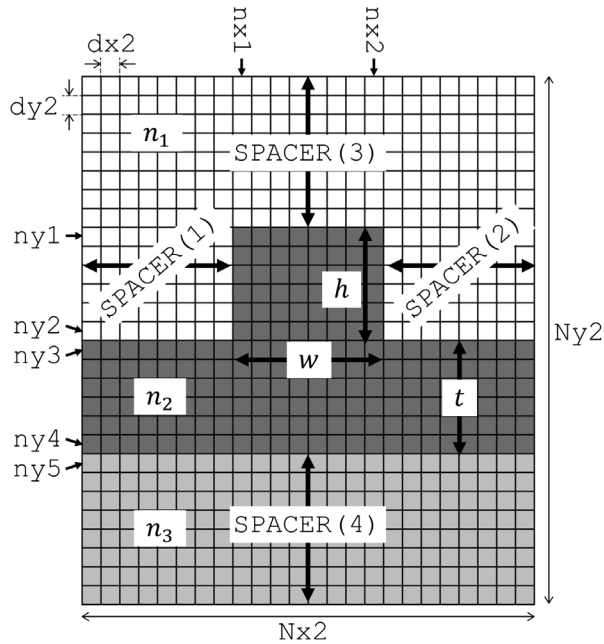


Figure 6.9 Grid strategy for modeling a rib waveguide on the $2\times$ grid.

than intended. While one cell may seem small and insignificant, being one cell off can greatly affect the convergence rate and accuracy of some devices. Similarly, the vertical array indices $ny1$, $ny2$, $ny3$, $ny4$, and $ny5$ are calculated.

From here, it is an easy task to assign permittivity values to the $2\times$ grid. The rib is added first on line 87, followed by the silicon layer on line 88, followed by the substrate on line 89. There is no need to assign values for the superstrate because the array $ER2$ was initialized with the relative permittivity of the superstrate. Observe on line 89 the array is filled from $ny5$ to $Ny2$ in the y -direction because $Ny2$ is the array index for the bottom of the grid. The last task from lines 91 to 98 is to extract the tensors arrays on the Yee grid from the $2\times$ arrays. $ERxx$, $ERyy$, and $ERzz$ are extracted from $ER2$ while $URxx$, $URyy$, and $URzz$ are extracted from $UR2$.

With the grid setup and waveguide constructed onto the Yee grid, it is time to perform the FDFD analysis of the waveguide. This section of code from lines 100 to 141 is generic, and the same code can be used to calculate the hybrid modes of any waveguide. Before the matrices for the eigenvalue problem can be calculated, the tensor arrays must be converted to diagonal matrices. For $ERxx$, this is done by reshaping the array to a column vector via $ERxx(:)$, declaring it as a sparse matrix via $\text{sparse}(ERxx(:))$, and then inserting it as a diagonal into a square matrix via $\text{diag}(\text{sparse}(ERxx(:)))$. This last step creates a sparse matrix by default. It is very important not to reverse the order of the $\text{sparse}()$ and $\text{diag}()$ commands as this will temporarily create a full matrix that will consume a lot of memory and take a long time to process if not crash the computer. The same operation is done for all the material tensor arrays to convert them to diagonal matrices. In this code, the

same variable names are used so the two-dimensional arrays are overwritten by the diagonal matrices. Next, the derivative matrices are constructed on lines 112 to 117 by calling the function `yeeDer2d()` described in Chapter 4. `NS` is an array containing the size of the grid N_x and N_y , `RES` is an array containing the grid resolution parameters dx and dy , and `BC` is an array containing numbers that define the boundary conditions to be used. For waveguide analysis, it is common to use Dirichlet boundary conditions so `BC` is set to `[0 0]`. The formulation used normalized grid coordinates so the input argument `RES` is multiplied by the free space wavenumber k_0 to pass normalized grid resolution terms to `yeeDer2d()`.

Given the diagonal materials matrices and the derivative matrices, the matrices for the eigenvalue problem are calculated on lines 119 to 123. This is done by first calculating the intermediate matrices P and Q and then calculating the matrix for the eigenvalue problem as $P*Q$. Rather than waste memory storing the $P*Q$ matrix, the product is used as the input argument to `eigs()` on line 127. The function `eigs()` is built into MATLAB to solve eigenvalue problems of sparse matrices. There are many different ways this function can be used and many different options for using it. For this example, three input arguments are passed to the function. The first input argument is the matrix $P*Q$. The second input argument is the number of modes to calculate from the eigenvalue problem. The parameter `NMODES` was defined in the dashboard to be 3, so three modes will be calculated that have eigenvalues closest to the value given as the third input argument. Thus, the third input argument needs to be set to the best guess for the eigenvalues of the guided modes. Recall from Chapter 2 for dielectric waveguides that the effective refractive index is close to the average refractive index calculated over the area of the guided mode. Given that the modes reside mostly inside of the silicon material, n_2 is a decent estimate of the effective refractive index of the guided modes. However, the eigenvalues are not the effective refractive index. From (6.43) and (6.46), the eigenvalue that would correspond to an effective index of n_2 is

$$\tilde{\gamma}^2 \approx -n_2^2 \quad (6.65)$$

The function `eigs()` will then return two matrices that are called `Exy` and `D2` in the MATLAB code. If there are M total cells on the grid, the eigenvector matrix `Exy` will have $2M$ rows and `NMODES` number of columns. Each column in `Exy` contains the \mathbf{e}_x and \mathbf{e}_y column vectors from (6.41). The matrix `D2` will be `NMODES`×`NMODES` and contain the eigenvalues along its center diagonal. Eigenvectors and eigenvalues come in pairs, so the m th column in `Exy` corresponds to the eigenvalue in the m th diagonal position of `D2`. For three modes, these matrices have the form below.

$$\mathbf{E}_{xy} = \left[\begin{bmatrix} \mathbf{e}_{x,1} \\ \mathbf{e}_{y,1} \end{bmatrix} \begin{bmatrix} \mathbf{e}_{x,2} \\ \mathbf{e}_{y,2} \end{bmatrix} \begin{bmatrix} \mathbf{e}_{x,3} \\ \mathbf{e}_{y,3} \end{bmatrix} \right] \quad \mathbf{D}^2 = \begin{bmatrix} \tilde{\gamma}_1^2 & 0 & 0 \\ 0 & \tilde{\gamma}_2^2 & 0 \\ 0 & 0 & \tilde{\gamma}_3^2 \end{bmatrix} \quad (6.66)$$

The square of the normalized complex propagation constant has no meaning. For this reason, it is a common practice to first calculate the square root of the

eigenvalue matrix to calculate the normalized complex propagation constants of the modes. This happens on line 128.

$$\mathbf{D} = \sqrt{\mathbf{D}^2} = \begin{bmatrix} \tilde{\gamma}_1 & 0 & 0 \\ 0 & \tilde{\gamma}_2 & 0 \\ 0 & 0 & \tilde{\gamma}_3 \end{bmatrix} \quad (6.67)$$

For photonics applications, it is usually the effective refractive index of the guided modes that is of interest. These are readily calculated from the normalized complex propagation constants simply by multiplying them by $-j$. At the same time, the effective refractive indices are stored in a one-dimensional array `NEFF` instead of a diagonal matrix by extracting the diagonal from the eigenvalue matrix and then multiplying by $-j$. This happens on line 128.

At this point, the finite-difference analysis of the rib waveguide is complete. Any remaining steps are considered postprocessing. Solving the eigenvalue problem only calculates the electric field components \mathbf{e}_x and \mathbf{e}_y , so a common next step is to calculate the other field components \mathbf{e}_z , $\tilde{\mathbf{h}}_x$, $\tilde{\mathbf{h}}_y$, and $\tilde{\mathbf{h}}_z$. The MATLAB code that does this extends from lines 131 to 141. In this code, `M` is the total number of points on the grid and is needed in order to extract the x and y field components from the eigenvectors. Immediately after calculating `M`, the x and y components of the electric field are extracted from the eigenvector matrix `Exy` on lines 133 and 134. Afterward, `Ex` and `Ey` will be rectangular matrices because they are essentially the top and bottom halves of the square matrix `Exy`. Next, the eigenvector matrix for the magnetic fields `Hxy` is calculated using (6.40). This matrix contains the x and y components of the magnetic fields. These are extracted from `Hxy` as `Hx` and `Hy` and are also rectangular matrices like `Ex` and `Ey`. Last, the z components of the fields are calculated using (6.26) and (6.27).

Another common thing to be done after finite-difference analysis is visualizing the fields. The MATLAB code to visualize the x and y components of the electric fields extends from lines 143 to 173. To make the figure compact, the other field components are not visualized. In most cases, the z component of the electric field is small and almost all information about the mode profile is contained in just x and y components. The code is easily modified to visualize additional field components if that is desired. The visualization code has a loop that iterates through all of the calculated modes and visualizes the x and y components horizontally in the figure window. The first step is to extract the column vectors \mathbf{e}_x and \mathbf{e}_y from columns of the `Ex` and `Ey` matrices, respectively, and reshape them back to the two-dimensional Yee grid. The second step is to normalize the values in the field arrays so that the maximum value is 1. The third step is to plot `Ex` on the left of the row and `Ey` on the right. The effective refractive index of the mode is added to the title of the plot of `Ex`.

While the MATLAB code may be completed at this point, the analysis of the waveguide is not at all complete until a convergence study has been performed. This will be discussed in more detail later, but good convergence was found for `NRES`=30 and spacer regions around $1.0\lambda_0$. Using these settings, the grid size was calculated to be `Nx`=269 and `Ny`=294 and the total calculation time was around 2.3 seconds on a laptop computer running a 2.30 GHz Intel Core i9-9880H.

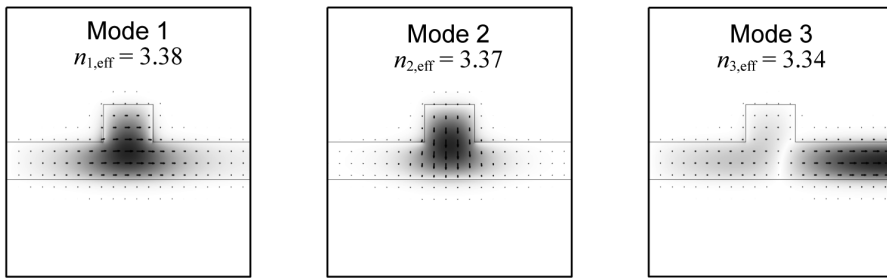


Figure 6.10 First three eigenmodes from the FDFD analysis of a rib waveguide. Gray shading conveys field amplitude and black arrows convey polarization. The first two modes are guided modes because their power is confined to the rib region. The third mode is not a guided mode because it has power increasing away from the rib region.

The first three eigenmodes calculated from this analysis and the effective refractive index for each are shown in Figure 6.10. In this figure, the gray shading conveys the amplitude of the electric field calculated as $\sqrt{E_x.^2 + E_y.^2}$. Simply by inspecting the images of the modes, it is evident that only the first two modes are guided by the rib waveguide. This is evident because the field power of these modes is clearly confined to the vicinity of the rib. The third eigenmode has power increasing away from the rib and reaching a maximum value at the grid boundary. This is not a mode guided by the rib waveguide so it can be concluded that this rib waveguide supports only two modes. It is generally desired to adjust the design of the waveguide so that only a single mode is supported and this finite-difference analysis is a great tool for doing this.

So, what is the meaning of the third eigenmode? Dirichlet boundary conditions were used so it is almost like the rib waveguide is encased inside of a very large metal waveguide. It is not exactly a metal waveguide because the two boundaries where Dirichlet boundary conditions were applied to the magnetic fields make those boundaries a *perfect magnetic conductor* (PMC). At the other two boundaries, Dirichlet boundary conditions were applied to the electric fields so those boundaries have a *perfect electric conductor* (PEC). Only a PEC acts like a true metal. The third eigenmode calculated by this analysis is a mode guided by the larger waveguide, but this does not have any physical meaning related to the rib waveguide and is just ignored.

6.3.2 MATLAB Implementation of Slab Waveguide Analysis

Slab waveguide analysis arises frequently enough in photonics and other areas of electromagnetic analysis that the ability to analyze them will prove to be a powerful capability. The effective refractive index of the guided modes can be used to approximate some three-dimensional structures as two-dimensional structures for easier simulation [2]. Slab waveguides are also elements of devices such as guided-mode resonance filters, grating couplers, leaky wave antennas, and more. The geometry and coordinate setup for the basic dielectric slab waveguide is illustrated in Figure 6.11 and is composed of a substrate, core, and superstrate. Without loss

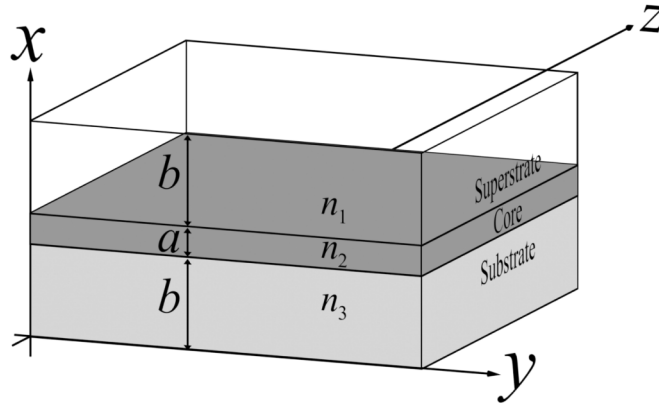


Figure 6.11 Dielectric slab waveguide where $n_2 > n_1$ and $n_2 > n_3$.

of generality, the propagation direction is set to the z -direction and the cross section of the waveguide is placed in the x -direction. The figure shows the waveguide, the coordinate axes, dimensions, and refractive indices. To support guided modes, the core must have a refractive index higher than both the substrate and superstrate. The substrate and superstrate can have different refractive indices, but both must be less than the core refractive index in order to form a waveguide. The thickness a of the core is the only dimension needed to define the slab waveguide. If $n_1 \neq n_3$, there will be some minimum value of a below which the slab waveguide will not support any guided modes. The dimension b is provided to ensure the superstrate and substrate regions are large enough to accurately analyze the waveguide. Dirichlet boundary conditions were used at the extreme top and bottom of the grid. These must be sufficiently far away from the core so that the guided modes decay to zero well before reaching the boundaries of the grid. Typically, $b > 3\lambda_0$. For this analysis, the top spacer region will be the medium above the core filled with refractive index n_1 and the bottom spacer region will be the medium below the core filled with refractive index n_3 .

The analysis will be performed at the free space wavelength $\lambda_0 = 1.55 \mu\text{m}$. The superstrate will be air ($n_1 = 1.0$), the core region will be silicon nitride ($n_2 = 2.0$), and the substrate will be fused silica ($n_3 = 1.5$). The thickness of the slab will be $a = 1500 \text{ nm}$. A correct choice for b will be determined through simulation, but a good guess to start with is three wavelengths $b = 3\lambda_0$. The MATLAB code to analyze this slab waveguide can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter6_slabwaveguide.m`. The program starts with a header that is identical to the header for the rib waveguide discussed previously, except that the first line conveys this code was given a different name. The header initializes MATLAB and defines the units that will be used. For this analysis, `micrometers` was set to 1 because it is a photonics simulation with dimensions closest to this scale.

The next section of the code from lines 12 to 32 is the dashboard. The dashboard contains all of the hard-coded numbers that define and control the simulation. First, the free space wavelength `lam0`, followed by whether it is E or H modes to be analyzed. To calculate E modes, the mode parameter should be set to `MODE = 'E.'` To calculate H modes, the mode parameter should be set to `MODE`

= 'H.' The second set of parameters defines everything that is needed about the slab waveguide. This includes the refractive index of the superstrate n_1 , refractive index of the core n_2 , refractive index of the substrate n_3 , and thickness of the core a . Observe that the parameter b is not defined here because b is not a parameter that defines the slab waveguide. Instead, it is a simulation parameter that must be made sufficiently large so that the grid boundaries are far enough away that the modes guided by the slab are calculated accurately. A sufficient amount of space must be placed between the waveguide and the Dirichlet boundary conditions, just like for the rib waveguide analysis. The third set of parameters defines everything that is needed to calculate the grid. The first parameter n_{\max} is the largest refractive index that will be assigned to any cells on the grid. This is needed in order to determine the smallest wavelength the analysis will be required to resolve, $\lambda_{\min} = \lambda_0/n_{\max}$. The second parameter is N_{RES} . This is the number of points that will be used to resolve the shortest wavelength λ_{\min} . Higher values of N_{RES} will improve accuracy by resolving the problem with more points, but the calculations will take longer and require more memory to run. Values in the range of 10 to 40 are typical, but it is necessary to test for convergence to be sure that sufficient resolution is being used. Last, the parameter b defines the amount of space to include outside of the core to ensure the guided modes decay to zero before reaching the boundary of the grid. This allows simple Dirichlet boundary conditions to be used. The last parameter defined in the dashboard is the number of modes to calculate, N_{MODES} . If the grid has 200 cells, 200 eigenmodes will be calculated unless specified otherwise. It is most efficient to calculate only a subset of the modes because in most simulations the vast majority of eigenmodes are not modes guided by the slab and have no physical meaning related to the slab waveguide. In this case, four modes will be calculated. If the number of supported modes is to be determined, N_{MODES} will need to be increased until the additional modes are clearly not guided modes.

The next section of code from lines 35 to 56 calculates the one-dimensional grid that will be used to represent the slab waveguide. The first thing in this section is to calculate the first guess at the grid resolution parameter dx . This is the first guess because dx is refined in the second step. The grid resolution dx is set equal to the minimum wavelength λ_{\min}/n_{\max} divided by N_{RES} . This calculation does not consider the thickness of the slab waveguide so it is highly unlikely that the dimensions will fit on the grid perfectly. In fact, at this point in the code, the core is 19.35 cells. Since it is not possible to fill in a fraction of a cell, the simulation is not able to accurately represent the thickness of the core. The second step in this section of the code adjusts dx so that the thickness of the slab is represented exactly by an integer number of cells on the grid. Adjusting dx in this manner greatly improves the convergence rate of the simulation so it will be possible to achieve high accuracy with a minimum number of points on the grid. To do this, line 42 calculates the number of cells currently representing the thickness of the core and rounds it up to the nearest integer. The resolution parameter dx is adjusted on line 43 by recalculating it as the slab thickness divided by the number of cells just calculated. After this step, dx will be slightly smaller than it was initially calculated. Next, the size of the grid is calculated on lines 45 and 46. First, the physical size of the grid S_x is calculated as two regions of thickness b and one region of thickness a . From this, line 47 calculates the total number of grid cells by dividing the physical size by the

cell size and rounding up to the nearest integer. Due to the rounding operation, the physical size of the grid S_x may be slightly off so line 48 recalculates it as the number of cells wide times the cell size. If needed, lines 54 to 56 calculate the $2\times$ grid parameters followed by the axis arrays to be used for graphics and visualization. The $2\times$ grid is used here for illustration purposes and consistency, but it is less common to use the technique for slab waveguide analysis.

The next section code from lines 58 to 81 builds the slab waveguide onto the one-dimensional Yee grid. This is done by building the slab waveguide onto the $2\times$ grid and then extracting the permittivity and permeability tensor arrays on the Yee grid from the $2\times$ grid. Before anything is built onto the $2\times$ grid, the permittivity and permeability arrays $ER2$ and $UR2$ on the $2\times$ grid are initialized to air. The second task is to calculate the array indices $nx1$ and $nx2$ of where the core begins and ends on the $2\times$ grid, respectively. The overall grid strategy for this is illustrated in Figure 6.12. The array index $nx1$ is calculated as one plus the number of the cells comprising the superstrate of size b . There is no need to get $nx1$ exact to anything. It is most important to calculate $nx2$ correctly relative to $nx1$. The array index $nx2$ is calculated as $nx1$ plus the number of cells for the slab rounded to the nearest integer minus one. With the calculated array indices, the slab waveguide is built onto the $2\times$ grid on lines 71 to 73. Line 71 adds the superstrate from point 1 up to point $nx1-1$. Line 72 adds the core from point $nx1$ to point $nx2$. Line 73 adds the substrate from point $nx2+1$ all the way down to point $Nx2$. Observe that the refractive index is being squared. That is because it is the relative permittivity that is being assigned to points on the grid where $\epsilon_r = n^2$. The last task is to extract the tensor arrays on the Yee grid from the $2\times$ arrays. ER_{xx} , ER_{yy} , and ER_{zz} are extracted from $ER2$ while UR_{xx} , UR_{yy} , and UR_{zz} are extracted from $UR2$.

With the grid setup and slab waveguide built onto the Yee grid, it is time to perform the FDFD analysis. The section of code from lines 83 to 115 is generic, and the same code can be used to calculate the eigenmodes of any slab waveguide. Before the matrices for the eigenvalue problem can be built, the material tensor arrays must

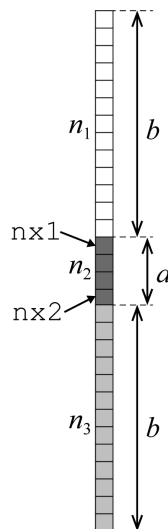


Figure 6.12 Grid strategy for a slab waveguide.

be converted into diagonal matrices using the same procedure followed in the rib waveguide analysis. This happens on lines 88 to 93. Next, lines 96 to 100 construct the derivative matrices by calling the function `yeeder2d()` described in Chapter 4. `NS` is an array containing the size of the grid. To build derivative matrices for the one-dimensional grid used here, the size of the grid in the y -direction is set to 1 by making `NS=[Nx 1]`. The resolution array is set to `RES=[dx 1]`, but the specific numerical value given for dy does not matter since it is not used. The formulation used normalized grid coordinates so the input argument `RES` is multiplied by the free space wavenumber k_0 to pass normalized grid resolution terms to `yeeder2d()`. The boundary condition array is set to `BC=[0 0]`, but the boundary conditions for the y -axis boundaries also do not matter because the y derivative matrices are not used in this analysis.

Given the diagonal materials matrices and the derivative matrices, the matrices for the generalized eigenvalue problem $\mathbf{Ax} = \lambda\mathbf{Bx}$ can be constructed. The matrices \mathbf{A} and \mathbf{B} are constructed differently depending on whether it is the E mode or H mode to be calculated. For the E mode, the matrices are constructed from (6.58). For the H mode, the matrices are constructed from (6.64). Be cautious because it is easy to forget to invert the \mathbf{U}_{Rxx} or \mathbf{E}_{Rxx} matrices or including the negative sign when calculating \mathbf{B} , only to get incorrect eigenmodes. The function `eigs()` is built into MATLAB to solve eigenvalue problems of sparse matrices. For this example, four input arguments are passed to the function. The first two input arguments are the \mathbf{A} and \mathbf{B} matrices. The third input argument `NMODES` was defined in the dashboard and is the number of eigenmodes to be calculated. The parameter `NMODES` was defined in the dashboard to be 4, so four modes will be calculated that have eigenvalues closest to the value given as the fourth input argument to `eigs()`. Thus, the fourth input argument is set to the best guess for the eigenvalues of the guided modes. Given that the modes reside mostly inside of the core region, n_2 is a good estimate of the effective refractive index of the guided modes. However, the eigenvalues are not directly the effective refractive index. From (6.43) and (6.46), the eigenvalue that would correspond to an effective index of n_2 is

$$\tilde{\gamma}^2 \approx -n_2^2 \quad (6.68)$$

From here, the function `eigs()` returns two matrices that are called `Fy` and `D2` in the MATLAB code. If there are N_x points on the grid, the eigenvector matrix `Fy` will have N_x rows and `NMODES` number of columns. Each column in `Fy` contains either the \mathbf{e}_y column vectors for the E mode or the $\tilde{\mathbf{h}}_y$ column vectors for the H mode. The eigenvector matrix was given the symbol \mathbf{F} instead of \mathbf{E} or \mathbf{H} because it could represent either electric or magnetic fields. The matrix `D2` will be of size `NMODES×NMODES` and contain the eigenvalues along its diagonal. When four modes are calculated for the E mode, these matrices have the form below.

$$\mathbf{F}_y = \begin{bmatrix} e_{y,1}(1) \\ e_{y,1}(2) \\ \vdots \\ e_{y,1}(N_x) \end{bmatrix} \begin{bmatrix} e_{y,2}(1) \\ e_{y,2}(2) \\ \vdots \\ e_{y,2}(N_x) \end{bmatrix} \begin{bmatrix} e_{y,3}(1) \\ e_{y,3}(2) \\ \vdots \\ e_{y,3}(N_x) \end{bmatrix} \begin{bmatrix} e_{y,4}(1) \\ e_{y,4}(2) \\ \vdots \\ e_{y,4}(N_x) \end{bmatrix} \quad (6.69)$$

$$\mathbf{D}^2 = \begin{bmatrix} \tilde{\gamma}_1^2 & 0 & 0 & 0 \\ 0 & \tilde{\gamma}_2^2 & 0 & 0 \\ 0 & 0 & \tilde{\gamma}_3^2 & 0 \\ 0 & 0 & 0 & \tilde{\gamma}_4^2 \end{bmatrix} \quad (6.70)$$

The square of the normalized complex propagation constant has no meaning so it is common practice to first calculate the square root of the eigenvalue matrix to calculate the normalized complex propagation constants of the modes.

$$\mathbf{D} = \sqrt{\mathbf{D}^2} = \begin{bmatrix} \tilde{\gamma}_1 & 0 & 0 & 0 \\ 0 & \tilde{\gamma}_2 & 0 & 0 \\ 0 & 0 & \tilde{\gamma}_3 & 0 \\ 0 & 0 & 0 & \tilde{\gamma}_4 \end{bmatrix} \quad (6.71)$$

The effective refractive index of the guided modes is readily calculated from the normalized complex propagation constants simply by multiplying by $-j$. At the same time, the effective refractive indices are stored in a one-dimensional array `NEFF` instead of a diagonal matrix by extracting the diagonal from the eigenvalue matrix and then multiplying by $-j$.

Before the finite-difference analysis of the slab waveguide can be considered complete, a convergence study must be performed. Typically, this is a plot of the effective refractive index of the guided modes as a function of the `NRES` parameter and as a function of the spacer region parameter `b`. Acceptable convergence was found at `NRES=20` and `b=3*lam0`. Any remaining steps after calculating the modes are considered postprocessing. This may include calculating the field components not directly calculated by solving the eigenvalue problem, visualizing the modes, or something else.

The MATLAB code to visualize the fields in the eigenvector matrix extends from lines 117 to 145. The other field components associated with the modes are not calculated or visualized because almost all information about the mode is contained in just \mathbf{e}_y for the E mode and just \mathbf{h}_y for the H mode. First, the figure window is prepared for visualization by clearing it and declaring a “hold on” statement that allows multiple graphic elements to be superimposed. First, the core of the slab waveguide is drawn to the figure window as a rectangle using MATLAB’s `fill()` function. Two arrays `x` and `y` are calculated and passed to this function that contain the vertices working around the perimeter of the rectangle. The rectangle is drawn as a light color of gray as `0.8*[1 1 1]`. After the core is drawn, the eigenmodes are drawn on top of this so that they can be visualized in relation to the geometry of the slab waveguide. To space the modes more easily, the eigenmodes are normalized so that the maximum value is 1. A `for` loop then draws the modes one at a time. The parameter `x0` is calculated to be the horizontal position of where the mode profile will be plotted. Two lines are drawn to visualize the mode. First is a thick white line and the second is a narrower blue line. This trick gives the line a white glow that will allow the mode profile to stand out regardless of what colors are drawn in the background to represent the waveguide. The lines are then labeled with the effective refractive index of the mode using MATLAB’s `text()` function.

The first six eigenmodes calculated from this analysis and the effective refractive indices are shown in Figure 6.13 for the E mode and Figure 6.14 for the H mode. Inspection of these results shows that only three modes are supported by the slab waveguide for both E and H modes. This is obvious for two reasons. First, the field power for the guided modes is clearly confined to the vicinity of the core. The last three eigenmodes have power increasing away from the core so they are not modes guided by the slab. Second, the effective refractive indices of the unguided modes are less than or equal to the substrate refractive index. Guided modes have the majority of their power in the core, leading to effective refractive indices greater than the refractive indices outside of the core. Like with the rib waveguide analysis, the unguided modes are actually modes guided by the larger waveguide that are artificially formed when using Dirichlet boundary conditions at the grid boundaries. These do not have a physical meaning related to the slab waveguide and should be ignored.

6.3.3 Animating the Slab Waveguide Mode

This section demonstrates a fun and educational way to visualize a slab waveguide mode. A GIF animation of a slab waveguide mode propagating through a slab waveguide is generated using the technique described in Chapter 1. Lines 117 to 144 from the original code presented in Section 6.3.2 are replaced with lines 117 to 193 of the alternative code presented in Section 6.3.3. The revised code can be downloaded at <https://empossible.net/fdbook/> and is called `Chapter6_animatedslabmode.m`. Lines 121 to 123 define the name and the number of frames of the GIF animation. Line 126 identifies which mode is to be animated. In this case, the second-order mode is chosen.

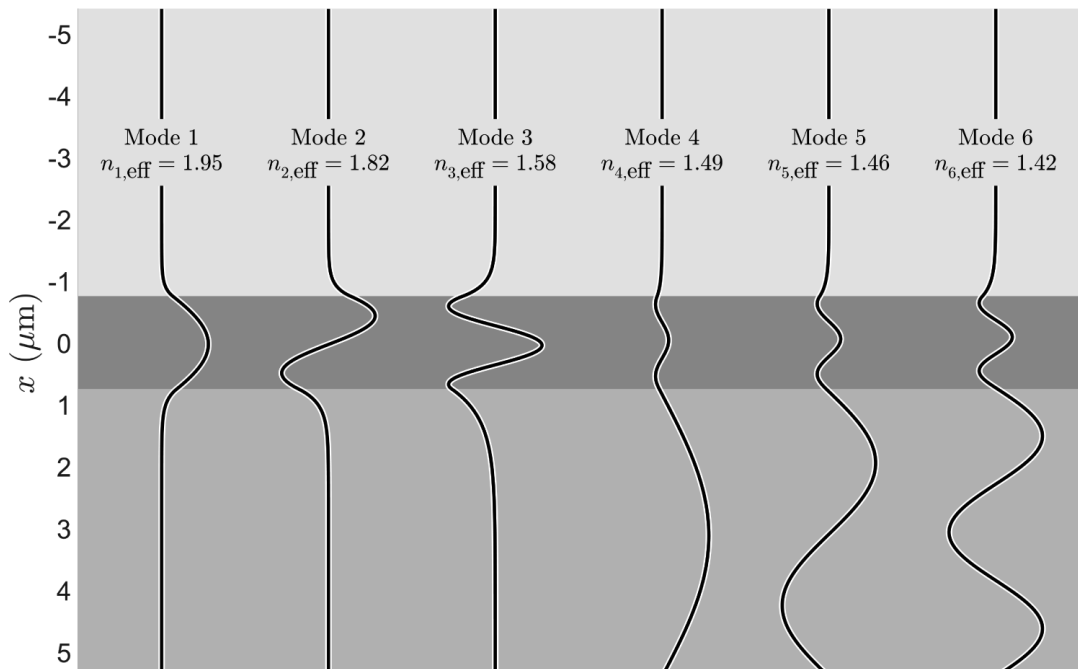


Figure 6.13 First six eigenmodes from E mode analysis of a slab waveguide.

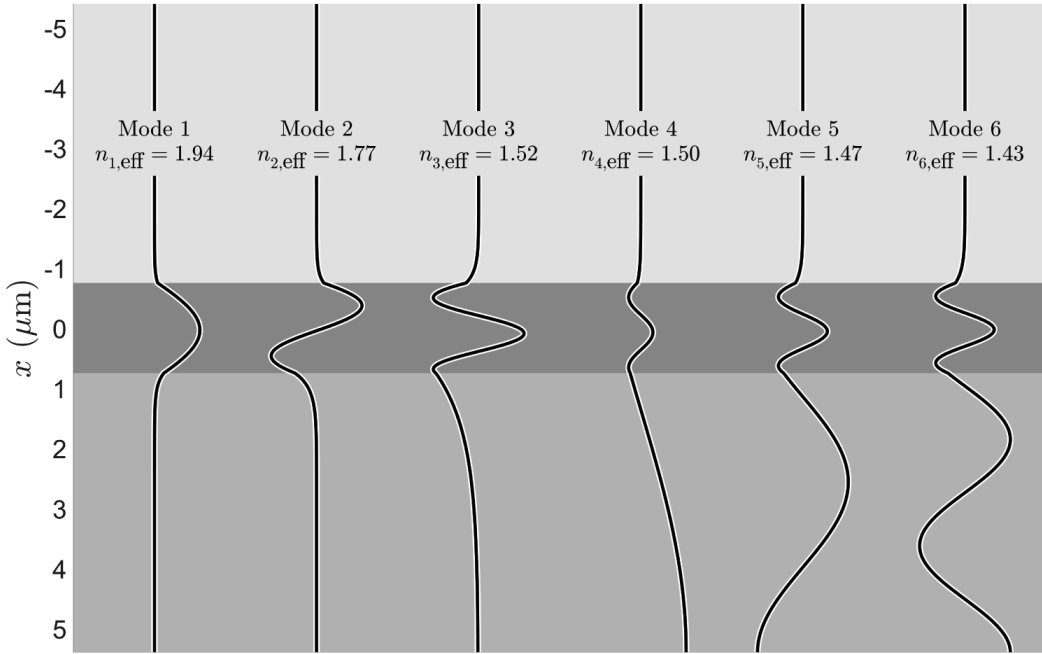


Figure 6.14 First six eigenmodes from H mode analysis of a slab waveguide.

Slab waveguide analysis is a one-dimensional problem, but a two-dimensional set of data is needed in order to calculate the field through a waveguide. Lines 128 to 134 calculate a two-dimensional grid for this purpose. The physical size in the second dimension is made to be twice the size of the cross section of the slab waveguide. With the grid calculated, the mode is calculated across the entire two-dimensional grid on lines 136 to 140. The mode is placed in each cross section of the grid, but the phase is added according to the effective refractive index of that mode using the following

$$E_y(x, y) = e_y(x) \exp(-jk_0 n_{\text{eff}} y) \quad (6.72)$$

After the complex field is calculated, lines 142 to 193 create the GIF animation following Chapter 1. Lines 145 to 147 of the main loop add the phase to the mode. This is the phase that will animate the mode. Lines 149 to 151 clear the figure window and issue the `hold on` command so that the field can be superimposed with the slab waveguide. The field is plotted using the `pcolor()` command on lines 153 to 155. The slab waveguide is drawn in white directly on top of this using the `fill()` command. In order to see the field, the slab waveguide is given a level of transparency. The levels of transparency are calculated on lines 157 to 164 to convey the relative refractive indices of the slab waveguide. The higher the refractive index, the less transparent. The three sections of the slab waveguide are drawn on lines 166 to 172. Lines 174 to 178 finish the figure and force the graphics to draw. Lines 180 to 192 capture a frame and add it to the GIF animation. Line 193 ends the main loop and the visualization is finished.

Figure 6.15 shows a single frame from the animation. The mode moves from left to right. The GIF animation can be inserted into a website or a presentation and

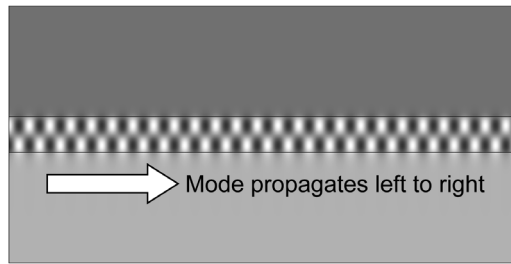


Figure 6.15 Single frame from an animation of the second-order slab waveguide mode.

is a very effective and interesting way to showcase the results of a slab waveguide calculation. Static images are quick and easy to generate, but are boring. You can stand above your competition by showing simple animations like this instead of static images.

6.3.4 Convergence

The most important step in any simulation is to perform a convergence study. To do this, identify all of the parameters in the code that can be adjusted to improve the accuracy and examine how the simulation results change as these parameters are varied. From this data, values for the parameters are identified where the simulation results are acceptable and calculation time is still sufficiently fast. To demonstrate, data will be presented for a convergence study of the rib waveguide covered previously. First, to study grid resolution, the effective refractive index of the fundamental mode was plotted as the grid resolution parameter N_{RES} was increased from 1 to 40. A typical convergence study will exhibit some sort of asymptotic behavior as the result converges to a final value. However, increasing grid resolution also increases simulation time and the memory required to perform the simulation. Both the effective refractive index and simulation time as a function of N_{RES} are shown in Figure 6.16. Convergence seems to begin at around $N_{RES}=10$. This value

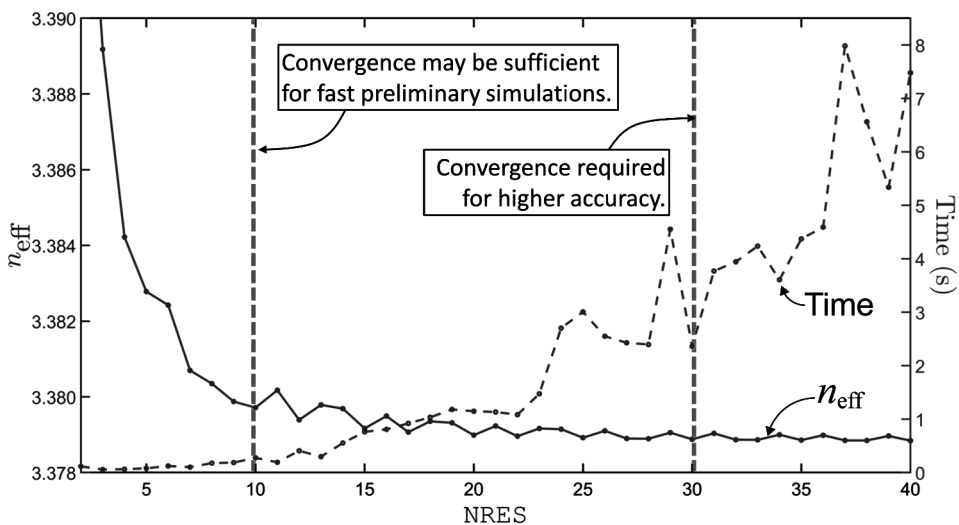


Figure 6.16 Convergence study of grid resolution for rib waveguide analysis.

for NRES can be used when only rough results are needed from the simulation. It is common to perform most of the preliminary simulations or start an optimization with such a choice for NRES. However, as a preliminary design is being refined, it is best to operate farther out on the convergence curve. A value of around $NRES=30$ seems to be a point where minimal improvement in the effective index is achieved while simulation time increases significantly. This would be a good point to obtain a final result for a design.

The grid resolution is not the only parameter to study convergence. There are also the spacer regions to study in order to determine how big they need to be. Figure 6.17 shows the effective refractive index of the fundamental mode and the simulation time to analyze the rib waveguide as a function of b . Preliminary convergence is observed somewhere around $0.4\lambda_0$ to $0.5\lambda_0$ that is sufficient for obtaining fast and rough results. A spacer region approaching $1.0\lambda_0$ is better for higher accuracy.

Convergence studies are a form of parameter sweep where the results of the analysis are plotted as some parameter is varied over a range of values. In this case, it was the NRES and b parameters that were varied. If the FDFD code is written correctly using a dashboard, the convergence study is as easy as wrapping all of the code after the dashboard into a big `for` loop that iterates over all of the different values of NRES or b . Other parameters sweeps are just as easy and the topic is covered in Chapter 9. It is important not to make any conclusions about the results of a simulation until convergence is studied and achieved. Convergence is rarely discussed in the literature and it is always assumed that the results provided are well converged. In commercial software, convergence studies are often done automatically for the user.

6.3.5 MATLAB Implementation for Calculating SPPs

Surface waves are electromagnetic modes confined at the interface of two semi-infinite media [8]. Surface waves are very similar to modes in slab waveguides in

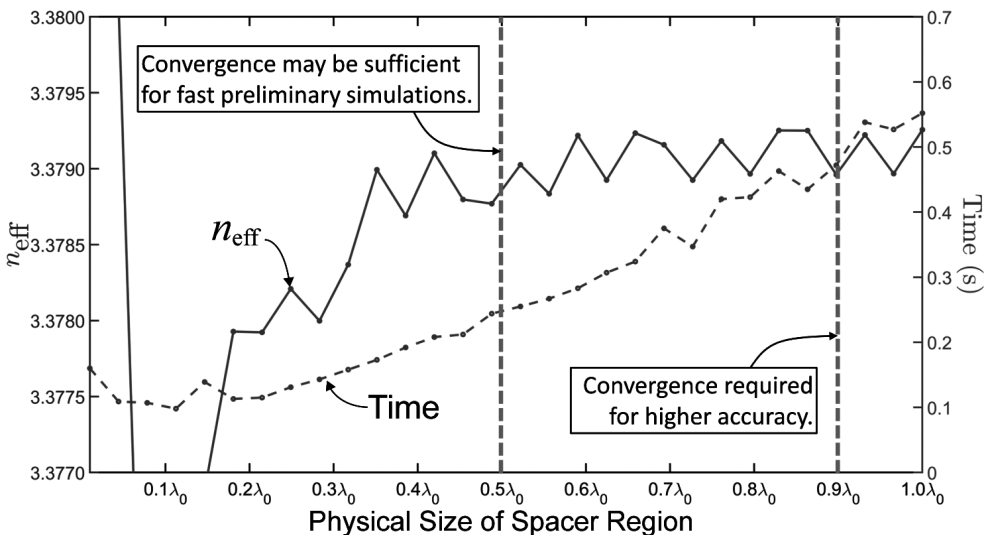


Figure 6.17 Convergence study of spacer region for rib waveguide analysis.

the sense that they are confined to a plane. In the context of FDFD, the numerical approach to calculate surface waves is identical to calculating modes in slab waveguides. Therefore, the MATLAB code described previously can be used here to calculate a surface wave. In the photonics community, a type of surface wave called an SPP is of great interest for subwavelength optics [9, 10] and has found many applications including sensors, waveguides, light sources, solar cells, and much more [11, 12]. An SPP is supported at the interface of a good metal and a dielectric. It involves a coupling between electromagnetic fields in the dielectric with oscillations of electrons in the metal. Figure 6.18 shows the geometry for calculating SPPs at the interface of two materials. These materials can be set to a dielectric and a metal just through proper choice of the relative permittivities $\epsilon_{r,d}$ and $\epsilon_{r,m}$. The dimensions b_1 and b_2 are not part of the definition of the structure supporting the SPP. These are numerical parameters that must be large enough to encompass the entire surface wave so that the top and bottom mediums look semi-infinite to the analysis. Also shown in this figure is a typical SPP confined at the interface. The mode decays exponentially in both directions away from the interface and typically decays more quickly on the metal side.

Boundary conditions require that the electric field tangential to a metal be equal to zero at the interface. The electric field of the E mode in FDFD is entirely tangential to the interface. If the electric field for the E mode is zero, this mode cannot describe SPPs. For this reason, it is only the H mode in FDFD that can describe SPPs. For an interface to support an SPP, the dielectric constant must have an opposite sign on either side of the interface [9]. The most common configuration

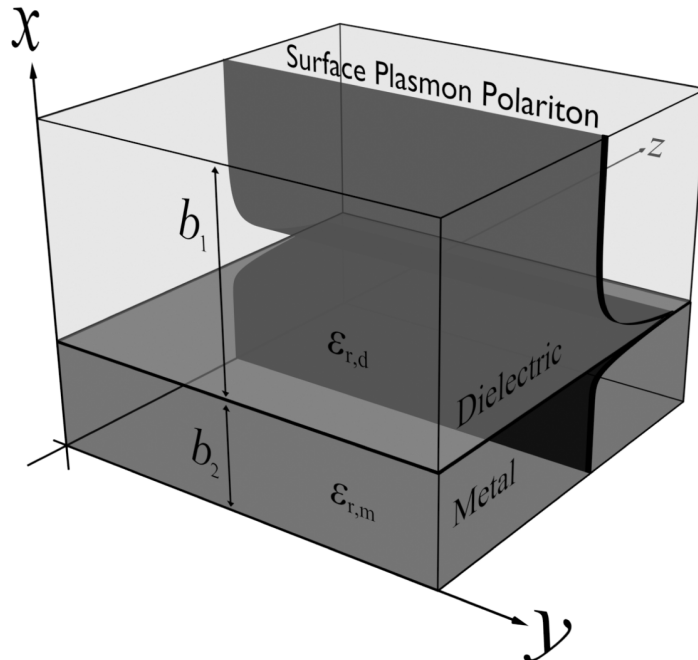


Figure 6.18 Geometry of an SPP at the interface of a metal and dielectric.

that satisfies this is a dielectric–metal interface. Given the relative permittivity of the dielectric $\epsilon_{r,d}$ and the relative permittivity of the metal $\epsilon_{r,m}$, the phase constant β for an SPP is given by [9]

$$\beta = k_0 \sqrt{\frac{\epsilon_{r,d}\epsilon_{r,m}}{\epsilon_{r,d} + \epsilon_{r,m}}} \quad (6.73)$$

Equation (6.73) will be used to estimate the eigenvalue for the FDFD eigenmode analysis. The phase constant β is related to the effective refractive index n_{eff} of the surface wave through $n_{\text{eff}} = \beta/k_0$. Combining this relation with (6.65) and (6.73) gives an expression to calculate the eigenvalue $\tilde{\gamma}^2$ for the surface wave analysis.

$$\tilde{\gamma}^2 \approx -\frac{\epsilon_{r,d}\epsilon_{r,m}}{\epsilon_{r,d} + \epsilon_{r,m}} \quad (6.74)$$

For this example, the dielectric was set to fused silica with $\epsilon_{r,d} = 2.31$ and the metal was set to silver where $\epsilon_{r,m} \approx -9.98 - j0.26$ at a wavelength of $\lambda_0 = 500$ nm. Given these values, the theoretical effective refractive index of the SPP was calculated to be $n_{\text{eff}} = 1.73 - j0.0068$ using (6.73).

The MATLAB code for calculating SPPs can be downloaded at <https://empossible.net/fdfdbook/>. The file is named `Chapter6_spp.m`. The code is essentially the same as that used for slab waveguide analysis. The header extends from lines 1 to 10 and only differs on line 1 where the name of the file is “Chapter6_spp.m.” The dashboard extends from lines 12 to 27. Line 17 defines the free space wavelength to be 500 nm. Lines 19 to 23 define the material properties and the dimensions of the superstrate and substrate. The parameter `erd` defines the relative permittivity of the dielectric on the top and the parameter `erm` defines the relative permittivity of the metal on the bottom. The dimensions of the top and bottom mediums, `b1` and `b2`, must be large enough to fully encompass the surface wave. The results of the simulation were found to converge when the dimensions were set to $b_1 = 2\lambda_0$ and $b_2 = 1\lambda_0$. The grid parameters are defined on lines 25 to 27. The maximum refractive index `nmax` is calculated from the maximum real part of the relative permittivities of the two mediums. The parameter `NRES` controls the grid resolution and is the number of points per wavelength the grid will have. This parameter is set to a value of 200 in the code provided, which is at the very start of where the analysis begins to converge.

Lines 29 to 47 calculate the grid for the problem. Line 34 calculates the grid resolution `dx` as the minimum wavelength $\lambda_{\text{min}}/n_{\text{max}}$ divided by `NRES`. There is no need to snap the grid to any critical dimension because there are no dimensions for this simulation. The dimensions `b1` and `b2` should not affect the properties of the surface wave at all if they are large enough, so there is no need to resolve these with an exact number of grid cells. Lines 36 to 39 calculate the size of the grid. The physical size `Sx` is simply the sum of the sizes of the top and bottom mediums `b1+b2`. Lines 41 to 43 calculate the axis array `xa` that defines the positions of the points along the grid. The value of `b1` is subtracted on line 43 to make `xa=0` at the interface between the two mediums. The $2 \times$ grid parameters are calculated on lines

45 to 47. The $2\times$ grid is useful for this simulation because the H mode has two electric field components at different positions on the grid.

The two mediums are built onto the grid on lines 49 to 68. First, lines 53 to 55 initialize the arrays `ER2` and `UR2` to all ones. Second, lines 57 to 60 incorporate the top and bottom mediums into the array `ER2`. The variable `nx` calculated on line 58 represents the array index for the bottom of the top medium. Line 59 fills in the points on the grid representing the top medium and line 60 fills in the points representing the bottom medium. Third, lines 62 to 65 extract the material tensor arrays `ERxx`, `ERzz`, and `URyy` on the Yee grid from the $2\times$ arrays on the $2\times$ grid. Since it is only the H mode that will ever be of interest in this code, the other material tensor arrays `ERyy`, `URxx`, and `URzz` were not calculated.

The FDFD analysis of the surface wave is performed on lines 67 to 91. First, lines 71 to 74 diagonalize the material tensor arrays `ERxx`, `ERzz`, and `URyy`. Second, lines 76 to 81 build the derivative matrices needed to build the eigenvalue problem. Third, the matrices `A` and `B` for the generalized eigenvalue problem are calculated on lines 83 to 85. These are the same matrices used for slab waveguide analysis. Fourth, lines 87 to 91 solve the generalized eigenvalue problem. Line 88 calculates the target eigenvalue. In this case, the exact eigenvalue is known and is calculated using (6.74). The generalized eigenvalue problem is solved using the built-in function `eigs()` on line 89. Only a single mode is calculated because there are no higher-order modes when dealing with surface waves. The effective refractive index of the surface wave is calculated on lines 90 and 91 exactly how it was for slab waveguide analysis.

At this point in the code, the analysis is finished. Lines 93 to 118 visualize the surface wave by superimposing it onto the top and bottom mediums. The effective refractive index is reported in the title of the figure. Lines 97 to 99 clear the figure window and issue the `hold on` command so the surface wave can be superimposed onto the top and bottom mediums. Lines 101 to 106 draw the top and bottom mediums using the `fill()` command. This is done before plotting the surface wave so that the mediums appear behind the surface wave. Lines 108 to 111 plot the surface wave. Line 109 normalizes the wave so it has unit amplitude. Line 110 draws a wide white line and then line 111 draws a narrower blue line. These two lines together give the blue line a nice white outline that helps it stand out over the colors in the background. Lines 113 to 118 finish the plot and give the plot its title with the effective refractive index.

While the MATLAB program is complete, the simulation is not finished until a convergence study is performed. This is particularly important for SPPs that converge slowly. Figure 6.19 shows a convergence study performed for the calculation of the SPP. The figure shows the real and imaginary parts of the effective refractive index as a function of the grid resolution parameter `NRES`. Preliminary convergence was observed at around `NRES=200`. Very slow convergence to the theoretical values is observed above this. The very fine grid resolution is typical for SPP simulations.

The results produced by the MATLAB code for two different points along the convergence trend are shown in Figure 6.20. Both produce a similar field profile, but the effective refractive index differs the most. The theoretical value for the effective refractive index of this SPP is $n_{\text{eff}} = 1.7335 - j0.006794$. Observe that the field decays most quickly in the metal. This is a typical behavior of SPPs.

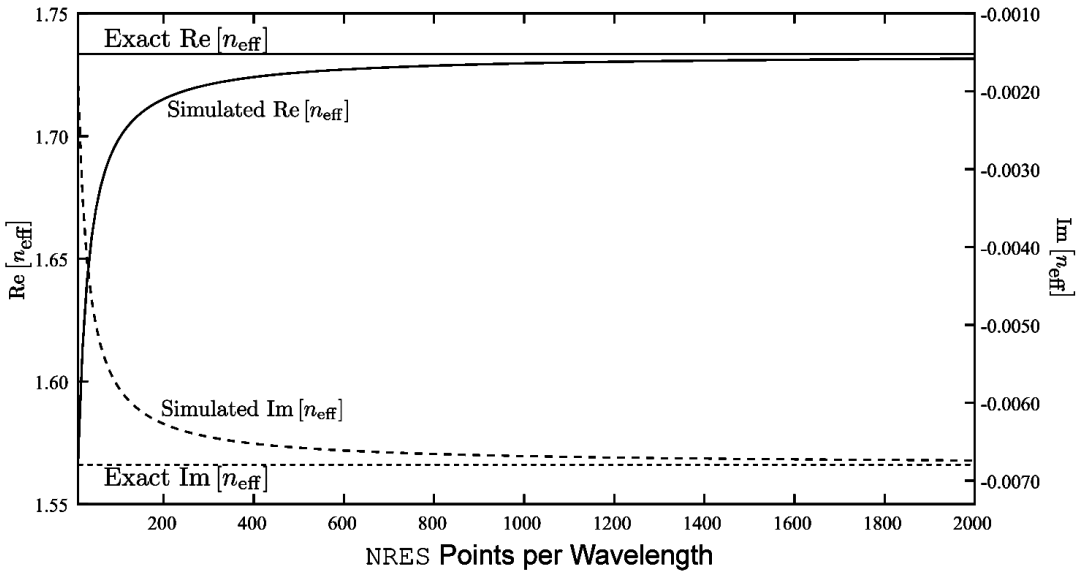


Figure 6.19 Convergence study for SPP calculation.

6.4 Implementation of Transmission Line Analysis

Analysis of transmission lines is nothing more than rigorous hybrid mode analysis followed by some additional postprocessing steps to calculate the transmission line parameters. Very often at radio frequencies, metals are specified solely in terms of

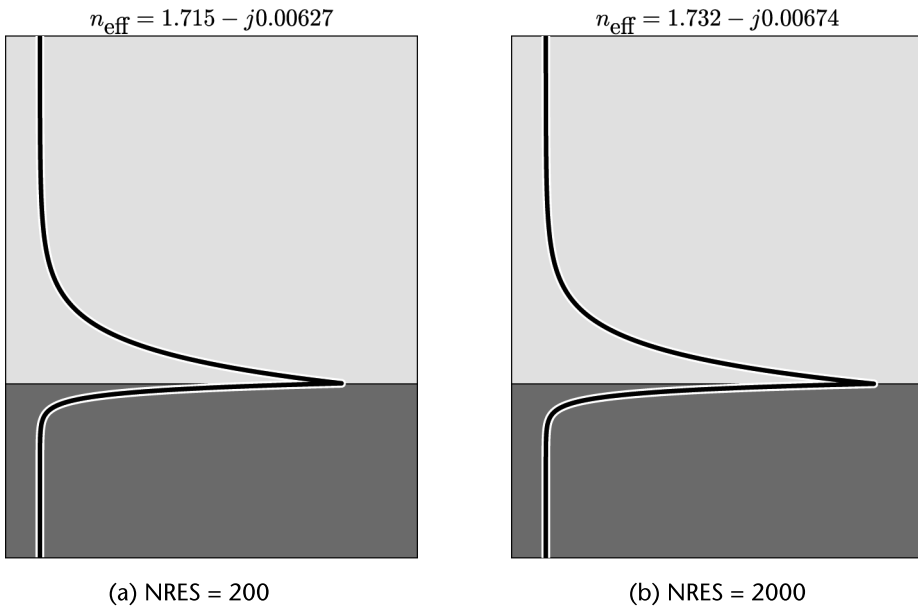


Figure 6.20 Results from SPP calculation for two different values of NRES.

their conductivity σ and dielectrics are specified in terms of their dielectric constant ϵ_r and loss tangent $\tan\delta$. However, the FDFD analysis describes all materials in terms of their complex permittivity and complex permeability. From Chapter 2, the complex relative permittivity $\tilde{\epsilon}_r$ of a metal can be estimated from just the conductivity σ as

$$\tilde{\epsilon}_r \equiv 1 + \frac{\sigma}{j\omega\epsilon_0} \quad (6.75)$$

Furthermore, the complex relative permittivity $\tilde{\epsilon}_r$ of a dielectric can be calculated from the real-valued dielectric constant ϵ_r and the loss tangent $\tan\delta$ according to

$$\tilde{\epsilon}_r \equiv \epsilon_r(1 - j\tan\delta) \quad (6.76)$$

Given the complex relative permittivity for the metal and dielectric, ordinary FDFD analysis of the transmission line can be performed using rigorous hybrid mode analysis.

In order to calculate correct transmission line parameters, the magnetic field components should be denormalized after the eigenmodes are calculated. This can be accomplished in a single step by incorporating the denormalization $\mathbf{h}_i = \tilde{\mathbf{h}}_i/(-j\eta_0)$ into (6.37). This is

$$\begin{bmatrix} \mathbf{h}_x \\ \mathbf{h}_y \end{bmatrix} = \frac{1}{-j\eta_0\tilde{\gamma}} \mathbf{Q} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} \quad (6.77)$$

Given the electric fields in the vicinity of the transmission line, the potential difference V_0 between the two conductors is calculated from a line integral of the electric field \vec{E} from the conductor at higher potential to the conductor at lower potential.

$$V_0 = \int_{L1}^{L2} \vec{E} \cdot d\vec{\ell} \quad (6.78)$$

Given the denormalized magnetic fields in the vicinity of the transmission line, the current I_0 through each of the conductors is calculated from a closed-contour line integral of the magnetic field \vec{H} around one of the conductors.

$$I_0 = \oint_L \vec{H} \cdot d\vec{\ell} \quad (6.79)$$

It follows that the characteristic impedance Z_0 is the voltage V_0 divided by the current I_0 .

$$Z_0 = \frac{V_0}{I_0} \quad (6.80)$$

The complex propagation constant γ is calculated by denormalizing the normalized propagation constant $\tilde{\gamma}$ that was calculated from the eigenvalue.

$$\gamma = k_0 \tilde{\gamma} \quad (6.81)$$

Recall from Chapter 2 that the distributed transmission line parameters R , L , G , and C are related to the characteristic impedance Z_0 and complex propagation constant γ through

$$Z_0 = \sqrt{\frac{R + j\omega L}{G + j\omega C}} \quad (6.82)$$

$$\gamma = \sqrt{(R + j\omega L)(G + j\omega C)} \quad (6.83)$$

Solving (6.82) and (6.83) for R , L , G , and C gives

$$R = \text{Re}[\gamma Z_0] \quad (6.84)$$

$$L = \frac{1}{\omega} \text{Im}[\gamma Z_0] \quad (6.85)$$

$$G = \text{Re}\left[\frac{\gamma}{Z_0}\right] \quad (6.86)$$

$$C = \frac{1}{\omega} \text{Im}\left[\frac{\gamma}{Z_0}\right] \quad (6.87)$$

To demonstrate, the microstrip transmission line illustrated in Figure 6.21 will be simulated at 1.0 GHz and the transmission line parameters R , L , G , and C will be calculated from the results. The microstrip is formed from copper onto an FR-4 substrate. The width of the copper line is 1 mm and its thickness is 35 μm . The thickness of the substrate between the ground plane and the line is 0.5 mm. At 1.0 GHz, copper has a conductivity of $5.8 \times 10^7 \Omega \cdot \text{m}$ and FR-4 has a dielectric constant of 4.4 and a loss tangent of 0.03.

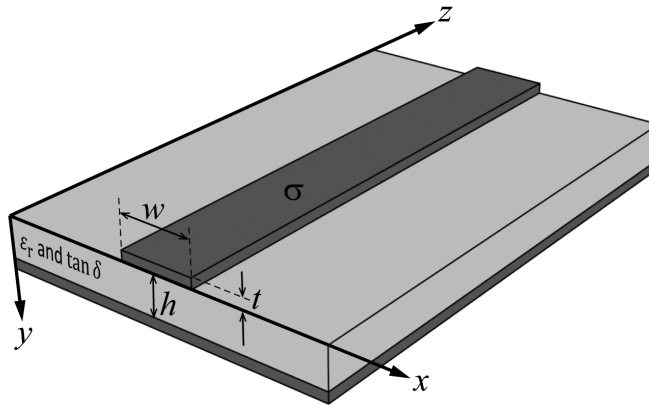


Figure 6.21 Geometry of a microstrip transmission line. For this example, $w = 1$ mm, $h = 0.5$ mm, $t = 35$ μm , $\epsilon_r = 4.4$, $\tan\delta = 0.03$, and $\sigma = 5.8 \times 10^7 \Omega \cdot \text{m}$.

The MATLAB code that analyzes the microstrip depicted in Figure 6.21 can be downloaded at <https://empossible.net/fdfdbook/>. The file is named `Chapter6_microstrip.m`. The majority of the code follows the ordinary hybrid mode analysis covered previously so only key differences in the MATLAB code will be discussed. The dashboard extends from lines 25 to 52. For source parameters, it is only the operating frequency that is defined and the free space wavelength is calculated from it. The material properties are defined from lines 34 to 41 in three parts. The first part defines the conductivity `sigma` of the copper and then calculates the complex dielectric constant `erm` from that. The second part defines the properties of the FR-4 in terms of the dielectric constant `er` and loss tangent `tand`. From these, the complex dielectric constant `erd` is calculated. Last, the dielectric constant `era` of the material above the microstrip is defined as air. The dimensions of the microstrip are defined from lines 43 to 46 where `w` is the width, `h` is the thickness of the dielectric, and `t` is the thickness of the microstrip line. Last, the grid parameters are defined from lines 49 to 52. A new parameter `NDIM` is defined that specifies how many cells the minimum feature size should be resolved with. For the microstrip line as defined, this minimum dimension is width `w` in the x -direction and conductor thickness `t` in the y -direction. For the microstrip, no spacer region is needed below the line due to the presence of the ground plane. For this reason, the spacer region below the line was set to a size of zero.

The grid is calculated on lines 54 to 93. Compared to the rib waveguide analysis, there is an extra step here to ensure the minimum dimensions are resolved by at least `NDIM` number of points. This happens from lines 62 to 73. Building the microstrip onto the Yee grid is performed on lines 95 to 130. Note that the $2\times$ grid was not used here because the rectangular geometry of the microstrip makes it easy to assign permittivity values directly to the tensor arrays `ERxx`, `ERyy`, and `ERzz`.

The finite-difference analysis from lines 132 to 176 is nearly identical to that of the rib waveguide but has two key differences. The first difference is the guess taken for the estimate of the eigenvalue of the lowest order mode. The incorporation of metals into the analysis makes estimating the eigenvalue for the fundamental mode more difficult than it was for dielectric waveguides. Line 159 uses (6.88) to estimate the effective relative permittivity of a microstrip transmission line [13].

$$\epsilon_{r,\text{eff}} \cong \frac{\epsilon_r + 1}{2} - \frac{\epsilon_r - 1}{2\sqrt{1 + 12 h/w}} \quad (6.88)$$

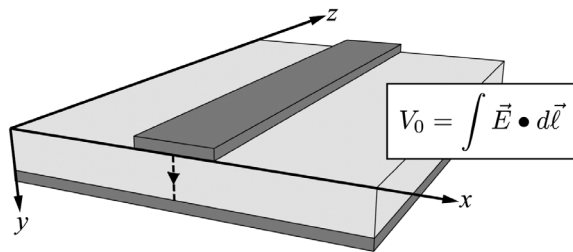
Given the effective permittivity, the estimate for the eigenvalue is

$$\gamma_0^2 \cong -\text{Re}[\epsilon_{r,\text{eff}}] \quad (6.89)$$

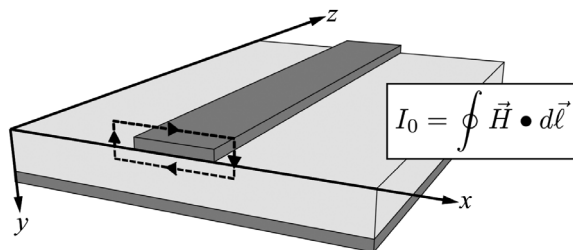
The second difference is that after the eigenvalue problem is solved, the complex propagation constant γ is calculated instead of the effective refractive index. This tends to be the more meaningful parameter for radio frequency circuit analysis [14]. In MATLAB, $\tilde{\gamma}$ is given the name `gamman`. Lines 164 to 166 extract \mathbf{e}_x and \mathbf{e}_y from the eigenvector matrix and calls them `Ex` and `Ey`. Line 168 uses (6.77) to calculate the denormalized magnetic field components \mathbf{h}_x and \mathbf{h}_y . Lines 172 to 176 reshape the field components back to the two-dimensional grid.

The last section of code from lines 178 to 228 postprocesses the fields calculated by the FDFD analysis to calculate the transmission line parameters and to visualize the field. Lines 183 to 184 use the array indices calculated when building the microstrip to perform the line integration of the electric field \vec{E} illustrated in Figure 6.22(a). This step calculates the potential difference V_0 between the ground plane and the microstrip line using (6.78). In this case, the positive direction is downward, consistent with the coordinates chosen for the analysis. The voltage V_0 calculated on line 184 is a complex number because it has an amplitude and a phase. Lines 186 to 192 also use the array indices calculated when building the microstrip to perform a line integration of the magnetic field \vec{H} that completely encircles the top conductor as illustrated in Figure 6.22(b). This step calculates the current I_0 through the line using (6.79) and is also a complex number because it has an amplitude and a phase. The variable s defined on line 187 defines the number of cells away from the microstrip line the path of integration should take. Lines 188 to 191 calculate the line integral above the line, to the right of the line, below the line, and to the left of the line, respectively. Line 192 sums the integrals to get total current I_0 , but negative signs are used where the integrations happen in the opposite direction of the Cartesian coordinates used in the analysis.

At this point, the difficult calculations are finished. Line 195 calculates the characteristic impedance Z_0 of the line directly from the voltage V_0 and current I_0 that was just calculated using (6.80). The complex propagation constant γ is calculated on line 196 from the normalized complex propagation constant γ_{norm} using (6.81). The effective refractive index is then calculated on line 197 from the



(a) Line integral to calculate current V_0



(b) Line integral to calculate current I_0

Figure 6.22 (a) Line integral of the electric field to calculate the potential difference between conductors. (b) Closed-contour line integral of the magnetic field to calculate the current through the line.

Table 6.1 Microstrip Parameters

Parameter	Value
Z_0	$50.5 + j0.60 \, \Omega$
γ	$0.52 + j37.7 \, \text{m}^{-1}$
R	$3.75 \, \Omega/\text{m}$
L	$303 \, \text{H/m}$
G	$19.2 \, \text{n}/\Omega \cdot \text{m}$
C	$119 \, \text{pF/m}$
n_{eff}	1.80
$\epsilon_{r,\text{eff}}$	3.24

imaginary part of γ . Lines 199 to 203 calculate the distributed transmission line parameters R , L , G , and C using (6.84) to (6.87).

The last step in the code displays the results of the FDFD analysis. Lines 205 to 213 display the numerical values for the transmission line parameters in the command window. Lines 216 and 217 calculate the magnitude of the electric field and normalize it to have a maximum value of one. Lines 219 to 230 visualize the electric field magnitude using MATLAB's `pcolor()` function.

After the program is written and runs correctly, it is a big mistake to assume the simulation is finished and the answer it gives is correct. A convergence study must be performed and the program must be benchmarked with known cases. The example given in this section can be used to benchmark your own code. In the dashboard, there are three variables that all affect the accuracy of the analysis. These are `NRES` that controls grid resolution relative to wavelength, `NDIM` that controls grid resolution relative to the minimum dimension, and `SPACER` that controls how much space to place around the device. Acceptable results were found for the values given in the dashboard and the transmission line parameters that come from the analysis are summarized in Table 6.1.

References

- [1] Rumpf, R. C., "Improved Formulation of Scattering Matrices for Semi-Analytical Methods That Is Consistent with Convention," *Progress in Electromagnetics Research*, Vol. 35, 2011, pp. 241–261.
- [2] Qiu, M., "Effective Index Method for Heterostructure-Slab-Waveguide-Based Two-Dimensional Photonic Crystals," *Applied Physics Letters*, Vol. 81, No. 7, 2002, pp. 1163–1165.
- [3] Chan, S. P., *et al.*, "Single-Mode and Polarization-Independent Silicon-On-Insulator Waveguides with Small Cross Section," *Journal of Lightwave Technology*, Vol. 23, No. 6, 2005, p. 2103.
- [4] Lousteau, J., *et al.*, "The Single-Mode Condition for Silicon-on-Insulator Optical Rib Waveguides with Large Cross Section," *Journal of Lightwave Technology*, Vol. 22, No. 8, 2004, pp. 1923–1929.

- [5] Rickman, A., G. Reed, and F. Namavar, "Silicon-on-Insulator Optical Rib Waveguide Loss and Mode Characteristics," *Journal of Lightwave Technology*, Vol. 12, No. 10, 1994, pp. 1771–1776.
- [6] Vivien, L., *et al.*, "Polarization-Independent Single-Mode Rib Waveguides on Silicon-on-Insulator for Telecommunication Wavelengths," *Optics Communications*, Vol. 210, No. 1–2, 2002, pp. 43–49.
- [7] Miller, S., *Optical Fiber Telecommunications*, New York: Elsevier, 2012.
- [8] Polo, J., T. Mackay, and A. Lakhtakia, *Electromagnetic Surface Waves: A Modern Perspective*: Newnes, 2013.
- [9] Pitarke, J., *et al.*, "Theory of Surface Plasmons and Surface-Plasmon Polaritons," *Reports on Progress in Physics*, Vol. 70, No. 1, 2006, p. 1.
- [10] Zayats, A. V., I. I. Smolyaninov, and A. A. Maradudin, "Nano-Optics of Surface Plasmon Polaritons," *Physics Reports*, Vol. 408, No. 3–4, 2005, pp. 131–314.
- [11] Welford, K., "Surface Plasmon-Polaritons and Their Uses," *Optical and Quantum Electronics*, Vol. 23, No. 1, 1991, pp. 1–27.
- [12] Zhang, J., L. Zhang, and W. Xu, "Surface Plasmon Polaritons: Physics and Applications," *Journal of Physics D: Applied Physics*, Vol. 45, No. 11, 2012, p. 113001.
- [13] El-Refaei, H., D. Yevick, and I. Betty, "Stable and Noniterative Bidirectional Beam Propagation Method," *IEEE Photonics Technology Letters*, Vol. 12, No. 4, 2000, pp. 389–391.
- [14] Pozar, D. M., *Microwave Engineering*, Hoboken, NJ: John Wiley & Sons, 2011.

FDFD for Calculating Photonic Bands

This chapter will discuss the formulation and implementation of finite-difference frequency-domain (FDFD) for calculating photonic band diagrams and isofrequency contours (IFCs). Photonic band diagrams are useful for identifying bandgaps and IFCs are useful when analyzing the dispersion of periodic structures. A brief discussion of constructing photonic band diagrams and IFCs will be covered prior to the implementation. Describing the theory of periodic structures and photonic bands is beyond the scope of this book so it will not be covered here. A very good introduction to the topic can be found in [1, 2]. The key concepts to understand are lattice vectors, Bloch wave vectors, unit cells, Brillouin zones (BZs), irreducible BZs (IBZs), and the key points of symmetry.

7.1 Photonic Bands for Rectangular Lattices

A *photonic band diagram* is a fast and simple, yet incomplete, description of the electromagnetic properties of a periodic structure. It is essentially a map of the frequencies of the modes that are allowed to propagate through a periodic structure as a function of the Bloch wave vector $\vec{\beta}$. The construction of a photonic band diagram is illustrated in Figure 7.1. It starts by drawing the BZ, highlighting the IBZ, and identifying the key points of symmetry around the IBZ, as depicted in the top-left of Figure 7.1. From there, a path is chosen around the perimeter of the IBZ that passes through the key points of symmetry. In this case, it was chosen to start at the origin Γ , progress to the key point X , then up to the key point M , and then back to Γ . The bottom-left of Figure 7.1 shows this path unfolded into a straight line. The unfolded path becomes the horizontal axis of the photonic band diagram. Working through this path in small increments, each point corresponds to a different Bloch wave vector. For each Bloch wave vector, the periodic structure is analyzed by solving the wave equation as an eigenvalue problem. The eigenvalues obtained from the solution convey the frequencies of the modes that are supported by the photonic crystal. These frequencies are plotted vertically above the point along with the horizontal axis corresponding to the current Bloch wave vector. If enough points are used along the horizontal axis, the discrete frequencies align to form the photonic bands. In the formulation presented in Section 7.2, k_0^2 is the eigenvalue and is interpreted as frequency squared. It may seem odd to interpret the free space wavenumber k_0 as frequency, but it is related to frequency ω through k_0

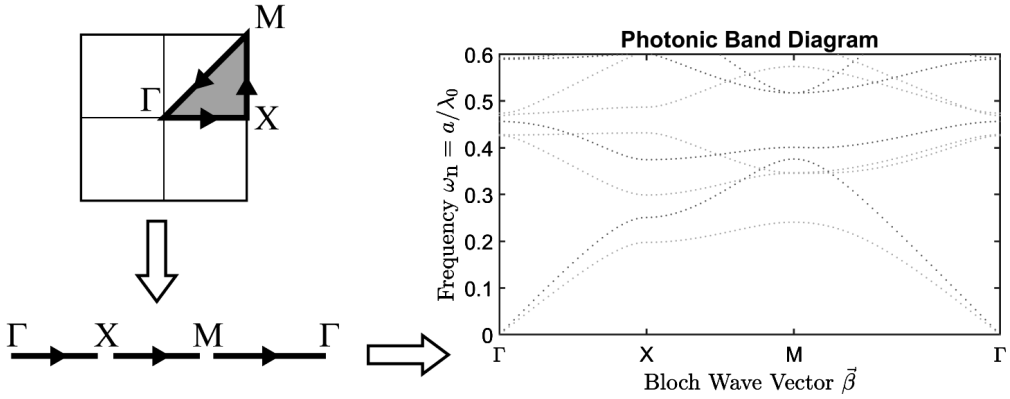


Figure 7.1 Construction of a photonic band diagram.

$= \omega/c_0$. Therefore, k_0 differs from frequency only by the constant c_0 which is the speed of light in a vacuum.

There are two primary ways to solve an eigenvalue problem $\mathbf{A}\mathbf{x} = k_0^2\mathbf{x}$. The fastest approach calculates only the eigenvalues of the eigenmodes, while a slower approach calculates both the eigenvalues and the eigenvectors. Only the eigenvalues are needed for calculating photonic bands. The eigenvectors are needed only if the fields associated with the modes are to be visualized or used in some other way. The allowed electromagnetic modes inside of a periodic structure are called *Bloch modes*. Instead of plotting the frequencies as k_0^2 , they are normalized. The i th normalized frequency $\omega_{n,i}$ is calculated from the i th eigenvalue $k_{0,i}^2$ according to

$$\omega_{n,i} = \frac{a}{2\pi} \sqrt{k_{0,i}^2} = \frac{a}{\lambda_{0,i}} \quad (7.1)$$

The normalized frequency ω_n has the very useful form of lattice spacing a divided by free space wavelength λ_0 . This normalization makes designing photonic crystals from band diagrams very easy. For example, suppose something interesting is observed at a normalized frequency of $\omega_n = 0.7$ and it is desired to design a photonic crystal to operate at a free space wavelength $\lambda_0 = 1300$ nm. The lattice spacing of the photonic crystal should be $a = \omega_n \lambda_0 = (0.7)(1300 \text{ nm}) = 910$ nm. For whatever crazy reason, the vertical axes of photonic band diagrams in the literature never seem to be labeled with a/λ_0 . Instead, they are labeled with either ω_n or even more often as $\omega a/2\pi c_0$. It is the author's opinion that $\omega a/2\pi c_0$ is a poor label because it is unnecessarily confusing and provides little insight into the actual meaning and utility of the normalized frequency.

It should be obvious that analyzing a photonic crystal only around the perimeter of the IBZ is missing information. To get a complete picture of the bands, the photonic crystal must be analyzed throughout the entire area of the IBZ. This data can then be unfolded to recover the data for the full BZ. To simplify this, envision a double for loop iterating over the entire BZ, and for each point simulated within the BZ, the discrete eigenfrequencies are plotted above the BZ. When this is done and the sampling is fine enough, the full photonic bands emerge as illustrated in

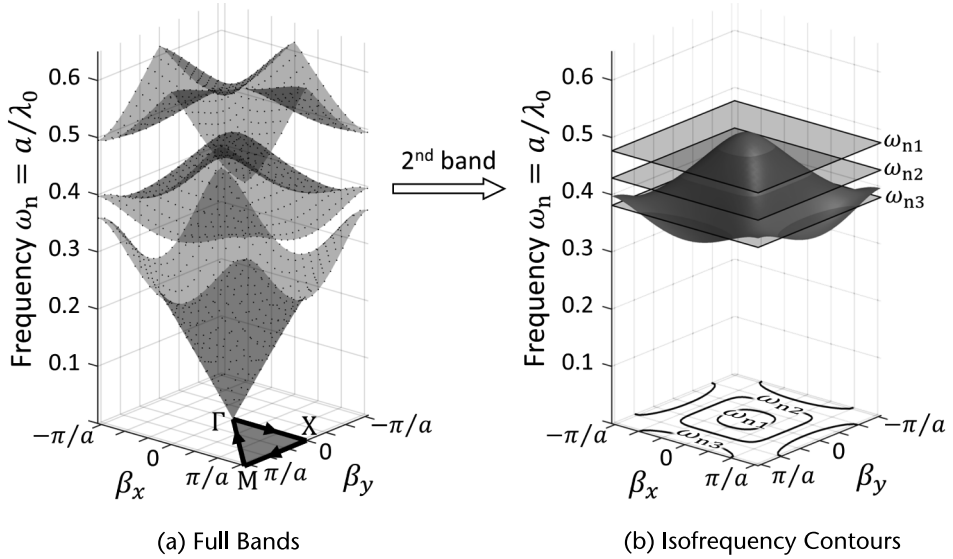


Figure 7.2 Concept of IFCs and relation to the photonic bands. (a) Full bands calculated throughout entire BZ. (b) Three IFCs constructed from the second band.

Figure 7.2(a). To understand how the full bands relate back to the photonic band diagram, the IBZ and key points of symmetry were added to this figure.

Instead of viewing the full photonic band diagram, as shown in Figure 7.2(a), it is more common to plot just the IFCs. Figure 7.2(b) illustrates how the IFCs are constructed from the full bands and how they are typically visualized. For clarity, only the second band is shown. Three planes representing three fixed frequencies slice through the second band. The IFCs are the lines formed where the constant frequency planes intersect the bands. The intersection lines are shown by themselves at the bottom of Figure 7.2(b). The lines at the bottom show how IFCs are typically visualized. If more constant-frequency planes are added, more IFC lines will appear. It is very important to understand how the IFCs relate back to the full bands and how the full bands relate back to the photonic band diagram.

The following sections explain how to calculate the photonic bands using the FDFD method for rectangular lattices. It is shown how to calculate and plot both photonic band diagrams and IFCs. Calculating photonic bands of oblique lattices like hexagonal photonic crystals requires modification to the basic FDFD algorithm [3, 4] that falls outside of the scope of this book. It is possible to choose a rectangular unit cell that perfectly represents a hexagonal lattice when arrayed. Analyzing this lattice, however, does not sufficiently enforce the phase conditions for a hexagonal array. Incorrect bands will be calculated if this is attempted.

7.2 Formulation for Rectangular Lattices

The starting point in FDFD for photonic band calculations is Maxwell's curl equations expanded into Cartesian coordinates. In these equations, the magnetic field

is normalized according to $\tilde{H} = -j\eta_0\vec{H}$. The grid coordinates are not normalized because k_0 will be the output of the algorithm and will not be known until the algorithm finishes. From Chapter 4 for two-dimensional analysis, the matrix equations when frequency is not known were

$$\mathbf{D}_y^e \mathbf{e}_z = k_0 \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (7.2)$$

$$-\mathbf{D}_x^e \mathbf{e}_z = k_0 \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (7.3)$$

$$\mathbf{D}_x^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = k_0 \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (7.4)$$

$$\mathbf{D}_y^h \tilde{\mathbf{h}}_z = k_0 \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (7.5)$$

$$-\mathbf{D}_x^h \tilde{\mathbf{h}}_z = k_0 \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (7.6)$$

$$\mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{D}_y^h \tilde{\mathbf{h}}_x = k_0 \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (7.7)$$

For photonic band calculations, the derivative matrices must incorporate periodic boundary conditions (PBCs) at all boundaries. For the PBCs, the Bloch wave vector $\vec{\beta}$ is used in place of \vec{k}_{inc} for the phase terms. Observe that (7.2) to (7.7) have separated into two independent sets of matrix equations. The first set will be called the E mode (TM) because the set only contains a single electric field term \mathbf{e}_z and the final equation to be solved will have only this term.

$$\mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{D}_y^h \tilde{\mathbf{h}}_x = k_0 \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (7.8)$$

$$\mathbf{D}_y^e \mathbf{e}_z = k_0 \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (7.9)$$

$$-\mathbf{D}_x^e \mathbf{e}_z = k_0 \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (7.10)$$

A matrix wave equation in the form of an eigenvalue problem is derived by solving (7.9) for $\tilde{\mathbf{h}}_x$, solving (7.10) for $\tilde{\mathbf{h}}_y$ and substituting both of these expressions into (7.8) to put the equation solely in terms of the electric field term \mathbf{e}_z . This gives

$$-\left(\mathbf{D}_x^h \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_x^e + \mathbf{D}_y^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_y^e\right) \mathbf{e}_z = k_0^2 \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (7.11)$$

$$\tilde{\mathbf{h}}_x = \frac{1}{k_0} \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_y^e \mathbf{e}_z \quad (7.12)$$

$$\tilde{\mathbf{h}}_y = -\frac{1}{k_0} \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_x^e \mathbf{e}_z \quad (7.13)$$

Equation (7.11) is the eigenvalue problem to be solved to calculate E mode (TM) bands. The eigenvalue is k_0^2 and the eigenvectors are \mathbf{e}_z . Most of the time, it is only

the eigenvalues that are obtained. If it is desired to calculate the field \mathbf{e}_z of the Bloch modes, it becomes necessary to calculate the eigenvectors. Equations (7.12) and (7.13) are only needed if $\tilde{\mathbf{h}}_x$ or $\tilde{\mathbf{h}}_y$ field components are desired to be calculated from \mathbf{e}_z .

The second set of equations from (7.2) to (7.7) will be called the H mode (TE) because the set only contains a single magnetic field term $\tilde{\mathbf{h}}_z$ and the final equation to be solved will have only this term.

$$\mathbf{D}_x^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = k_0 \mu_{zz} \tilde{\mathbf{h}}_z \quad (7.14)$$

$$\mathbf{D}_y^h \tilde{\mathbf{h}}_z = k_0 \epsilon_{xx} \mathbf{e}_x \quad (7.15)$$

$$-\mathbf{D}_x^h \tilde{\mathbf{h}}_z = k_0 \epsilon_{yy} \mathbf{e}_y \quad (7.16)$$

A matrix wave equation in the form of an eigenvalue problem is derived by solving (7.15) for \mathbf{e}_x , solving (7.16) for \mathbf{e}_y and substituting both of these expressions into (7.14) to put the equation solely in terms of the magnetic field term $\tilde{\mathbf{h}}_z$. This gives

$$-(\mathbf{D}_x^e \epsilon_{yy}^{-1} \mathbf{D}_x^h + \mathbf{D}_y^e \epsilon_{xx}^{-1} \mathbf{D}_y^h) \tilde{\mathbf{h}}_z = k_0^2 \mu_{zz} \tilde{\mathbf{h}}_z \quad (7.17)$$

$$\mathbf{e}_x = \frac{1}{k_0} \epsilon_{xx}^{-1} \mathbf{D}_y^h \tilde{\mathbf{h}}_z \quad (7.18)$$

$$\mathbf{e}_y = -\frac{1}{k_0} \epsilon_{yy}^{-1} \mathbf{D}_x^h \tilde{\mathbf{h}}_z \quad (7.19)$$

Equation (7.17) is the eigenvalue problem to be solved to calculate H mode (TE) bands. The eigenvalue is k_0^2 and the eigenvectors are $\tilde{\mathbf{h}}_z$. Equations (7.18) and (7.19) are only needed if \mathbf{e}_x or \mathbf{e}_y field components are desired to be calculated from $\tilde{\mathbf{h}}_z$.

7.3 Implementation of Photonic Band Calculation

Calculating photonic bands using FDFD consists of three major steps, as illustrated in Figure 7.3. Step 1 calculates everything that is needed for the FDFD analysis to happen and starts by initializing MATLAB. It is followed by the dashboard, the section of code where all of the simulation parameters are defined. After the dashboard, the setup moves on to calculating the grid, which includes the number of cells and the resolution. With the grid calculated, a single unit cell of the periodic structure is built onto the grid producing the tensor arrays \mathbf{E}_{Rxx} , \mathbf{E}_{Ryy} , \mathbf{E}_{Rzz} , \mathbf{U}_{Rxx} , \mathbf{U}_{Ryy} , and \mathbf{U}_{Rzz} used by the FDFD method. Ordinary photonic bands apply only to photonic crystals of an infinite extent. This can be modeled in FDFD with just a single unit cell as long as PCBs are used at all boundaries. After converting the material arrays to diagonal matrices, the setup moves on to calculate the list of Bloch wave vectors that will each be simulated separately. For photonic band diagrams, this entails choosing a path around the IBZ and resolving the path with enough

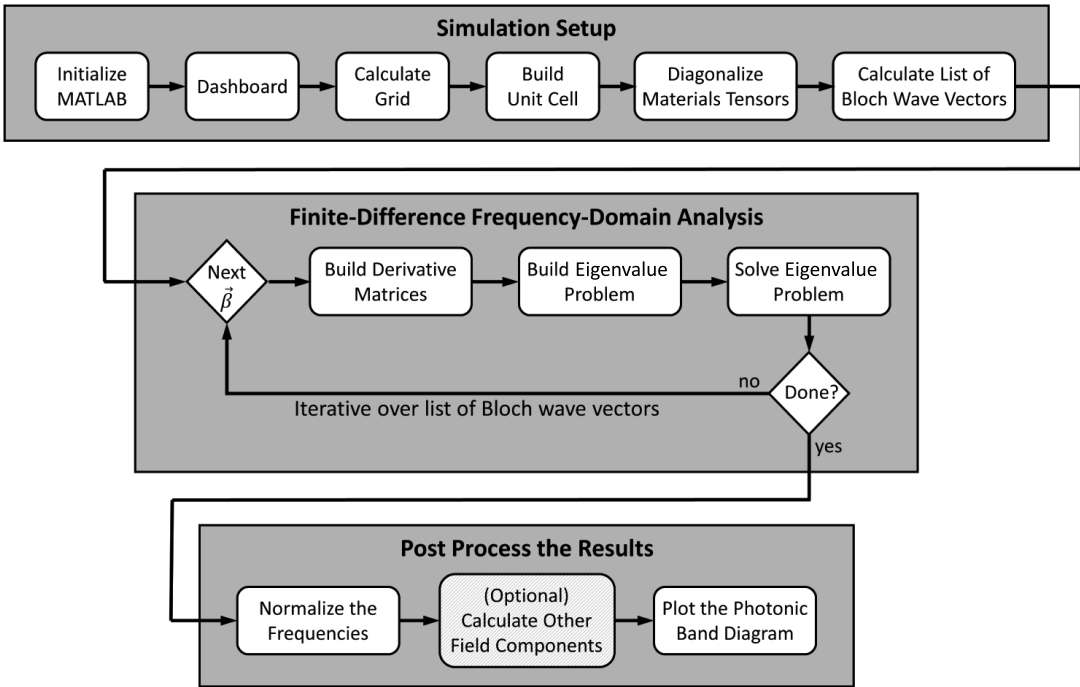


Figure 7.3 Block diagram of FDFD calculation of photonic bands.

steps that the eigenvalues will form continuous bands when they are plotted. For IFCs, the list of Bloch wave vectors will fill the entire area of the IBZ.

With the simulation setup, the code performs the actual FDFD analysis. It is composed of a large loop that iterates over the list of Bloch wave vectors calculated in the setup. The first task inside of this loop is to build the derivative matrices where the Bloch wave vector $\vec{\beta}$ is used in place of \vec{k}_{inc} for the PBCs. The second task is to calculate the A and B matrices of the eigenvalue problem from the diagonal materials matrices and the derivative matrices. The eigenvalue problem is constructed from (7.11) for the E mode (TM) or from (7.17) for the H mode (TE). The third task is to solve the eigenvalue problem for just the eigenvalues using the MATLAB function `eigs()`. It should be mentioned that the corresponding eigenvectors can also be calculated if it is desired to know the fields. The eigenvalues are recorded in an array before the loop repeats. It is usually bad practice to calculate the eigenvalues of all the eigenmodes. Instead, it is more numerically efficient to calculate only a small set corresponding to the lowest-order eigenmodes. It is rare for the higher-order modes of a photonic crystal to be used because the bands are so dense it is hard to isolate them for practical use. This is accomplished by calculating only the smallest eigenvalues. MATLAB makes obtaining a solution like this very easy.

After the eigenvalues are calculated for each Bloch wave vector, they are normalized according to (7.1). If the eigenvectors were also calculated, the other field components can be calculated at this point if they are needed. For photonic band diagrams, only the eigenvalues are required. The results are then visualized in a professional photonic band diagram. It is the convention in photonic band diagrams to illustrate the BZ with the IBZ highlighted and the key points of symmetry labeled

so that the horizontal axis can be interpreted. Sometimes a picture of the lattice is also provided.

7.3.1 Description of MATLAB Code for Calculating Photonic Band Diagrams

The MATLAB code described here calculates the photonic band structure of the two-dimensional lattice shown in Figure 7.4. The lattice is a square array of air cylinders cut through a dielectric slab. The lattice spacing is a , the radius of the holes is $r = 0.48a$, the relative permittivity of the dielectric is $\epsilon_{\text{fill}} = 12.0$, and the relative permittivity of the holes is $\epsilon_{\text{hole}} = 1.0$. It is typical to specify the lattice spacing simply as $a=1$. That is because the results of the simulation can be scaled to operate at any frequency or wavelength. It is also unnecessary to specify the thickness of the dielectric slab because the analysis assumes it is infinitely thick in the z -direction. It is important to remember that every simulation is always three-dimensional. However, when the device is infinitely extruded in the z -direction and wave propagation is restricted to the xy plane as illustrated in Figure 7.4, the math of the analyses reduces to just two dimensions. The basic band structure calculations are for infinitely periodic devices, but the analysis is fast and usually approximates the behavior of a finite lattice very well.

The MATLAB code described here can be downloaded at <https://empossible.net/fdbook/> and is called `Chapter7_photonicbanddiagram.m`. Lines 1 to 24 are the header and dashboard. MATLAB is initialized by closing all figure windows with `close all`, clearing the command window with `clc`, and the most important thing is to clear all variables from memory with `clear all`. Next is the dashboard where all of the hard-coded numbers that control the entire simulation are defined. The first group of code in this dashboard defines all of the parameters that describe the photonic crystal to be analyzed. This includes the lattice spacing a , the radius of the holes r , and the relative permittivity of the holes ϵ_{hole} and the dielectric ϵ_{fill} . The lattice spacing a is set to the numerical value of 1, and there is rarely an occasion to define it differently. It is not necessary to specify the units of the lattice spacing a because the normalized frequency will be calculated relative to a .

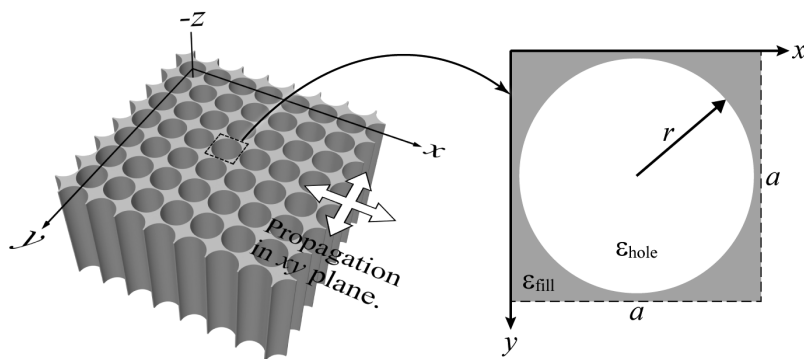


Figure 7.4 Square lattice for photonic band calculation. For this device, $r = 0.48a$, $\epsilon_{\text{hole}} = 1.0$, and $\epsilon_{\text{fill}} = 12.0$.

The second group of code in this dashboard defines all of the parameters related to the numerical calculations. This includes the size of the grid used to build the unit cell N_x and N_y , the approximate number of points to be used along the horizontal axis of the band diagram N_{BETA} , the number of photonic bands to calculate for each mode N_{BANDS} , and the maximum normalized frequency wn_{max} to show on the band diagram. A convergence study is needed in order to determine how many points should be used on the grid. It is recommended to start with values like $N_x=N_y=10$ until the code is working to ensure it runs fast while the code is being developed. It is similarly recommended to start with a low value for N_{BETA} around 20 to 50. This number does not affect the accuracy of the simulation at all, but higher values will resolve the bands more finely. Published band diagrams usually contain at least 200 points. It is only necessary to calculate as many bands as will appear below the maximum frequency to be displayed wn_{max} . This usually puts N_{BANDS} on the order of 5 to 10. The higher-order bands (10th band and above) tend to get crazy and are rarely of interest in practice. wn_{max} is set to focus the attention of the band diagram on just the bands of interest.

The MATLAB code to calculate the grid optimized to represent the unit cell resides in lines 24 to 41. The number of points, N_x and N_y , was specified in the dashboard so the only remaining parameters to calculate for the Yee grid are the resolution parameters dx and dy . Given the size of the unit cell a , the resolution parameters are easily calculated as $dx=a/N_x$ and $dy=a/N_y$. From these, the $2\times$ grid parameters N_{x2} , N_{y2} , $dx2$, and $dy2$ are calculated followed by the grid axes and a meshgrid on the $2\times$ grid. Observe that the meshgrid is centered at (0, 0) by centering both axis vectors $xa2$ and $ya2$ about zero.

The next step is to build a single unit cell on the grid. The MATLAB code that performs this operation resides in lines 42 to 65. The unit cell is constructed on the $2\times$ grid due to the curved geometry. Line 47 builds a circle of radius r centered in the array $ER2$. At this point, the array $ER2$ contains ones inside of the circle and zeros outside. For FDFD analysis, line 48 converts the array $ER2$ to numerical values of relative permittivity. Line 49 sets the relative permeability to all ones. Given the unit cell on the $2\times$ grid, the next group of code extracts the material tensor arrays ER_{xx} , ER_{yy} , ER_{zz} , UR_{xx} , UR_{yy} , and UR_{zz} that are defined on the standard Yee grid. The last group of code in this section visualizes the $ER2$ array to the first of three subplots in the current figure window.

The next step in the code is to calculate the list of Bloch wave vectors that steps around the perimeter of the IBZ in small increments. This is perhaps the most difficult part of the band calculation and the code to do this extends from lines 66 to 102. Lines 71 and 72 calculate the reciprocal lattice vectors T_1 and T_2 that describe the symmetry and size of the BZ [2]. There are only two lattice vectors because it is a two-dimensional lattice being analyzed. The reciprocal lattice vector T_1 is in the x -direction and its magnitude is set equal to the width of the full BZ, $|\vec{T}_1| = 2\pi/a$. Reciprocal lattice vector T_2 is in the y -direction and its magnitude is set equal to the height of the full BZ, $|\vec{T}_2| = 2\pi/a$. Lines 75 to 77 calculate the key points of symmetry at the vertices of the IBZ. These are Γ , X , and M . It is a good practice to calculate the key points of symmetry from the reciprocal lattice vectors, as is done here. The key point Γ (G in MATLAB) is always at the center of the BZ so it is set equal to position zero. The key point X is located at the far-right side of the BZ. This

distance is half of the BZ so \mathbf{x} is calculated as half of the reciprocal lattice vector \mathbf{T}_1 . The key point \mathbf{M} is located at the upper-right corner of the BZ and is calculated as half of \mathbf{T}_1 plus half of \mathbf{T}_2 . Line 80 creates an array \mathbf{KP} containing the key points of symmetry in the order that defines the path that will be taken around the IBZ. Line 81 records the symbols for the key points of symmetry that will be used when plotting the photonic band diagram. These are stored in the sequence \mathbf{KL} .

The most complicated part of calculating the list of Bloch wave vectors resides from lines 83 to 102. These two sections of code calculate the full list of Bloch wave vectors that will define the horizontal axis of the photonic band diagram. The spacing of the points corresponding to discrete values of the Bloch wave vector will be approximately equal. However, it is necessary to convey the relative lengths of the edges of the IBZ by using more points along the longer edges. To calculate the spacing, lines 84 to 88 calculate the total length of the path taken around the IBZ and store that value in the variable \mathbf{LIBZ} . This is done by adding the length of each individual edge in the chosen path. Observe that this piece of code functions correctly no matter how many, or in what order, the key points of symmetry are specified in the variable \mathbf{KP} . It would be very bad practice to hard code this loop to be `for m = 1 : 3` because this would only ever work for three points. Lines 91 to 101 calculate the list of Bloch wave vectors in a way that maintains the relative lengths of the edges taken along the IBZ. The variable \mathbf{dibz} calculated on line 91 is the approximate length of the increments as the total length divided by the number beta points, $\mathbf{dibz} = \mathbf{LIBZ}/\mathbf{NBETA}$. Next, the list of Bloch wave vectors \mathbf{BETA} is initialized with the first key point of symmetry on line 92. The array \mathbf{KT} will contain the array indices of \mathbf{BETA} that correspond to the key points of symmetry. It is initialized on line 93 with a value of 1 because the first element in the array \mathbf{BETA} is a key point of symmetry. Later, \mathbf{KT} will be used as tick mark positions for the key points of symmetry when creating the photonic band diagram. \mathbf{NBETA} is the number of points along the horizontal axis. It is recalculated after the list of Bloch wave vectors is formed. \mathbf{NBETA} is initialized here to 1 on line 94 because the array \mathbf{BETA} contains only a single Bloch wave vector at this place in the code. Starting at line 95, the loop iterates over each edge around the IBZ so that its endpoints are defined by the key points of symmetry. \mathbf{dK} is a vector pointing from the first key point to the second key point along the IBZ edge. \mathbf{N} is the number of Bloch wave vectors along the edge and is calculated as the length of the edge $|\mathbf{dK}|$ divided by the increment length \mathbf{dibz} . Line 98 adds all of the points along the IBZ edge, extending from one point away from the first key point all the way to the last key point along the edge. The first number added to the end of the array \mathbf{BETA} starts one point away from the key point of symmetry because the key point will already be the last element entered in the array \mathbf{BETA} . After the points are added to \mathbf{BETA} , the number of points is added to \mathbf{NBETA} to keep track of the total number of points in the array \mathbf{BETA} . The last point added to the array \mathbf{BETA} is a key point so its array index is recorded in the array \mathbf{KT} . After completing all of this, it is time to calculate the photonic bands!

Before iterating through the list of Bloch wave vectors, two things are taken care of in lines 107 to 117. First, the material tensor arrays \mathbf{ER}_{xx} , \mathbf{ER}_{yy} , \mathbf{ER}_{zz} , \mathbf{UR}_{xx} , \mathbf{UR}_{yy} , and \mathbf{UR}_{zz} must be converted into diagonal matrices. This is done in three steps. First, they are reshaped into one-dimensional arrays using $\mathbf{ER}_{xx}(:)$. Second, they are made sparse using `sparse()`. Third, the sparse one-dimensional array is placed

along the center diagonal of a square sparse matrix using `diag()`. All of this could also have been accomplished using the `spdiags()` function built into MATLAB. The last task before entering the main loop is to initialize the arrays that will store the band data. The first array `WNTE` will store the normalized frequencies for the TE mode and the second array `WNTM` will store the normalized frequencies for the TM mode.

The MATLAB code for the main loop that calculates the band data resides from lines 119 to 147. The main loop iterates `nbeta` from a value of 1 up to `NBETA`, which is the total number of points along the horizontal axis of the photonic band diagram. The first step in the main loop is to fetch the Bloch wave vector for the current iteration from the array `BETA` and store it in the variable `beta`. The second step is to build the derivative matrices that will be used to calculate the eigenvalue problems for both TE and TM modes. The derivative matrices are different for each iteration because the Bloch wave vector is different for each iteration and is used for the PBCs. The third and fourth steps build and solve the eigenvalue problems for the TE and TM modes separately and store the results in the arrays `WNTE` and `WNTM`. The TM mode analysis builds the *A* and *B* matrices for the generalized eigenvalue problem according to (7.11). The TE mode analysis builds the *A* and *B* matrices for the generalized eigenvalue problem according to (7.17). Otherwise, these two pieces of code are identical. The eigenvalue problem is solved using the MATLAB function `eigs()` that solves sparse eigenvalue problems. The function `eigs()` is given four input arguments. The first two input arguments are the matrices *A* and *B* that define the generalized eigenvalue problem being solved. The third input argument is the number of eigenvalues to calculate, which was defined as `NBANDS` in the dashboard. The fourth input argument is a value that the solver will find eigenvalues closest to. A value of zero is given here in order to find the eigenvalues of the lowest-order photonic bands. The function `eigs()` is able to calculate both eigenvalues and eigenvectors, but only the eigenvalues are needed for a photonic band diagram. For this reason, only a single output variable is given. The function `eigs()` runs much faster when it recognizes that it does not have to calculate eigenvectors. The eigenvalues are then sorted in ascending order because there are times when the order of the eigenmodes is scrambled. The raw eigenvalues are in the form of $k_{0,i}^2$ and stored in the array `WNTM` for the TM modes and in the array `WNTE` for the TE modes. When the main loop finishes the calculations, the eigenvalues are normalized to put them in the form of $\omega_{n,i} = a/\lambda_{0,i}$. This happens in lines 150 and 151.

The last task in the code is to draw the photonic band diagram from all of the data calculated in the program so far. The MATLAB code that draws a simple band diagram extends from lines 153 to 174. The figure window was divided into three subplots and the band diagram is drawn into the second and third subplots combined. This is done using the `subplot(1,3,2:3)` command so that the band diagram is given more space than the diagram of the unit cell. After this, the bands are drawn using discrete points, which looks better than continuous lines when few points are used. When many points are used to construct the band diagram, continuous lines look good as well. The plot command is given the array of numbers `1:NBETA` to use as the horizontal axis. The last set of code formats the view of the graphics. The *x*-axis limits are set to 1 and `NBETA`. The *y*-axis limits are set to 0 and `wnmax` that was defined in the dashboard. It would be improper to label the horizontal axis with

the numbers ranging from 1 to NBETA. Instead, the horizontal axis should be labeled with the key points of symmetry placed at the correct locations. This is accomplished using the `set()` command. The first input argument `gca` gets the properties of the current axes that are to be changed. The first property to be changed is `XTick` which are the tick mark locations along the horizontal axis where labels will be inserted. The tick marks were previously calculated and stored in the array `KT`. The second property to be changed is `XTickLabel` which is a sequence of strings containing the labels to put at the tick marks. These were defined previously in the sequence `KL`. After this, the x - and y -axes are labeled and the diagram is given a title.

The figure created by the MATLAB code described above is provided in Figure 7.5. This is very typical of the type of informal graphics a person would use during their day-to-day analysis work. It is good to always visualize the unit cell along with the band data to ensure any changes to the unit cell are incorporated correctly. Dressing up the graphics any more than this for casual work is not needed and will slow down the code.

The results shown in Figure 7.5 are little more than an indication that the code runs without error. It is not any kind of indication that the results are correct. A convergence study must be performed along with a benchmark simulation to build confidence that the results are correct and accurate. A convergence study examines the position of the bands as the grid resolution is increased by increasing the number of points N_x and N_y . After plotting two band diagrams calculated from different values of N_x and N_y , it was observed that the first band for the TE mode fluctuated the most at the midpoint between X and M along the horizontal axis. Figure 7.6 plots the normalized frequency of this band at $\vec{\beta} = 0.5\vec{T}_1 + 0.25\vec{T}_2$ for increasing values of N_x and N_y . Convergence seems to first happen at around $N_x=N_y=20$.

Given that convergence occurs for values of N_x and N_y greater than 20, the final band diagram was calculated and a more professional photonic band diagram was

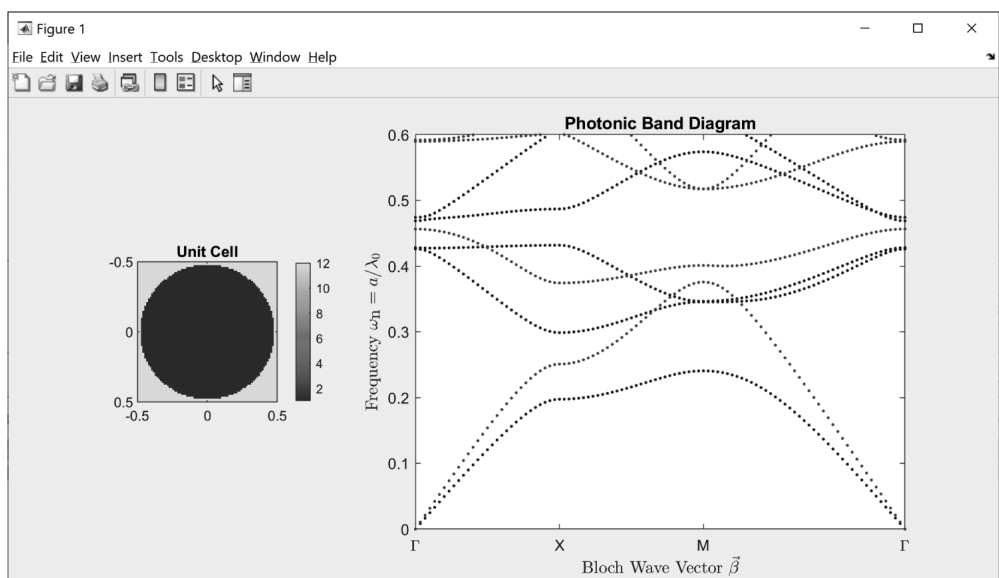


Figure 7.5 Unit cell and photonic band diagram.

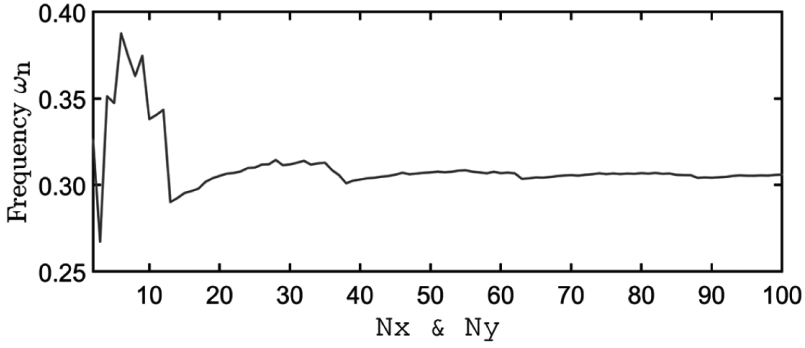


Figure 7.6 Convergence study for photonic band calculation for the lattice shown in Figure 7.4. Plot shows ω_n for the first TE mode at $\vec{\beta} = 0.5\vec{T}_1 + 0.25\vec{T}_2$.

created. This band diagram can be used for you to benchmark your own code. While simple and functional, the band diagram in Figure 7.5 is not appropriate for publication. In this diagram, the TE and TM bands can barely be distinguished from each other. Font sizes and line widths should be adjusted to be more visible. The band diagram should also show the BZ, IBZ, and the key points of symmetry. A professional band diagram depicting the same band data is provided in Figure 7.7. In this figure, the small bandgap appearing in the frequency range $0.46 \leq \omega_n \leq 0.47$ is highlighted in gray across the band diagram. If the lattice is not drawn somewhere else, as shown in Figure 7.4, it is recommended to provide a picture of the lattice as an inset in the band diagram.

7.3.2 Description of MATLAB Code for Calculating IFCs

The MATLAB code to calculate IFCs can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter7_IFCs.m`. The code is essentially just a modified

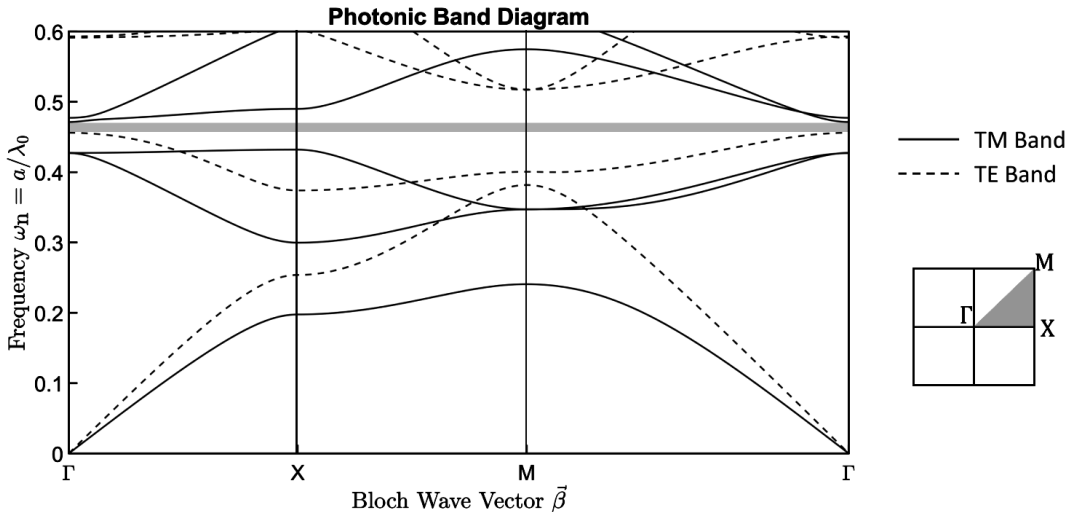


Figure 7.7 Professional photonic band diagram for the lattice described in Figure 7.4.

version of the code described previously that calculated photonic band diagrams. The header where MATLAB is initialized and the parameters describing the photonic crystal are identical to the previous code. There is a slightly different set of parameters that control the FDFD simulation from lines 19 to 24. The number of points on the Yee grid N_x and N_y is defined directly in the dashboard. In addition, the number of discrete points that will be used to fill in the IBZ with band data is defined through the variables NB_x and NB_y . The more points used, the more accurately resolved the IFCs will be. $NBANDS$ defines how many bands are stored in memory. It is usually only the lower-order bands that are of interest so this number should be kept small.

Lines 25 to 65 calculate the grid and build the unit cell of the photonic crystal onto that grid. This code is identical to that for calculating photonic bands. Lines 66 to 80 calculate the list of Bloch wave vectors like before, but the list is constructed differently. For photonic band diagrams, the list of Bloch wave vectors marched around the perimeter of the IBZ. For IFCs, the list of Bloch wave vectors must fill the entire area of the BZ. A fast and efficient IFC calculator will only analyze the lattice at points within the IBZ. All other points in the BZ can be copied from a point within the IBZ. To start, a meshgrid of all of the points throughout the entire BZ is calculated on lines 71 to 73. The Bloch wave vector components are stored in the arrays BX and BY . Line 76 calculates `ind_ibz` that contains all of the array indices of Bloch wave vectors in $BETA$ that resides within the IBZ. Lines 77 and 78 convert the linear array indices in `ind_ibz` to two-dimensional array indices where all the points of the BZ are viewed as elements in a two-dimensional array. Last, line 79 determines the total number of points in the IBZ from the length of the array `ind_ibz`.

At this point, two small tasks must be performed before the FDFD analysis. Lines 85 to 91 form the diagonal materials matrices that will be used to build the eigenvalue problems. Then, lines 93 to 95 initialize the arrays where the band data will be stored. $WNTE$ will store the normalized frequencies for the TE mode while $WNTM$ will store the normalized frequencies for the TM mode.

The main loop from lines 97 to 127 calculates and stores the band data for all points within the IBZ. Line 103 of the main loop fetches the next Bloch wave vector from the list. Lines 105 to 109 call the `yeeDer2d()` function to build the derivative matrices. The Bloch wave vector is given as an input argument to incorporate the correct PBCs. Lines 112 to 117 calculate the TM mode bands and store the data in the array $WNTM$. Lines 112 and 113 build the A and B matrices of the generalized eigenvalue problem $A\mathbf{x} = \lambda B\mathbf{x}$ in (7.11). Line 114 solves the generalized eigenvalue problem and returns the eigenvalues in the array D . Since only the eigenvalues are returned from the call to `eigs()`, the eigenvalues are returned in a linear array instead of a matrix. Line 115 calculates k_0 from the eigenvalue k_0^2 and drops any small imaginary component that may exist due to numerical error. Line 116 then sorts the eigenvalues in increasing order so that the lowest order modes are first. Line 117 normalizes the eigenvalues according to (7.1) and stores them in $WNTM$. The process is repeated for TE mode analysis in lines 120 to 125 and the results are stored in $WNTE$. The primary difference is the calculation of the matrices A and B that come from (7.17).

The main loop only calculated the band data within the IBZ. To visualize the band data more intuitively, the band data throughout the entire BZ is constructed

from the band data within just the IBZ. This is done for speed and efficiency because only the points within the IBZ have to be analyzed. This is performed on lines 130 to 144 by iterating through the bands and unfolding the data for both TE and TM modes. Line 133 grabs the data for the current TE band. The array w_n spans the entire BZ, but at this point, only the points in w_n that lie within the IBZ contain actual band data. Line 134 unfolds the IBZ data to fill a quadrant of the BZ in the array w_n . However, doing this doubled the diagonal elements so line 135 corrects this. Lines 136 and 137 mirror the band data from the quadrant to fill the entire BZ. The complete array w_n is then copied back into the array W_{NTE} , overwriting the original data in this array. This unfolding procedure is repeated for the TM mode on lines 140 to 145.

The final portion of the code visualizes the IFCs for the first two TE and TM mode bands. Lines 152 to 161 plot the IFCs for the first TE band. Line 154 fetches the band data into the array w_n . Line 155 calculates an array of 20 numbers that extend from the lowest value in w_n to the highest value in w_n . The numbers in this array will become the frequency values where the IFCs are drawn. Line 156 calls MATLAB's `contour()` function that calculates and plots the IFC lines. Lines 157 to 161 format and label the plot to look professional. Lines 163 to 172 repeat this for

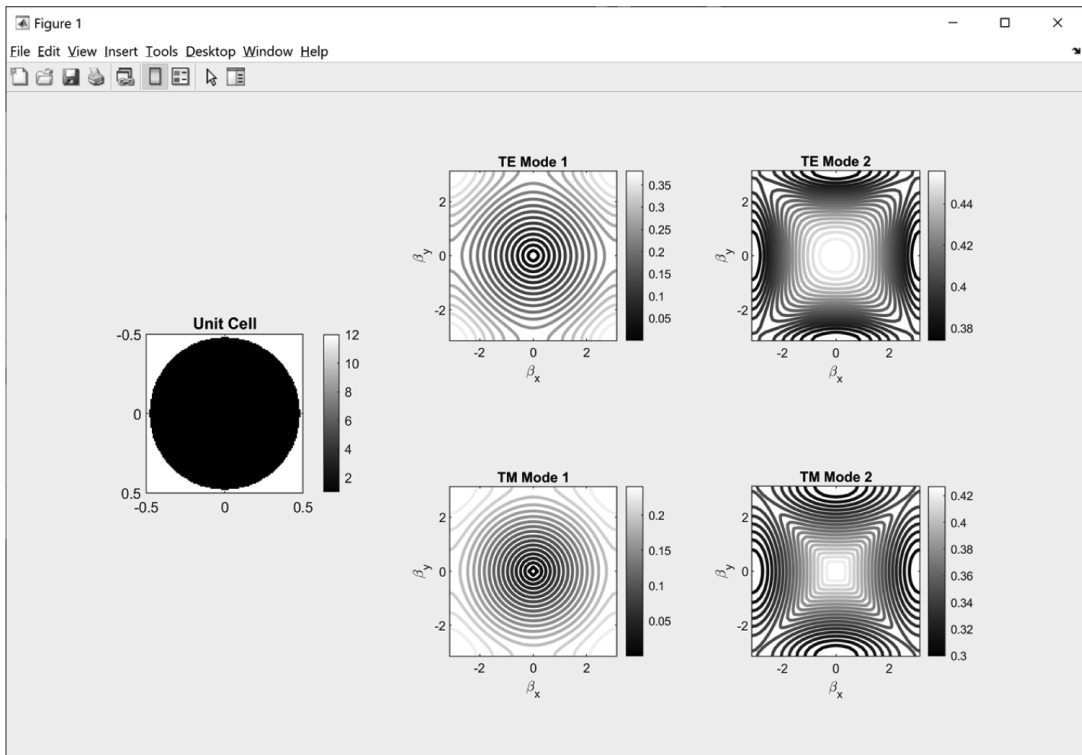


Figure 7.8 Unit cell and IFCs for the first two TE and TM bands for the lattice described in Figure 7.4.

the second TE band. Lines 174 to 194 repeat all of this for the first two TM bands. The figure created by the MATLAB code described above is provided in Figure 7.8.

Like before, the results shown in Figure 7.8 are little more than an indication that the code runs without error. A convergence study must be performed along with a benchmark simulation to build confidence that the results are correct and accurate. It was previously shown that convergence occurs for values of N_x and N_y above 20. In this code, $N_x=N_y=60$ was used with little impact on speed and a more professional set of IFCs was created to showcase the results. This data can be used for you to benchmark your own IFC code and is provided in Figure 7.9.

The IFCs are much more useful than the photonic band diagram for analyzing the dispersion properties of photonic crystals. For example, flat IFCs lead to a phenomenon called self-collimation where beams are forced to propagate along a single direction without spreading [2, 5, 6]. The direction of self-collimation will be perpendicular to the flat IFC. After inspection of the IFCs in Figure 7.9, flat IFCs can be identified for each of the four cases. The first TM band will self-collimate at around $\omega_n = 0.217$ because the IFC is flat for waves propagating along the diagonals. Identifying the normalized frequency more precisely is more easily accomplished using color or by experimenting with which contours are drawn. In this case, self-collimation will occur for waves propagating diagonally through the lattice because the bands are flat in the diagonal directions. This normalized frequency will be used in Chapter 8 where a self-collimating photonic crystal is simulated. The first TE

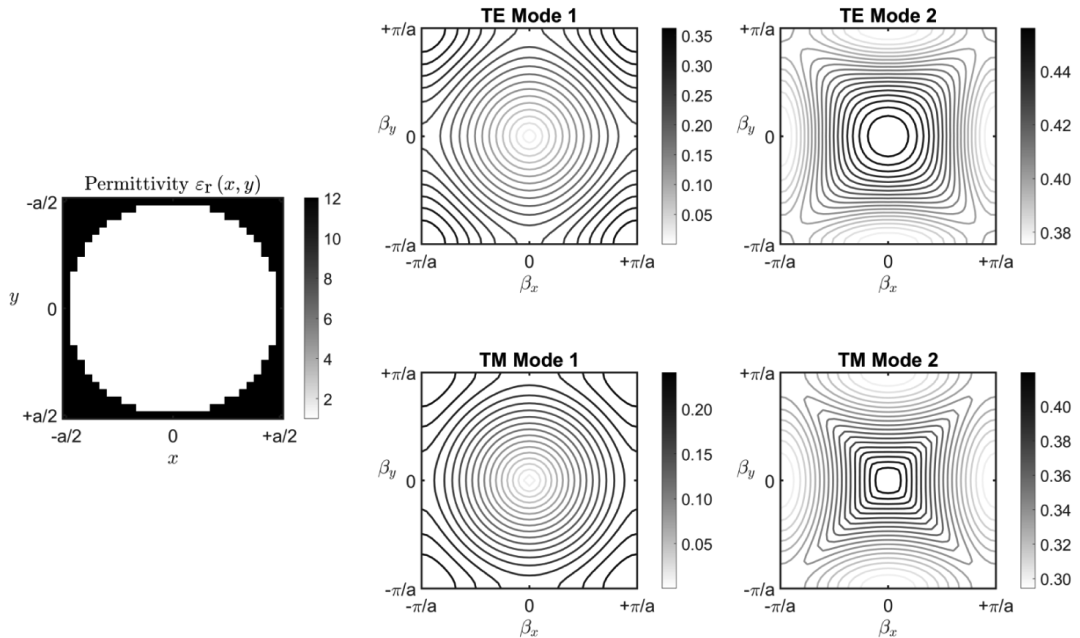


Figure 7.9 Professional IFCs for the lattice in Figure 7.4.

band will self-collimate in the same directions at a normalized frequency around $\omega_n = 0.28$. The second TM band will self-collimate in directions parallel to the x - and y -axes at a normalized frequency around $\omega_n = 0.44$. The second TE band will self-collimate in the same directions at around $\omega_n = 0.43$.

References

- [1] Joannopoulos, J. D., *et al.*, *Molding the Flow of Light*, Princeton, NJ: Princeton Univ. Press, 2008.
- [2] Rumpf, R. C., "Engineering the Dispersion and Anisotropy of Periodic Electromagnetic Structures," *Solid State Physics*, Vol. 66, 2015, pp. 213–300.
- [3] Aghaie, K. Z., S. Fan, and M. J. Dignonnet, "Birefringence Analysis of Photonic-Bandgap Fibers Using the Hexagonal Yee's Cell," *IEEE J. of Quantum Electronics*, Vol. 46, No. 6, 2010, pp. 920–930.
- [4] Guo, S., *et al.*, "Photonic Band Gap Analysis Using Finite-Difference Frequency-Domain Method," *Optics Express*, Vol. 12, No. 8, 2004, pp. 1741–1746.
- [5] Digaum, J. L., *et al.*, "Tight Control of Light Beams in Photonic Crystals with Spatially-Variant Lattice Orientation," *Optics Express*, Vol. 22, No. 21, 2014, pp. 25788–25804.
- [6] Rumpf, R. C., and J. J. Pazos, "Optimization of Planar Self-Collimating Photonic Crystals," *Journal of the Optical Society of America A*, Vol. 30, No. 7, 2013, pp. 1297–1304.

FDFD for Scattering Analysis

This chapter covers the formulation and implementation of finite-difference frequency-domain (FDFD) for two-dimensional scattering simulations. A powerful and versatile technique is described to incorporate a wide variety of sources into FDFD [1]. Sources covered in this chapter include plane waves, Gaussian beams, and guided modes. The general flow of FDFD is discussed which is common to most scattering simulations. Before simulating devices, a sequence of simpler simulations is suggested that help identify and troubleshoot any problems which may exist in a newly written code. After this, three device examples are given that include a diffraction grating, a photonic crystal, and an optical-integrated circuit (OIC). Complete MATLAB codes are given and explained for all of these devices.

8.1 Formulation of FDFD for Scattering Analysis

The following discussion covers the formulation of FDFD for two-dimensional scattering analysis where all of the equations are derived that will be typed into MATLAB and solved. First, the matrix wave equations will be derived and will have the general form $\mathbf{A}\mathbf{f} = 0$. Second, the QAAQ implementation of the total-field/scattered-field (TF/SF) technique will be introduced as a simple and versatile way of incorporating many different types of sources into FDFD simulations [1]. With a source incorporated, the matrix equation will have the form $\mathbf{A}\mathbf{f} = \mathbf{b}$ that can be solved for the field as $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$. Postprocessing the field will be discussed in order to calculate reflection and transmission from two periodic devices and an OIC.

8.1.1 Matrix Wave Equations for Two-Dimensional Analysis

For most scattering problems, the frequency is known at the start of the simulation. For this reason, formulation begins with the following matrix equations derived in Chapter 4 where the grid was normalized by multiplying the spatial coordinates by the free space wavenumber k_0 .

$$\mathbf{D}_y^e \mathbf{e}_z - \mathbf{D}_z^e \mathbf{e}_y = \mu_{xx} \tilde{\mathbf{h}}_x \quad (8.1)$$

$$\mathbf{D}_z^e \mathbf{e}_x - \mathbf{D}_x^e \mathbf{e}_z = \mu_{yy} \tilde{\mathbf{h}}_y \quad (8.2)$$

$$\mathbf{D}_x^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = \mu_{zz} \tilde{\mathbf{h}}_z \quad (8.3)$$

$$\mathbf{D}_{y'}^h \tilde{\mathbf{h}}_z - \mathbf{D}_z^h \tilde{\mathbf{h}}_{y'} = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (8.4)$$

$$\mathbf{D}_z^h \tilde{\mathbf{h}}_x - \mathbf{D}_{x'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (8.5)$$

$$\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_{y'} - \mathbf{D}_y^h \tilde{\mathbf{h}}_{x'} = \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (8.6)$$

In Chapter 2, it was shown that it is possible to reduce the math of three-dimensional problems to two dimensions when: (1) the device is uniform along z , (2) wave propagation is restricted to the xy plane, and (3) materials are isotropic or diagonally anisotropic. When these conditions are met, any derivative in the z -direction must be zero because nothing varies in the z -direction. For this reason, $\mathbf{D}_{z'}^e = \mathbf{D}_z^h = 0$ and (8.1) to (8.6) reduce to

$$\mathbf{D}_{y'}^e \mathbf{e}_z = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x \quad (8.7)$$

$$-\mathbf{D}_{x'}^e \mathbf{e}_z = \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y \quad (8.8)$$

$$\mathbf{D}_{x'}^e \mathbf{e}_y - \mathbf{D}_y^e \mathbf{e}_x = \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \quad (8.9)$$

$$\mathbf{D}_{y'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x \quad (8.10)$$

$$-\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_z = \boldsymbol{\epsilon}_{yy} \mathbf{e}_y \quad (8.11)$$

$$\mathbf{D}_{x'}^h \tilde{\mathbf{h}}_{y'} - \mathbf{D}_y^h \tilde{\mathbf{h}}_{x'} = \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \quad (8.12)$$

Observe that these equations have decoupled into two independent sets of equations. Equations (8.7), (8.8), and (8.12) contain \mathbf{e}_z , $\tilde{\mathbf{h}}_x$, and $\tilde{\mathbf{h}}_y$, and describe the E mode (TM polarization). A matrix wave equation in terms of just \mathbf{e}_z is derived by solving (8.7) for $\tilde{\mathbf{h}}_x$, solving (8.8) for $\tilde{\mathbf{h}}_y$, and substituting both of these expressions into (8.12). This gives

$$\left(\mathbf{D}_{x'}^h \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{x'}^e + \mathbf{D}_{y'}^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{y'}^e + \boldsymbol{\epsilon}_{zz} \right) \mathbf{e}_z = 0 \quad (8.13)$$

$$\tilde{\mathbf{h}}_x = \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{y'}^e \mathbf{e}_z \quad (8.14)$$

$$\tilde{\mathbf{h}}_y = -\boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{x'}^e \mathbf{e}_z \quad (8.15)$$

Equations (8.9) to (8.11) contain $\tilde{\mathbf{h}}_z$, \mathbf{e}_x , and \mathbf{e}_y , and describe the H mode (TE polarization). A matrix wave equation in terms of just $\tilde{\mathbf{h}}_z$ is derived by solving (8.10) for \mathbf{e}_x , solving (8.11) for \mathbf{e}_y , and substituting both of these expressions into (8.9). This gives

$$\left(\mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{yy}^{-1} \mathbf{D}_{x'}^h + \mathbf{D}_y^e \boldsymbol{\epsilon}_{xx}^{-1} \mathbf{D}_{y'}^h + \boldsymbol{\mu}_{zz} \right) \tilde{\mathbf{h}}_z = 0 \quad (8.16)$$

$$\mathbf{e}_x = \boldsymbol{\epsilon}_{xx}^{-1} \mathbf{D}_{y'}^h \tilde{\mathbf{h}}_z \quad (8.17)$$

$$\mathbf{e}_y = -\boldsymbol{\epsilon}_{yy}^{-1} \mathbf{D}_x^h \tilde{\mathbf{h}}_z \quad (8.18)$$

In standard form $\mathbf{A}\mathbf{f} = \mathbf{b}$, the matrix wave equations are written as

$$\mathbf{A}_e \mathbf{e}_z = \mathbf{0} \quad (8.19)$$

$$\mathbf{A}_h \tilde{\mathbf{h}}_z = \mathbf{0} \quad (8.20)$$

where

$$\mathbf{A}_e = \mathbf{D}_{x'}^h \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{x'}^e + \mathbf{D}_{y'}^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{y'}^e + \boldsymbol{\epsilon}_{zz} \quad (8.21)$$

$$\mathbf{A}_h = \mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{yy}^{-1} \mathbf{D}_{x'}^h + \mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{xx}^{-1} \mathbf{D}_{y'}^h + \boldsymbol{\mu}_{zz} \quad (8.22)$$

The matrices \mathbf{A}_e and \mathbf{A}_h are called *wave matrices* and they enforce Maxwell's equations across the Yee grid. Note that neither (8.19) nor (8.20) is yet solvable. Attempting to solve (8.19) for \mathbf{e}_z gives $\mathbf{e}_z = \mathbf{A}_e^{-1} \mathbf{0} = \mathbf{0}$ and attempting to solve (8.20) for $\tilde{\mathbf{h}}_z$ gives $\tilde{\mathbf{h}}_z = \mathbf{A}_h^{-1} \mathbf{0} = \mathbf{0}$. Both of these are trivial solutions containing all zeros.

8.2 Incorporating Sources

Equations (8.19) and (8.20) are not solvable because all they do is enforce Maxwell's equations. No source, or excitation, has been defined from which to calculate the fields. Injecting sources into an FDFD simulation can be complicated and confusing. In this book, a very simple and powerful technique will be used that makes injecting sources and postprocessing the fields easy. Over the years, the author and his research team have come to call this the QAAQ technique [1], pronounced as “quack” like the sound made by a duck.

The TF/SF technique identifies points on the grid as either a *total-field* (TF) quantity or a *scattered-field* (SF) quantity [1–3]. TF quantities contain both the source and the fields scattered by the device being simulated. SF quantities contain only fields scattered from a device and not the source. In the physical world, everything is TF. When the finite-difference equations are written for each point on the grid, some finite-difference equations adjacent to the interface between TF and SF regions will contain both TF and SF quantities. Equations must contain only SF quantities or only TF quantities, not both. The source is injected when the finite-difference equations are corrected.

The following sections will describe the QAAQ technique in detail, but a summary is described here to put all of the implementation steps in context. First, the source field is calculated across the entire grid and reshaped into a column vector \mathbf{f}_{src} . Second, the SF masking matrix \mathbf{Q} is constructed to identify all of the SF cells on the grid. Third, the column vector \mathbf{b} that describes the source is calculated according to $\mathbf{b} = (\mathbf{Q}\mathbf{A} - \mathbf{A}\mathbf{Q})\mathbf{f}_{\text{src}}$, where \mathbf{A} is the wave matrix. Now the matrix wave equation has the form $\mathbf{A}\mathbf{f} = \mathbf{b}$ which can be solved as $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$. That is it!

8.2.1 Derivation of the QAAQ Equation

The derivation of the QAAQ equation can be difficult to understand, but is presented in this section for completeness. Fortunately, it is not necessary to understand the derivation in order to use the QAAQ equation. Using the SF masking matrix \mathbf{Q} , it is possible to isolate the source function \mathbf{f}_{src} to just the TF region or to just the SF region. This is done according to

$$\mathbf{f}_{\text{src,TF}} = (\mathbf{I} - \mathbf{Q})\mathbf{f}_{\text{src}} \quad (8.23)$$

$$\mathbf{f}_{\text{src,SF}} = \mathbf{Q}\mathbf{f}_{\text{src}} \quad (8.24)$$

At grid cells immediately adjacent to the TF/SF interface, there exist finite-difference equations in the TF region that contain some SF terms. These quantities are not compatible. The source \mathbf{f}_{src} must be added to these SF terms to make them look like TF terms. When this is done as $\mathbf{A}(\mathbf{f} + \mathbf{f}_{\text{src,SF}}) = \mathbf{0}$, the numerical values that are added to the matrix equation $\mathbf{A}\mathbf{f} = \mathbf{0}$ are $\mathbf{A}\mathbf{f}_{\text{src,SF}}$. However, the correction terms should only be added to the TF finite-difference equations. The \mathbf{Q} matrix is used a second time to isolate the correction terms to just the TF region as $(\mathbf{I} - \mathbf{Q})\mathbf{A}\mathbf{f}_{\text{src,SF}}$. Altogether, the column vector \mathbf{b}_{TF} that must be added to the matrix equation to correct all the TF equations containing SF terms is

$$\mathbf{b}_{\text{TF}} = (\mathbf{I} - \mathbf{Q})\mathbf{A}\mathbf{f}_{\text{src,SF}} \quad (8.25)$$

$$\mathbf{A}\mathbf{f} + \mathbf{b}_{\text{TF}} = \mathbf{0} \quad (8.26)$$

Similarly, there exist finite-difference equations in the SF region that contain some TF terms. The source must be subtracted from these TF terms to make them look like SF terms. When this is done as $\mathbf{A}(\mathbf{f} - \mathbf{f}_{\text{src,TF}}) = \mathbf{0}$, the numerical values that are subtracted from the matrix equation $\mathbf{A}\mathbf{f} = \mathbf{0}$ are $\mathbf{A}\mathbf{f}_{\text{src,TF}}$. However, the correction terms should only be subtracted from the SF finite-difference equations. The \mathbf{Q} matrix is used a second time to isolate the correction terms to just the SF region as $\mathbf{Q}\mathbf{A}\mathbf{f}_{\text{src,TF}}$. Altogether, the column vector \mathbf{b}_{SF} that must be subtracted to correct all the SF equations containing TF terms is

$$\mathbf{b}_{\text{SF}} = \mathbf{Q}\mathbf{A}\mathbf{f}_{\text{src,TF}} \quad (8.27)$$

$$\mathbf{A}\mathbf{f} - \mathbf{b}_{\text{SF}} = \mathbf{0} \quad (8.28)$$

To implement the TF/SF technique, correction terms for both the TF and SF regions are needed. Combining the correction terms from (8.25) and (8.27) gives

$$\mathbf{A}\mathbf{f} + \mathbf{b}_{\text{TF}} - \mathbf{b}_{\text{SF}} = \mathbf{0} \quad (8.29)$$

The column vectors \mathbf{b}_{TF} and \mathbf{b}_{SF} come from quantities known at the start of the simulation so they can both be given numerical values and moved to the right-hand side of the equation. This gives an expression for the overall source vector \mathbf{b} .

$$\mathbf{b} = -\mathbf{b}_{\text{TF}} + \mathbf{b}_{\text{SF}} = -(\mathbf{I} - \mathbf{Q})\mathbf{A}\mathbf{f}_{\text{src,SF}} + \mathbf{Q}\mathbf{A}\mathbf{f}_{\text{src,TF}} \quad (8.30)$$

$$\mathbf{A}\mathbf{f} = \mathbf{b} \quad (8.31)$$

Furthermore, (8.23) and (8.24) are substituted into (8.30) to put the equation for \mathbf{b} completely in terms of the original source function \mathbf{f}_{src} . After simplifying, the result is the famous QAAQ equation.

$$\mathbf{b} = (\mathbf{QA} - \mathbf{AQ})\mathbf{f}_{\text{src}} \quad (8.32)$$

In this equation, \mathbf{A} is the wave matrix derived previously. At first glance, it might seem that this equation will always give $\mathbf{b} = 0$. However, recall from Chapter 1 that the order of matrix multiplication cannot be reversed so $\mathbf{QA} \neq \mathbf{AQ}$ and the source vector \mathbf{b} will not be all zeros. When considering E and H modes in FDFD, it may make more sense to write the QAAQ equation as follows.

$$\mathbf{A}_e \mathbf{e}_z = \mathbf{b}_e \quad \mathbf{b}_e = (\mathbf{QA}_e - \mathbf{A}_e \mathbf{Q})\mathbf{f}_{\text{src}} \quad (8.33)$$

$$\mathbf{A}_h \tilde{\mathbf{h}}_z = \mathbf{b}_h \quad \mathbf{b}_h = (\mathbf{QA}_h - \mathbf{A}_h \mathbf{Q})\mathbf{f}_{\text{src}} \quad (8.34)$$

With the source vector calculated, the fields can be solved for the E and H modes according to

$$\mathbf{e}_z = \mathbf{A}_e^{-1} \mathbf{b}_e \quad (8.35)$$

$$\tilde{\mathbf{h}}_z = \mathbf{A}_h^{-1} \mathbf{b}_h \quad (8.36)$$

It should be noted that the QAAQ equation will work regardless of the grid used or the order of accuracy of the finite differences. In fact, the QAAQ equation can be applied to other numerical techniques like the finite element method when node elements are used. This is because the specifics of the underlying math and the locations of the points on the grid were not part of the derivation.

The source function \mathbf{f}_{src} , SF masking matrix \mathbf{Q} , and source vector \mathbf{b} for a plane wave source are provided in Figure 8.1. Each of these has been reshaped back to a two-dimensional grid to visualize them in the most meaningful way. The shaded

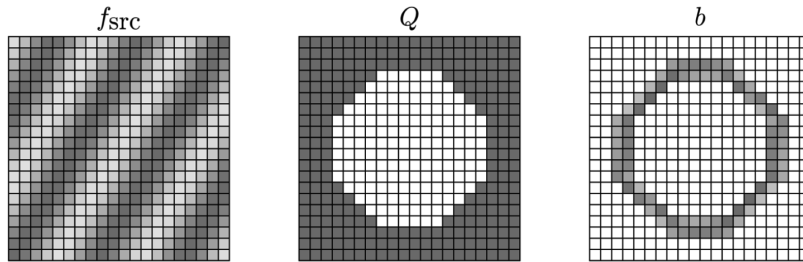


Figure 8.1 Representative data from QAAQ technique: (a) source function, (b) scattered-field masking function just prior to becoming a diagonal matrix, and (c) source vector reshaped to a two-dimensional array for visualization.

regions in the \mathbf{Q} array are cells identified to be in the SF. Observe that the source vector \mathbf{b} is only non-zero in the cells immediately adjacent to the TF/SF interface because these are the cells that required corrections to the finite-difference equations.

8.2.2 Calculating the Source Field $f_{\text{src}}(x,y)$

The source field $f_{\text{src}}(x,y)$ is the field that would result from a simulation if nothing was added to the simulation to cause the source to scatter. It must be very close to a rigorous solution to Maxwell's equations or spurious waves will be injected into the simulation. The source field will be calculated across the entire grid, even though it is only strictly needed in the grid cells immediately adjacent to the TF/SF interface. This practice allows the TF/SF interface to be changed without having to identify the new points adjacent to the interface and calculating the source at these new points. The code will be simpler with minimal impact on efficiency, and TF and SF regions can be altered at any time without further consideration.

It is possible to calculate the source field directly from analytical equations or numerical calculations. Plane waves, cylindrical waves, and Gaussian beam sources can all be calculated from analytical equations. Guided modes in waveguides and Bloch modes in photonic crystals often do not have analytical equations for them, but the source fields can still be calculated numerically using the techniques covered in Chapters 6 and 7. Some common source functions are illustrated in Figure 8.2. Figure 8.2(a) is a plane wave source that is commonly used when simulating scattering from an object or device. The analytical equation and MATLAB code to calculate this source function are

$$f_{\text{src}}(x,y) = \exp\left[-j\left(k_{x,\text{inc}}x + k_{y,\text{inc}}y\right)\right] \quad (8.37)$$

```
fsrc = exp(-1i*(kxinc*X + kyinc*Y));
```

In the MATLAB equation above, the variables X and Y are the meshgrid parameters for the Yee grid. Figure 8.2(b) shows a cylindrical wave source. This can be used to model radiation from a dipole or as a quick and easy source to excite waves in all directions at the same time. The analytical equation and MATLAB code to calculate a cylindrical source function are

$$f_{\text{src}}(x,y) = \frac{1}{\sqrt{r}} \exp(-jkr) \quad (8.38)$$

```
R = sqrt(X.^2 + Y.^2);
fsrc = exp(-1i*k*R)./sqrt(R);
```

Figure 8.2(c) shows a Gaussian beam source. This source has many applications including when it is desired to only illuminate one part of a device or when the simulation is to be more consistent with experiments. Gaussian beams are excellent for visualizing things like refraction and excitation of surface waves. The analytical equation and MATLAB code to calculate this source function are given below. Incorporating an angle θ for the Gaussian beam is very easy and only requires

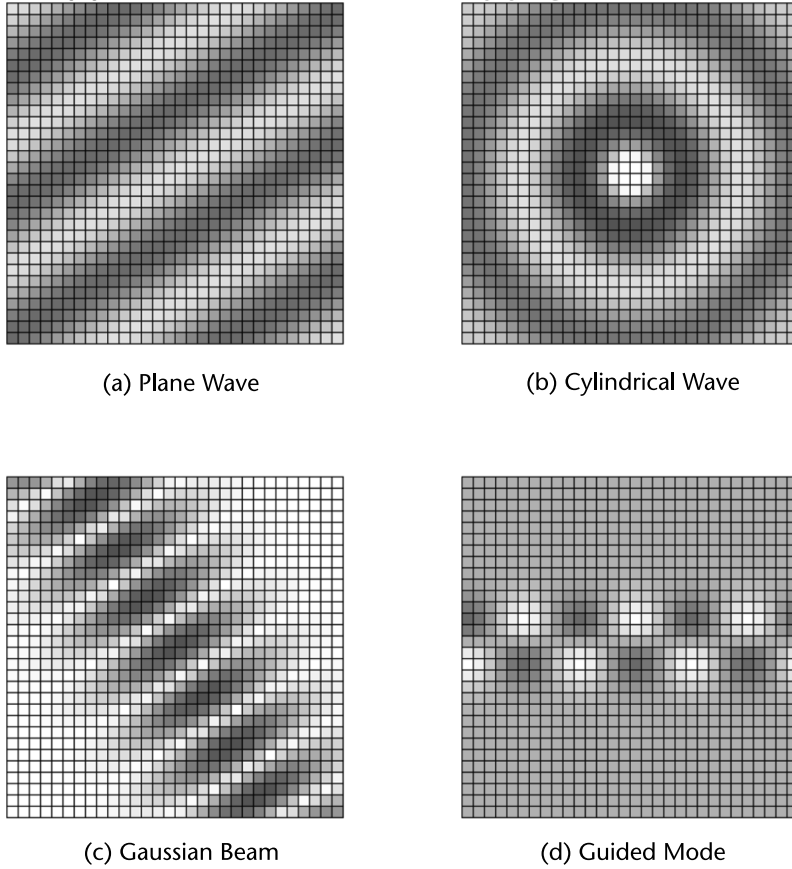


Figure 8.2 Common source functions in FDTD: (a) plane wave, (b) cylindrical wave, (c) Gaussian beam, and (d) guided mode.

that the meshgrid from which the source is calculated to be rotated by the desired angle θ . Rotating meshgrids were described in more detail in Chapter 1.

$$f_{\text{src}}(x, y) = \exp\left[-\left(\frac{x}{w}\right)^2\right] \exp(-jky) \quad (8.39)$$

```
[TH,R] = cart2pol(X,Y);
[XR,YR] = pol2cart(TH+theta,R);
fsrc = exp(-(XR/W).^2).*exp(-1i*k*YR);
```

Equation (8.39) is a highly simplified form of the more general equation for a Gaussian beam [4]. Therefore, the Gaussian beam used here is not a rigorous solution to Maxwell's equations and does not include the divergence or radial phase of a true Gaussian beam. Extra care must be given when injecting this source. The Gaussian beam can be injected without problems as long it is injected from a single TF/SF interface that is mostly perpendicular to the beam. This will be demonstrated in Section 8.4.3 where a photonic crystal will be illuminated by a Gaussian beam.

Figure 8.2(d) shows a guided wave source. This source is used whenever it is desired to excite a waveguide with one of its guided modes in order to simulate guided-wave devices. For many waveguides, no analytical expression exists for the guided mode. In these cases, a waveguide mode calculation from Chapter 6 can be performed in the cross section of the grid at the input of the waveguide. This will require the FDFD algorithm to be modified to perform a slab waveguide analysis when calculating the source field. After the desired mode is calculated, the source field is calculated by extrapolating the mode across the grid in the $+x$ -direction by adding phase according to the effective refractive index as $\exp(-jk_0 n_{\text{eff}} x)$. When done correctly, the mode should look as if it has propagated across the entire grid through a uniform waveguide. In this case, a second-order guided mode is being launched in the $+x$ -direction and a slab waveguide analysis from Chapter 6 was used to calculate the mode.

8.2.3 Calculating the SF Masking Matrix \mathbf{Q}

Building the \mathbf{Q} matrix is quite easy, but making the correct choice for the TF and SF regions is not as straightforward. It is best to think in terms of the line formed at the interface between TF and SF regions instead of the regions themselves. The TF/SF interface must completely cross through the source function in order to completely describe the source. For this reason, different sources may require different TF/SF interfaces. As long as the TF/SF interface completely crosses through the source in at least one place, additional TF and SF regions can be added arbitrarily as long as they do not disrupt the original TF/SF interface. Ideally, there is only one TF/SF interface, and the interface is perpendicular to the direction of the source.

Figure 8.3 illustrates good and poor choices of TF and SF regions for a Gaussian beam source simulated in a vacuum where no scattering should occur. Figure 8.3(a) shows the source function calculated across the entire grid using (8.39). The ideal TF/SF interface is one that crosses completely through the source in a direction perpendicular to the direction of the source, as shown in Figure 8.3(b). No waves are observed in the SF region, and the simulated beam diverges as it propagates as a physical beam would do. Figure 8.3(c) shows a first poor choice where the TF/SF interface forms a box around the simulation. This is a poor choice because the source field and simulated source are very different from each other at the far-right side of the grid. The source field is not a rigorous solution to Maxwell's equations so it does not diverge like the simulated beam. A significant wave is observed in the SF region at the right side of the simulation despite there being nothing for the beam to scatter from to produce an SF wave. This false wave appearing in the SF region will introduce significant errors in any postprocessing calculations. The second type of poor choice is shown in Figure 8.3(d) where the TF/SF interface does not completely cross through the source. This also exhibits a strong wave in the SF region where no waves should be observed. Last, Figure 8.3(e) shows a third poor choice where the TF/SF interface is not perpendicular to the direction of the source. Not only will this amplify problems due to numerical dispersion, but the diverging simulated wave fails to match the source field over the entire interface because the source field is not a rigorous solution to Maxwell's equations. As a consequence,

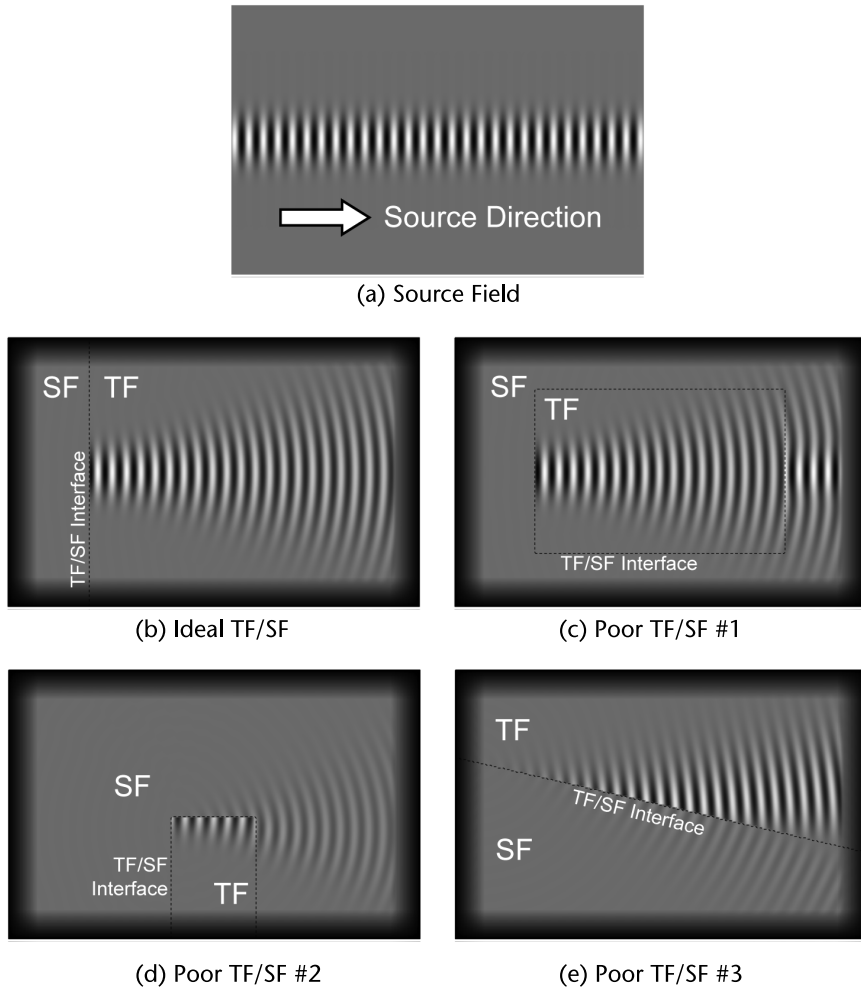


Figure 8.3 Examples of good and poor TF/SF choices for a Gaussian beam source: (a) source field, (b) ideal TF/SF, (c) poor TF/SF #1, (d) poor TF/SF #2, and (e) poor TS/SF #3.

the beam observed in the TF region is distorted and waves are observed in the SF region that will produce errors in postprocessing calculations. While a Gaussian beam was demonstrated here, it is straightforward to apply these same principles to all sources.

After proper TF and SF regions are chosen, a discrete function Q is constructed on a grid that identifies each cell as either TF or SF. Points filled with 0's indicate TF cells while points filled with 1's indicate SF cells. The same techniques discussed in Chapter 1 can be used to build the array Q . From there, the SF masking matrix \mathbf{Q} is constructed by reshaping the discrete Q function into a one-dimensional array, declaring it as sparse, and then placing it along the diagonal of a matrix. The MATLAB code to perform this diagonalization is

```
Q = diag(sparse(Q(:)));
```

The order of the functions in the above code is critical. If the order of `diag()` and `sparse()` is swapped, there will be a moment when a very large full matrix is formed in memory. This will be very slow and require a lot of memory if it does not crash the program entirely.

8.2.4 Compensating for Numerical Dispersion

Numerical dispersion becomes a more serious problem when there are multiple TF/SF interfaces, see Figure 8.3(c), or when there is a TF/SF interface extending along the direction of the source, see Figure 8.3(e). Recall from Chapter 4 that numerical dispersion causes a simulated wave to propagate slower than an analytical wave. This makes the simulated source and the source field become out of phase over distance, even when the source field is a rigorous solution to Maxwell's equations [5]. Additional and undesired waves will be launched from the TF/SF interface where there exists a misalignment between the simulated source and the analytical source wave. The problem is particularly severe when the source overlaps TF/SF interfaces over large distances. These problems were illustrated in Figure 8.3 in the context of a Gaussian beam source.

When such nonideal TF/SF regions are necessary, measures must be taken to minimize or compensate for the numerical dispersion. It is possible to perfectly cancel numerical dispersion for waves in one specific direction. This is accomplished by lowering the overall refractive index across the entire grid by the correct amount to make numerical waves propagate exactly at the same speed as physical waves. The direction to cancel numerical dispersion is usually chosen to be the direction of the source.

Suppose numerical dispersion is canceled by multiplying the background refractive index n by the constant ψ . In this case, the numerical wave vector would exactly match the analytical wave vector, and the numerical dispersion relation for a two-dimensional Yee grid becomes

$$\left[\frac{\omega(\psi n)}{c_0} \right]^2 = \left[\frac{2}{\Delta x} \sin\left(\frac{k_x \Delta x}{2}\right) \right]^2 + \left[\frac{2}{\Delta y} \sin\left(\frac{k_y \Delta y}{2}\right) \right]^2 \quad (8.40)$$

Solving this expression for the factor ψ and recognizing that $k_0 = \omega/c_0$ gives

$$\psi = \frac{1}{k_0 n} \sqrt{\left[\frac{2}{\Delta x} \sin\left(\frac{k_x \Delta x}{2}\right) \right]^2 + \left[\frac{2}{\Delta y} \sin\left(\frac{k_y \Delta y}{2}\right) \right]^2} \quad (8.41)$$

It is possible to perfectly cancel numerical dispersion for a wave propagating in the direction of $\vec{k} = k_x \hat{a}_x + k_y \hat{a}_y$ in two steps. First, the factor ψ is calculated using (8.41). In MATLAB, the variable ψ will be given the name `psi`. Second, after building the materials onto the grid, the relative permittivity `ER2` and relative permeability `UR2` arrays are each adjusted by the factor `psi` across the entire grid according to

$$\text{ER2} = \text{psi} * \text{ER2} \quad (8.42)$$

$$\text{UR2} = \text{psi} * \text{UR2} \quad (8.43)$$

Observe that both the permittivity ϵ_r and permeability μ_r are adjusted by the same factor ψ . This happens because it is actually the refractive index n that must be adjusted by the factor ψ and $\psi n = \sqrt{\psi \mu_r \psi \epsilon_r}$. If the $2\times$ grid is not used, the individual tensor elements must all be scaled by the factor ψ according to

$$\epsilon_{rxx} = \psi \epsilon_{rxx} \quad (8.44)$$

$$\epsilon_{ryy} = \psi \epsilon_{ryy} \quad (8.45)$$

$$\epsilon_{rzz} = \psi \epsilon_{rzz} \quad (8.46)$$

$$\mu_{rxx} = \psi \mu_{rxx} \quad (8.47)$$

$$\mu_{ryy} = \psi \mu_{ryy} \quad (8.48)$$

$$\mu_{rzz} = \psi \mu_{rzz} \quad (8.49)$$

Figure 8.4 illustrates the utility of compensating for numerical dispersion. Both simulations in this figure are identical, other than one compensates for numerical

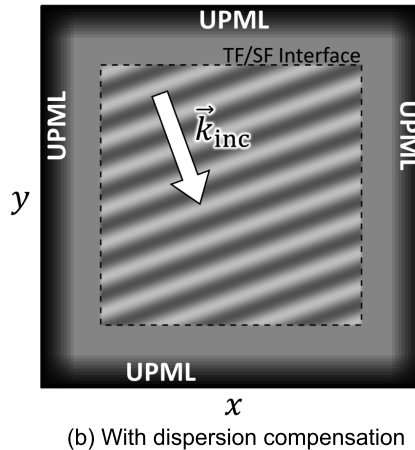
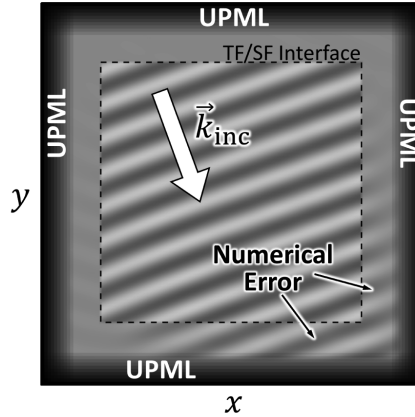


Figure 8.4 Illustration of compensating for numerical dispersion with a plane wave source at $NRES = 10$. (a) No dispersion compensation leads to numerical error at the far side of the TF/SF interface. (b) Compensating for numerical dispersion eliminates the numerical error of the source at the TF/SF interface.

dispersion while the other does not. Each simulation launches a plane wave through a vacuum with a square TF region at the center of the simulation. The grid resolution was chosen to be $N_{RES}=10$ to ensure numerical dispersion is easily observed. Figure 8.4(a) shows the field calculated by the simulation when numerical dispersion is present. Observe the wave in the SF region in the lower-right part of the simulation. No wave should be observed here because no device is present to scatter waves. This wave is the numerical error that arises because the simulated wave and analytical source wave become out of phase. The TF/SF corrections made to the finite-difference equations cannot cancel the waves in the far SF region. The numerical error is the most serious at the points on the far side of the grid where the phase difference is most severe. The numerical error wave appearing in the SF region should not be there and so it will lead to more errors if the fields in the SF region are postprocessed to calculate transmission, scattering pattern, or something else from the simulation. Figure 8.4(b) shows the field calculated by the same simulation but numerical dispersion has been compensated. The numerical error has been completely eliminated.

Even though this technique only perfectly cancels numerical dispersion for a single direction, it still reduces numerical dispersion in all directions. This is one argument why it is best to make the grid resolution parameters dx and dy nearly equal to each other. If no single direction is the most important, consider compensating for a wave propagating at an angle of 22.5° off of the x -axis. This provides a good compromise between the extremes of numerical dispersion that tend to be maximum in the x - and y -directions and minimum in the diagonal directions.

8.3 Calculating Reflection and Transmission for Periodic Structures

Section 2.9 from Chapter 2 discussed how reflection and transmission from diffraction gratings are handled in the context of diffraction orders. All periodic structures in FDFD can be handled as a diffraction grating to calculate reflection and transmission. In the discussion from Chapter 2, the surface of the grating was in the xy plane, and the diffraction orders carried power away from the device in the $+z$ - and $-z$ -directions. This was illustrated in Figures 2.6 and 2.7. In the coordinate system chosen for two-dimensional FDFD simulations, the surface of the diffraction grating will be in the xz plane, and the diffraction orders will carry power away from the grating in the $+y$ - and $-y$ -directions. For this reason, the equations derived for calculating diffraction efficiency, reflectance, and transmittance require modification in order to be consistent with the new coordinates.

The source wave vector \vec{k}_{inc} incident at angle θ_{inc} lies in the xy plane, so $k_{z,inc} = 0$ and the $k_{x,inc}$ and $k_{y,inc}$ components are calculated as

$$k_{x,inc} = k_0 n_{ref} \sin \theta_{inc} \quad (8.50)$$

$$k_{y,inc} = k_0 n_{ref} \cos \theta_{inc} \quad (8.51)$$

In this case, the incident wave encounters a diffraction grating that leads to an infinite expansion for the x components of the wave vectors associated with the

diffraction orders. Boundary conditions require that this expansion is the same for both reflected and transmitted diffraction orders so $k_{x,\text{ref}}(m) = k_{x,\text{trn}}(m) = k_x(m)$. The expansion is calculated as

$$k_x(m) = k_{x,\text{inc}} - m \frac{2\pi}{\Lambda} \quad (8.52)$$

where Λ is the period of the grating and m is the diffraction order. The longitudinal components of the wave vectors of the diffraction orders may be different on the reflection side and transmission side because the medium can be different on each side. For this reason, they must be calculated separately using the dispersion relation written for both sides.

$$k_{y,\text{ref}}(m) = -\sqrt{(k_0 n_{\text{ref}})^2 - k_x^2(m)} \quad (8.53)$$

$$k_{y,\text{trn}}(m) = \sqrt{(k_0 n_{\text{trn}})^2 - k_x^2(m)} \quad (8.54)$$

When the simulation finishes, the field is analyzed to calculate the complex amplitudes of the diffraction orders in three steps. First, the fields are extracted from the cross section of the grid at both the reflection and transmission planes. For the E mode (TM polarization) these are $E_{z,\text{ref}}(x)$ and $E_{z,\text{trn}}(x)$. Second, the phase tilt due to an oblique angle of incidence is removed. Recall that Bloch modes in periodic structures have the form of an amplitude function times a phase function. Since the source is a plane wave with unit amplitude, any cross section of the source gives the phase function of the Bloch mode. To isolate the amplitude portion of the Bloch mode in both the reflection and transmission planes, the fields extracted from the grid are divided by the cross section of the source $\exp(-jk_{x,\text{inc}}x)$.

$$\begin{aligned} a_{\text{ref}}(x) &= E_{z,\text{ref}}(x) \div \exp(-jk_{x,\text{inc}}x) \\ a_{\text{trn}}(x) &= E_{z,\text{trn}}(x) \div \exp(-jk_{x,\text{inc}}x) \end{aligned} \quad (8.55)$$

Third, the amplitudes of the diffraction orders are calculated from the discrete Fourier transform of the amplitude functions calculated in (8.55). Using a fast Fourier transform (FFT) algorithm, these are calculated as

$$\begin{aligned} E_{0,\text{ref}}(m) &= \text{FFT}[a_{\text{ref}}(x)] \\ E_{0,\text{trn}}(m) &= \text{FFT}[a_{\text{trn}}(x)] \end{aligned} \quad (8.56)$$

Give some extra thought any time an FFT is used because most FFT algorithms must be scaled and shifted to produce correct Fourier values. Now that the amplitudes of the diffraction orders are known, the diffraction efficiencies are calculated from equations derived in Chapter 2, but with the coordinates chosen for two-dimensional FDFD.

$$R_{\text{DE}}(m) = \frac{|E_{0,\text{ref}}(m)|^2}{|E_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{y,\text{ref}}(m)}{k_{y,\text{inc}}} \right] \quad (8.57)$$

$$T_{\text{DE}}(m) = \frac{|E_{0,\text{trn}}(m)|^2}{|E_{0,\text{inc}}|^2} \text{Re} \left[\frac{\mu_{\text{r,ref}} k_{y,\text{trn}}(m)}{\mu_{\text{r,trn}} k_{y,\text{inc}}} \right] \quad (8.58)$$

For the H mode (TE polarization), the complex amplitudes of the diffraction orders on the reflection and transmission sides are written as $\tilde{H}_{0,\text{ref}}(m)$ and $\tilde{H}_{0,\text{trn}}(m)$, respectively. In Chapter 2, these were vector quantities, but for the H mode in FDFD the magnetic field has only a z component. From these, the diffraction efficiencies are calculated as

$$R_{\text{DE}}(m) = \frac{|\tilde{H}_{0,\text{ref}}(m)|^2}{|\tilde{H}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{y,\text{ref}}(m)}{k_{y,\text{inc}}} \right] \quad (8.59)$$

$$T_{\text{DE}}(m) = \frac{|\tilde{H}_{0,\text{trn}}(m)|^2}{|\tilde{H}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\varepsilon_{\text{r,ref}} k_{y,\text{trn}}(m)}{\varepsilon_{\text{r,trn}} k_{y,\text{inc}}} \right] \quad (8.60)$$

Observe the negative sign inserted into (8.53) that is canceled by the negative signs inserted into (8.57) and (8.59). It is a habit for many computational scientists to avoid doing any calculations that are just undone later in the code. For this reason, some people drop all of these negative signs in their FDFD codes so that $k_{y,\text{ref}}$ is always a positive number.

8.4 Implementation of the FDFD Method for Scattering Analysis

A block diagram for a basic FDFD method for scattering analysis is shown in Figure 8.5. It is composed of six major steps that are each composed of a sequence of smaller steps. First, the simulation begins by initializing MATLAB, defining the units to be used in the simulation and defining any constants that may be needed later in the code. The second major step is the all-important dashboard where all of the parameters are defined that control all aspects of the simulation. This includes parameters that describe the source, the device dimensions and material properties, and parameters that control numerical aspects such as grid resolution and perfectly matched layer (PML) size.

The real work begins in the third major step where the grid is constructed. All the following steps will operate on this grid. A first guess is made at the grid resolution parameters Δx and Δy by resolving the minimum wavelength λ_{min} by N points. The minimum wavelength is the free space wavelength of the simulation λ_0 divided by the maximum refractive index n_{max} found anywhere on the grid.

$$\Delta x = \Delta y = \frac{\lambda_0}{n_{\text{max}} N} \quad (8.61)$$

This calculation of grid resolution ignores the dimensions of a device. Some dimensions are very critical to model accurately on the grid, such as the period of

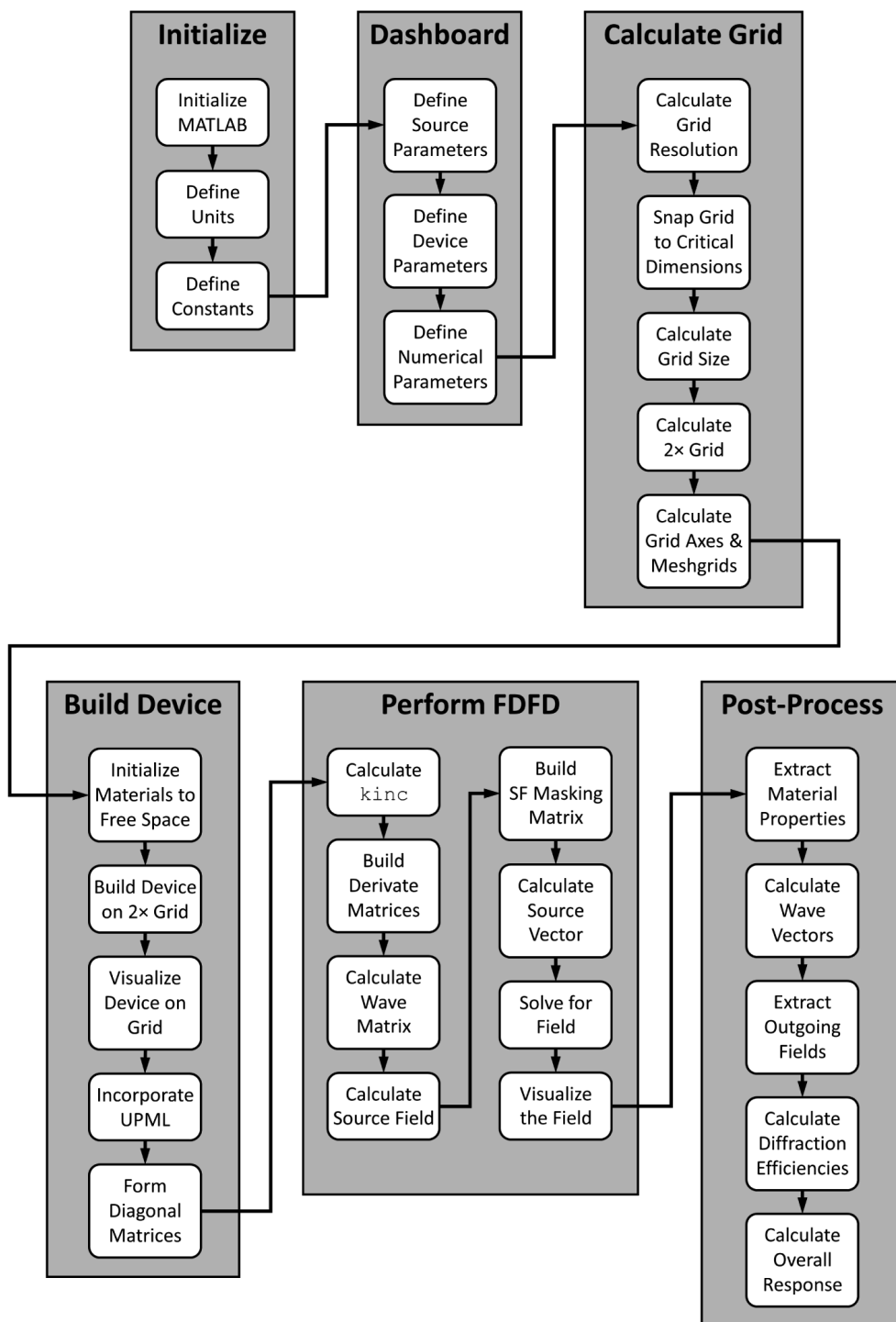


Figure 8.5 Block diagram of FDFD for scattering analysis.

a periodic structure and the groove depth of a diffraction grating. The next step in FDFD that greatly improves accuracy and convergence rate is to adjust the grid resolution parameters Δx and Δy so that critical dimensions are represented exactly by an integer number of grid cells. From here, the grid resolution parameters are fixed so the next thing is to determine the size of the grid in terms of width S_x and height S_y . The physical size is determined first with enough space included to fit the device being simulated, absorbing boundaries, and any extra space around the device that may be needed. The total number of cells on the grid, N_x and N_y , is calculated by dividing the physical size by the grid resolution. At this point, the grid has been calculated, and from these parameters, the $2\times$ grid parameters are calculated. The last step is to calculate the grid axes and meshgrid parameters if they are needed.

With the grid calculated, the next major step is to build the device onto the grid. For many devices, especially those with curved geometries, it is advantageous to build them using the $2\times$ grid technique. Regardless, the process starts by initializing both the relative permittivity $\epsilon_r(x,y)$ and the relative permeability $\mu_r(x,y)$ to vacuum. The device is constructed onto the $2\times$ grid and visualized. It is an excellent practice to visualize as much as possible in the code for troubleshooting, verification, and learning the algorithm better. At this point, the uniaxial PML (UPML) is incorporated while the device is still on the $2\times$ grid. It is very convenient to build the UPML onto the $2\times$ grid. The function `addupml2d()` is written to incorporate the UPML and extract the material arrays for the standard Yee grid. The last step is to form the diagonal material matrices from the material arrays.

With the device constructed onto the grid, finally, it is time to implement the FDFD method. It starts by calculating the wave vector \vec{k}_{inc} of the incident wave. This is needed for the periodic boundary conditions (PBCs) in the derivative matrices as well as in postprocessing to calculate reflectance and transmittance. After the incident wave vector, the derivative matrices are constructed by calling the function `yeder2d()` that was described in Chapter 4. Given the materials matrices and derivative matrices, the wave matrix A is calculated using (8.21) for the E mode (TM polarization) or using (8.22) for the H mode (TE polarization). Next, the source field f_{src} is calculated, followed by the SF masking matrix Q , and then finally the source vector b using the QAAQ equation in (8.32). All of the steps so far in the FDFD algorithm run relatively quickly. It is the next single line of code that is the slowest computation where the field is calculated by solving $f=A\backslash b$. When this is finished, it is always a good practice to visualize the field. Correct fields are generally smooth and continuous. Any sort of noise or spikes in the solution is an indication that something may be wrong. Keep in mind that the field may be discontinuous at the TF/SF interface because the source is not present in the SF region but is present in the TF region.

At this point, the FDFD algorithm is finished and the program can move on to postprocessing the simulation results. A common postprocessing flow is analyzing transmission and reflection from a periodic structure so that is what is shown in the block diagram. Here, the simulated field is analyzed to calculate the diffraction efficiencies of all of the diffraction orders as well as the overall reflectance and transmittance. The first step is to extract the material properties where the reflected and transmitted fields are to be analyzed on the grid. Second, the wave vector expansion for the diffraction orders is calculated, including both the tangential

and longitudinal components of the wave vectors. Third, the cross section of the field on the reflected side of the device is extracted from the grid above the TF/SF interface inside of the SF region. The cross section of the field on the transmitted side of the device is extracted from the grid just above the bottom UPML. Fourth, from all of the information calculated so far, the diffraction efficiency of each of the diffraction orders is calculated. Last, the overall reflectance is calculated by adding all of the diffraction efficiencies of the reflected diffraction orders. The overall transmittance is calculated by adding all of the diffraction efficiencies of the transmitted diffraction orders.

8.4.1 Standard Sequence of Simulations for a Newly Written FDFD Code

Before any new devices are simulated, there should be a standard sequence of simulations performed to help identify and troubleshoot problems with a newly written code. The recommended sequence is shown in Figure 8.6 where PBCs were used at the x -axis boundaries to simulate a periodic structure. The first simulation is a complete vacuum with the TF/SF interface positioned at the center of the grid vertically launching a wave at normal incidence, as depicted in Figure 8.6(a). The TF/SF interface is centered to better see if there are any backward waves or reflections. These would be more difficult to detect if the TF/SF interface were tight against the top PML. If the FDFD code is working correctly, the simulation will calculate 100% transmission, 0% reflection, and no wave should be visible in the SF region above

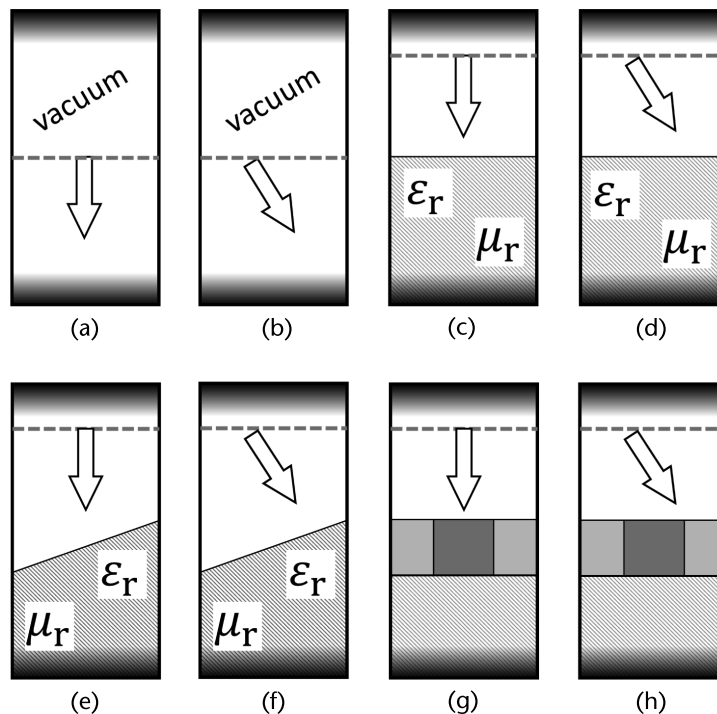


Figure 8.6 Standard sequence of simulations for a newly written FDFD code.

the TF/SF interface. The specific values for reflectance and transmittance should be within a fraction of a percent. At $N_{RES}=10$, a working FDFD simulation may calculate numbers like $R = 5.910^{-5}$ and $T = 0.998$. If anything wrong is detected when simulating transmission through a vacuum, the mistake will be easier to find and better-educated guesses for values of intermediate parameters can be made with such a simple simulation.

Next, an angle of incidence is incorporated, perhaps $\theta_{inc} = 30^\circ$, as depicted in Figure 8.6(b). This is mostly to test that the PBCs are working correctly. If everything is correct, the field should look like what is shown in Figure 8.7(a). The source wave is visible in the TF region below the TF/SF interface, and no wave is visible in the SF region above the TF/SF interface. This simulation should produce 0% reflectance and 100% transmittance with a numerical error well under 1%. When there is a mistake in the code, the field may look like what is shown in Figure 8.7(b). The field below the TF/SF interface does not resemble a pure plane wave, indicating that at least one additional wave is present that is interfering with the source wave. A wave is also visible above the TF/SF interface that is likely the same wave that is interfering with the source wave below the TF/SF interface. Since a reflected wave is observed in the TF region, it is most likely the bottom PML that is reflecting and not necessarily a problem with the TF/SF source. For this case, the reflectance was calculated to be 20% and transmittance calculated to be 195% at $N_{RES}=20$, clearly an incorrect simulation result. Keep an open mind when troubleshooting because backward waves could be due to a problem with the TF/SF source, a problem with the PML, a problem with the boundary conditions, or something else entirely. Most errors at this stage all produce fields that look much like what is shown in Figure 8.7(b).

After the FDFD code can simulate vacuum correctly, the next thing is to move the TF/SF interface to around two cells below the top PML and then build a single material interface onto the grid. This is illustrated in Figure 8.6(c). In this case, the Fresnel equations discussed in Chapter 2 can be used to verify the simulation results are correct. First, the bottom half of the grid can be filled with $\epsilon_r = 9.0$. For normal incidence, this should produce exactly 25% reflectance and 75% transmittance. The most common mistake here is not calculating the diffraction efficiency of the transmitted diffraction orders correctly because that equation contains extra terms that are easily missed. Next, swap the permittivity and permeability such that $\epsilon_r = 1.0$ and $\mu_r = 9.0$ and ensure the simulation still gives 25% reflectance and 75% transmittance. Next, set $\epsilon_r = \mu_r = 3.0$ in the bottom half of the grid and verify reflectance is near 0% and transmittance is near 100%. This case produces no reflections because the impedance is constant throughout the grid. Next, incorporate an angle of incidence and repeat various combinations of ϵ_r and μ_r , as illustrated in Figure 8.6(d). Using the Fresnel equations, an infinite number of simple simulations are possible. Choose a few simple ones and then move on.

Next, it is best to simulate an asymmetric diffraction grating like the one that will be simulated in Section 8.4.2. An asymmetric diffraction grating is illustrated in Figure 8.6(e) and is an excellent device to verify that the diffraction orders are being handled correctly in the FDFD code. The same asymmetric diffraction grating is simulated again, but with an angle of incidence incorporated, as illustrated in Figure 8.6(f). The last simulation recommended for testing new codes is a wavelength (or frequency) sweep of a guided-mode resonance filter (GMRF), as illustrated in Figure

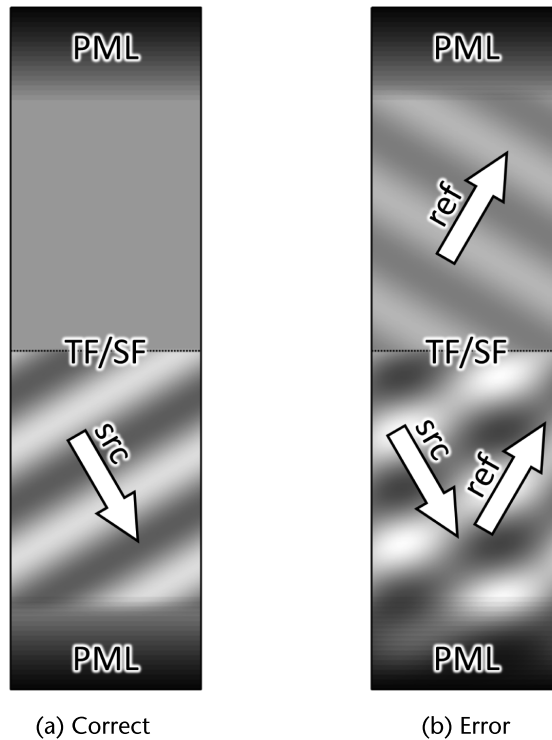


Figure 8.7 (a) Correct simulation where the source wave in the TF region is visible and no backward wave is observed. (b) Example where bottom UPML is not working correctly and reflections are observed.

8.6(g, h). GMRFs are extremely sensitive devices. If anything is wrong in the code, a GMRF will tend to amplify the problem so that more types of mistakes can be identified and corrected. This type of simulation is a form of parameter sweep that will be discussed in Chapter 9.

The first device that will be discussed in detail in the following sections is an asymmetric diffraction grating that can serve as a device to use for benchmarking. The second device that will be simulated is a self-collimating photonic crystal illuminated by a Gaussian beam source. This simulation will illustrate how information from photonic band calculations can be used in a scattering simulation. The last simulation will be an OIC. It will make multiple uses of slab waveguide analysis to reduce the OIC to two dimensions, to calculate the source, and finally to analyze reflection and transmission from the circuit.

8.4.2 FDFD Analysis of a Sawtooth Diffraction Grating

Figure 8.8 shows the sawtooth diffraction grating that will be used to benchmark the FDFD code. The asymmetric nature of the sawtooth is excellent for verifying that the diffraction efficiency calculations are correct. The period of the grating is Λ and the depth of the grooves is d . To simplify the simulation, the diffraction grating will be assumed to be infinitely periodic in the x -direction and infinitely

extruded in the z -direction. In addition, the space above and below the diffraction grating will be assumed to be semi-infinite. This means the superstrate material ϵ_{r1} will extend off to infinity above the grating and the substrate material ϵ_{r2} will extend off to infinity below the grating. This example will have air above the grating so $\epsilon_{r1} = 1.0$. The dielectric constant of the diffraction grating was chosen to be $\epsilon_{r2} = 9.0$. It is good to benchmark using devices with higher permittivity to better exercise the code. The period of the grating was chosen to be 1.5 cm so that a few diffraction orders will exist above and below the diffraction grating when simulated at a frequency of 30 GHz. When designing this problem, it was observed there was a more even distribution of power among the diffraction orders when the grating depth was set to 1.0 cm. This value was chosen to make the design more suitable for benchmarking. Changing the grating depth away from this value will change the diffraction efficiencies of the diffraction orders.

Figure 8.8 (Right) shows the grid strategy used to simulate most periodic structures in FDFD. Only a single unit cell in the x -direction has to be stored in memory and simulated. PBCs at the x -axis boundaries make the grating appear to be infinitely periodic in the x -direction. The device itself is built onto the grid so that the grating resides close to the center of the grid vertically. UPMLs are placed at the y -axis boundaries to absorb outgoing waves. Spacer regions are included above and below the grating to ensure that any evanescent fields decay very close to zero before touching the UPMLs. Evanescent fields are fields that extend outside of the device, but are not propagating waves and stay confined to the surface of the device. They are sort of a temporary place for electromagnetic power before coupling into

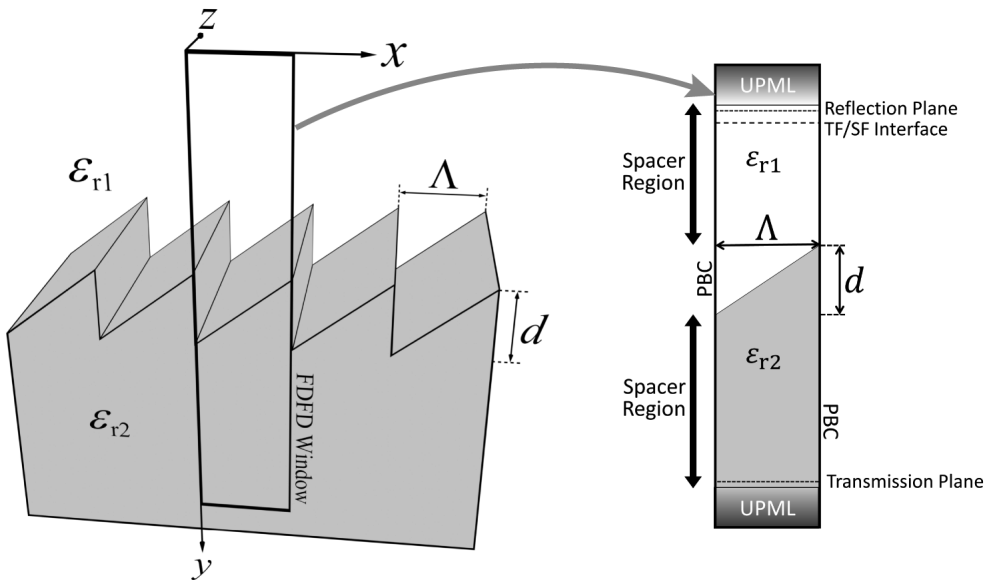


Figure 8.8 (Left) Perspective view of a sawtooth diffraction grating. (Right) FDFD grid strategy for the same sawtooth diffraction grating.

a diffraction order and escaping the device. The top UPML and a few rows of cells below the top UPML have been made the SF. The rest of the cells on the grid are TF.

The MATLAB code to simulate this diffraction grating using FDFD can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter8_sawtooth.m`. The header for the program extends from lines 3 to 29 where MATLAB is initialized, units are defined, and electromagnetic constants are defined in terms of the specified units. Lines 4 to 6 initialize MATLAB by closing all of the figure windows, clearing the command window, and then clearing all variables from memory. In lines 9 to 23, the units for the simulation are defined. Here `meters` and `seconds` were set to 1. This is typical for microwave simulations, but for optical simulations consider setting `micrometers` to 1. It is a good practice to have all numbers in a simulation as close to the numerical value of 1 as possible. Proper selection of the scale of the units is a good way to do this. Last, lines 26 to 29 calculate various electromagnetic constants that may be used in the simulation including free space permittivity ϵ_0 , free space permeability μ_0 , free space impedance N_0 , and the speed of light in vacuum c_0 . This is usually the author's standard header for all simulations so the units and constants are all defined even if they end up not being used.

The code extending from lines 31 to 54 is called the dashboard. This is where all aspects of the program are controlled. All hard-coded numbers go in this section and no work is to be performed here. After the dashboard, never hard-code any numbers. Perform all operations relative to variables defined in the dashboard. Lines 35 to 40 define everything that is needed about the plane wave source. This includes frequency f_0 , angle of incidence θ , and which electromagnetic mode (E or H) to simulate stored in the variable `MODE`. Consider taking the opportunity to calculate the vacuum wavelength λ_{m0} and free space wavenumber k_0 at this stage because sometimes other numbers in the dashboard are calculated using these parameters. Lines 42 to 46 define everything that is needed about the diffraction grating including the period L , depth d , relative permittivity of the superstrate ϵ_{r1} , and relative permittivity of the substrate ϵ_{r2} . It is often best to sketch the device on paper and define the variables on that sketch much like what is shown in Figure 8.8. Lines 48 to 53 define parameters associated with the numerical aspects of FDFD. `NRES` defines the resolution of the grid as the number of grid cells per minimum wavelength. `SPACER` is an array of two numbers that define how much space should be included above and below the diffraction grating. For ordinary scattering simulations, usually, a quarter to a half wavelength is sufficient. Some devices can exhibit large evanescent fields, such as resonant devices or devices operating near a diffraction order cutoff or a guided mode cutoff. For these devices, the spacer regions should be made larger so that the evanescent fields do not extend into the UPML regions. `NPML` is an array of two numbers that define the numerical size of the UPML regions at the top and bottom of the grid, respectively. `ermax` is an intermediate parameter that determines the maximum relative permittivity that will be placed onto the grid. `nmax` is calculated from `ermax` and is the maximum refractive index that will be placed onto the grid. This is needed to determine the shortest wavelength in the simulation from which the grid resolution will be calculated. If there were any relative permeability, there would be a `urmax` parameter defined and this would be used in addition to `ermax`.

for the calculation of n_{\max} . After the dashboard, no more hard-coded numbers should ever be used!

At this point, the real work for FDFD can begin. The first major task is to calculate a grid that is optimized for analyzing the device defined in the dashboard. This happens from lines 55 to 88. Lines 60 and 61 make the first guess at grid resolution. It sets the grid resolution parameters dx and dy so that the smallest wavelength in the simulation is resolved by N_{RES} number of points. Lines 64 to 68 adjust dx and dy so that the critical dimensions of the device are resolved exactly by an integer number of grid cells. For diffraction gratings, the period L is almost always chosen to be the critical dimension in the x -direction. The calculation L/dx is the number of cells it would take to resolve the period L on the current grid. Since the period was not considered in the calculation of dx , L/dx is most likely not an integer. This is rounded up to the nearest integer to calculate n_x . At this point, n_x is interpreted as the number of cells it is desired to represent the period exactly. To make this happen, the resolution parameter dx is recalculated as L/n_x . This same procedure is done for the critical dimension in the y -direction. For the diffraction grating, the grating depth d was chosen as the critical dimension. Only a single critical dimension can be chosen for each direction for a uniform grid. Now the grid resolution is finalized and will not be adjusted again. Lines 70 to 77 calculate both the physical size and numerical size of the grid that will contain the device and simulated fields. The physical size in the x -direction is S_x while the physical size in the y -direction is S_y . The typical calculation steps are: (1) define physical size to include the spacer regions, (2) calculate the total number of cells to encompass this region plus the UPML regions, and (3) recalculate the physical size so that it includes the UPML regions. At this point, the Yee grid is calculated. The next step is to calculate the $2\times$ grid, which describes the same physical space as the Yee grid, but with twice as many points. N_{x2} and N_{y2} describe how many points the $2\times$ grid contains in the x - and y -directions, respectively. $dx2$ and $dy2$ are the grid resolution parameters for the $2\times$ grid. Then, the grid axes x_a and y_a for the Yee grid are calculated along with the grid axes x_{a2} and y_{a2} for the $2\times$ grid. These are one-dimensional arrays that contain the position of each cell on the grid. Last, the meshgrid parameters X and Y for the Yee grid are calculated. If needed, the meshgrid parameters $X2$ and $Y2$ for the $2\times$ grid can also be calculated here.

After an optimized grid is calculated, lines 90 to 127 build the device onto this grid, add the UPML and form the diagonal materials matrices. Lines 94 to 96 initialize the relative permittivity $ER2$ and relative permeability $UR2$ on the $2\times$ grid. The array $ER2$ is set to all values of $er1$ throughout the entire $2\times$ grid. The array $UR2$ is set to all values of 1. Lines 98 to 106 build the asymmetric sawtooth geometry onto the $2\times$ grid. The variable $ny1$ represents the array index in the y -direction of the top of the diffraction grating. It is calculated to be below the top UPML, below the top spacer region, and one extra cell vertically in case both the UPML and spacer region are defined to be zero. Note that the size of the top UPML is multiplied by two on line 99. That is because the array N_{PML} contains the size of the UPML on the Yee grid, but the device is being constructed on the $2\times$ grid. The variable $ny2$ represents the array index in the y -direction of the bottom of the diffraction grating.

It is calculated as the array position $ny1$, plus the number of cells the grating tooth is tall, minus one cell. After this, a loop is performed that iterates ny from the top $ny1$ to the bottom $ny2$ of the grating tooth. Line 102 calculates the variable f to go from 0 to 1 as the loop progresses from the top to the bottom of the grating tooth. However, it is best to not have f be exactly equal to 0 or 1 because that would be an unused layer. Given f , nx is calculated on line 103 to be the number of cells wide the grating tooth is at the vertical position defined by ny . Line 104 sets the points in $ER2$ to a value $er2$ that resides inside of the grating tooth. After the loop finishes, line 108 assigns all points in $ER2$ below the grating tooth to be $er2$, representing the semi-infinite substrate. At this point, the device is constructed onto the $2\times$ grid and it is visualized on lines 110 to 115. Lines 118 and 119 call the function `addupm12d()` that incorporates the UPML onto the grid and extracts the six Yee grid materials arrays ER_{xx} , ER_{yy} , ER_{zz} , UR_{xx} , UR_{yy} , and UR_{zz} . Last, lines 121 to 127 form the sparse diagonal materials matrices that will be used in the next section of code to build the wave matrix A .

It is finally time to perform FDFD! This starts by calculating the source wave vector. Line 134 calculates the refractive index in the semi-infinite region above the diffraction grating. Lines 135 and 136 calculate the x and y components of the source wave vector incident at angle θ . Line 137 assembles the wave vector components into a single vector. Lines 139 to 143 call the function `yeeDer2d()` to build the four derivative matrices. Line 142 defines PBCs at the x -axis boundaries and Dirichlet boundary conditions at the y -axis boundaries. If the UPML is functioning correctly, it should not matter what boundary conditions are used at the y -axis boundaries. Given this, Dirichlet is chosen to avoid incorporating complex numbers into the derivative matrices that can sometimes slow obtaining a solution. Observe on line 143, the grid resolution parameters in RES are multiplied by k_0 and the incident wave vector k_{inc} is divided by k_0 to build the derivative matrices with a normalized grid. Lines 144 to 150 calculate the wave matrix A . An `if` statement is used to build a different wave matrix depending on which electromagnetic mode was defined in the dashboard through the variable $MODE$. Line 153 calculates the source field in the array f_{src} . This is the source wave calculated throughout the entire grid as if the diffraction grating were not present. The source is given unit amplitude. Lines 155 to 159 build the SF masking matrix Q . Line 156 calculates ny to be the array index for the location in the y -direction of the TF/SF interface. Line 157 initializes the array Q to be all zeros, which makes Q identify all points on the grid as TF. Line 158 then sets all points in Q from the top of the grid down to ny to be all 1's. This defines the top portion of the grid above ny to be the SF region. Line 159 forms a sparse diagonal matrix Q which is the SF masking matrix. Line 162 calculates the source vector b using the QAAQ equation. Line 165 is the most intensive computational step in FDFD and is also the shortest and simplest line of code in the entire program. It is here that the matrix equation $Af = b$ is solved for the field f via backward matrix division. For the E mode, the field f represents the field component E_z while for the H mode the field f represents the field component H_z . It was given the name f to be generic and represent the field for both E and H modes. After solving, f is a column vector so line 166 reshapes it back to the two-dimensional Yee grid, which is N_x -by- N_y points. Lines 168 to 175 visualize the calculated field. It is always a good

practice to visualize the field because many problems can be identified and even diagnosed just by examining the field. Expect to see a discontinuity of the field at the TF/SF interface because the source is removed in the SF region.

With the field calculated, FDFD is finished and the program can move on to postprocessing. For periodic structures, it is often more meaningful to calculate reflection and transmission by analyzing the diffraction orders. A simpler way would be to simply integrate the Poynting vector across the grid, but this circumvents information from the diffraction orders that are very often quite meaningful. Lines 177 to 218 perform this postprocessing step and report the results. Lines 181 to 187 extract the material properties where the reflected and transmitted fields are being analyzed and calculate the refractive index in these locations. Lines 189 to 193 calculate the wave vector components of all of the diffraction orders. Line 190 calculates an array m that contains the integer numbers of the diffraction orders in the order that FDFD will calculate them. The limits were chosen to have N_x total number of integers and to place the zero-order (i.e., $m=0$) at the correct position for both odd and even values of N_x . Line 191 calculates an array k_x that contains the tangential components of the diffraction orders. These are the same for both reflected and transmitted diffraction orders so only a single array is calculated and used for both. The longitudinal components of the diffraction orders will be different in the reflected and transmitted regions if the materials in these regions are different. They are calculated separately on lines 192 and 193 as k_{yref} and k_{ytrn} using the dispersion relation for each medium. Line 196 extracts a cross section of the field f_{ref} at a location just under the top UPML, but still in the SF region, and removes the phase tilt due to an oblique angle of incidence. This location ensures that any fields present at that location are solely reflected from the device. Line 197 extracts a cross section of the field f_{trn} at a location just above the bottom UPML and removes the phase tilt. This location ensures that any field at this location is transmitted through the device. Lines 200 and 201 calculate the complex amplitudes of the diffraction orders. The array a_{ref} contains the complex amplitudes of the reflected diffraction orders and the array a_{trn} contains the complex amplitudes of the transmitted diffraction orders. These terms are complex because they describe both the amplitude and the phase of the diffraction orders. The phase tilt and the phase of the diffraction orders are different quantities. The phase tilt is the phase across the grid due to an oblique angle of incidence. The phase of a diffraction order is its overall phase relative to the source.

Given the complex amplitudes and the material properties where the diffraction orders are being analyzed, the diffraction efficiencies are calculated in the arrays RDE and TDE . Line 204 calculates the array RDE that contains the diffraction efficiencies of the reflected diffraction orders. This calculation is the same for both E and H modes. Lines 205 to 209 calculate the array TDE that stores the diffraction efficiencies of all the transmitted diffraction orders. The equation to calculate the diffraction efficiencies of the transmitted diffraction orders is different for E and H modes. An `if` statement is used to select the proper equation. In the end, the array TDE will contain the diffraction efficiencies of the transmitted diffraction orders. After the diffraction efficiencies are calculated, the overall reflectance REF is calculated on line 212 by adding all of the diffraction efficiencies of the reflected diffraction orders. The overall transmittance TRN is calculated on line 213 by adding all of the

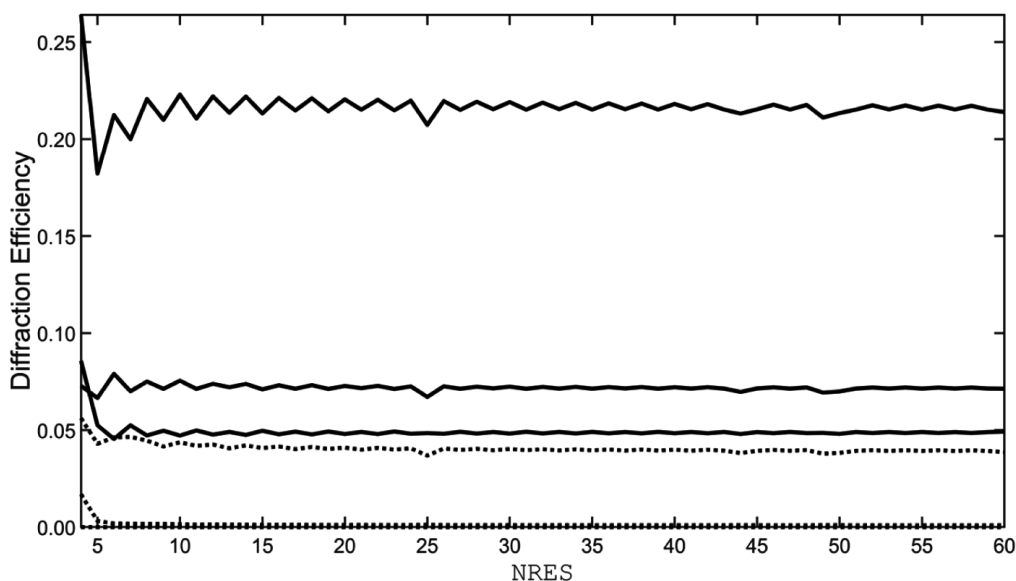


Figure 8.9 Convergence study for FDFD analysis of sawtooth diffraction grating.

diffraction efficiencies of the transmitted diffraction orders. Last, the reflectance and transmittance are added together to assess overall power conservation. If the simulation contains no loss or gain, $\text{REF} + \text{TRN}$ should equal 1. Due to numerical error, conservation is never exactly equal to 1 and is usually off of this value by a small fraction of 1%. Lines 217 and 218 display the diffraction efficiencies in a simple manner. Line 217 calculates the array indices where the diffraction efficiency is not zero. Line 218 displays a three-column array of the non-zero diffraction orders. The first column is the diffraction order number, the second column is the diffraction efficiency of the reflected diffraction orders, and the third column is the diffraction efficiency of the transmitted diffraction orders.

Even if the MATLAB program described above runs without error, the answer it reports cannot be trusted until a convergence study is performed. To do this, Figure 8.9 plots the diffraction efficiency of all the diffraction orders as a function of NRES. The lines are not labeled because it is just the overall trend that is of interest. Convergence is obtained around NRES equal to 20.

Table 8.1 summarizes the diffraction efficiencies at 30 GHz for both E and H modes for two different angles of incidence obtained at $\text{NRES} = 100$. This data can be used to benchmark and test your own FDFD code.

8.4.3 FDFD Analysis of a Self-Collimating Photonic Crystal

In Chapter 7, a lattice was analyzed using isofrequency contours (IFCs) to identify at what frequency the lattice self-collimates. Four cases where the lattice would self-collimate were identified. For this example, self-collimation will be simulated for the first band of the E mode. In this case, self-collimation was found to occur in the diagonal direction at a normalized frequency of $\omega_n = 0.217$. To simulate self-collimation occurring along the x -axis through a grid, the lattice will need to

Table 8.1 Diffraction Efficiency Calculations for Sawtooth Grating

m	$\theta_{\text{inc}} = 0^\circ$		$\theta_{\text{inc}} = 30^\circ$	
	$R_{\text{DE}}(\text{m})$	$T_{\text{DE}}(\text{m})$	$R_{\text{DE}}(\text{m})$	$T_{\text{DE}}(\text{m})$
-4		E: 0.09% H: 0.63%		
-3		E: 2.16% H: 0.36%		E: 0.41% H: 0.62%
-2		E: 36.85% H: 34.79%		E: 15.95% H: 11.19%
-1	E: 1.56% H: 0.31%	E: 3.90% H: 17.16%		E: 16.95% H: 21.61%
0	E: 3.16% H: 0.54%	E: 1.98% H: 5.89%	E: 1.82% H: 0.10%	E: 4.53% H: 4.86%
1	E: 8.12% H: 3.32%	E: 20.60% H: 0.72%	E: 8.99% H: 3.93%	E: 1.93% H: 7.18%
2		E: 4.17% H: 24.53%	E: 11.49% H: 16.39%	E: 3.29% H: 4.92%
3		E: 12.30% H: 3.62%		E: 14.15% H: 14.59%
4		E: 5.14% H: 8.15%		E: 19.02% H: 12.85%
5				E: 1.48% H: 1.78%
Overall	E: $R = 12.85\%$ H: $R = 4.17\%$	E: $T = 87.18\%$ H: $T = 95.85\%$	E: $R = 22.30\%$ H: $R = 20.41\%$	E: $T = 77.72\%$ H: $T = 79.61\%$
Conservation	E: $R + T = 100.03\%$ H: $R + T = 100.02\%$		E: $R + T = 100.02\%$ H: $R + T = 100.02\%$	

be rotated about the z -axis by 45° . The concept for the simulation is illustrated in Figure 8.10. The left part of this figure shows the lattice that will be simulated. It will be illuminated by a diverging Gaussian beam incident at an angle of 20° . This angle was chosen to demonstrate that self-collimation will occur in the x -direction regardless of the angle of incidence. Self-collimation for this lattice will fail if a large enough angle of incidence is chosen. Observe the primitive unit cell simulated in Chapter 7 is at a 45° angle relative to the x - and y -axes in this figure. To simplify constructing this lattice on the Yee grid, an alternative unit cell is identified that aligns with the x - and y -axes. The alternative unit cell has lattice spacing of $b = a\sqrt{2}$, where a is the lattice spacing of the original unit cell analyzed in Chapter 7. Recall from Chapter 7 that $\epsilon_{r1} = 1.0$, $\epsilon_{r2} = 12.0$, and $r = 0.48a$.

The grid strategy that will be used to simulate the photonic crystal is shown in Figure 8.11. A lattice of approximately $60a \times 40a$ will be centered in a grid. No spacer regions will be used above and below the lattice because all waves should be confined vertically to the center of the lattice. Spacer regions of size $6\lambda_0$ will be used on the left and right sides of the lattice to visualize the source and wave exiting

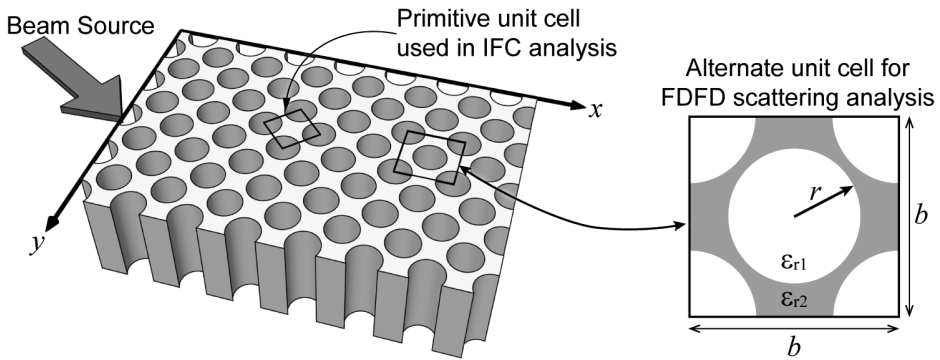


Figure 8.10 (Left) Self-collimating lattice to be simulated by FDFD. (Right) Alternative unit cell used to build the lattice in FDFD.

the lattice. A diverging Gaussian beam will be launched at a 20° angle of incidence and illuminate the center of the left face of the lattice. The TF/SF technique will be used to launch the source and the TF/SF interface will span the perimeter of the grid staying just outside of the UPMLs. It is not possible to use just a single TF/SF interface at the left side of the grid because the source beam is wide enough that it would overlap the top or bottom UPML. The TF/SF interface where sources are injected cannot cross a UPML because the source functions do not account for decay in the UPML. The sources must have zero amplitude where the TF/SF interface crosses a UPML. To avoid the source overlapping the UPML in this simulation, the TF/SF interface was made to form a square around the entire grid. The UPMLs will be 20 cells large and be placed along all four grid boundaries.

The MATLAB code to simulate the self-collimating photonic crystal can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter8_photoniccrystal.m`. Lines 1 to 9 are the header of the program where MATLAB is initialized and units for the simulation are defined. In this case, only the units of degrees are defined.

The dashboard for the simulation extends from lines 11 to 36. This is where all of the hard-coded numbers are defined that control all aspects of the simulation.

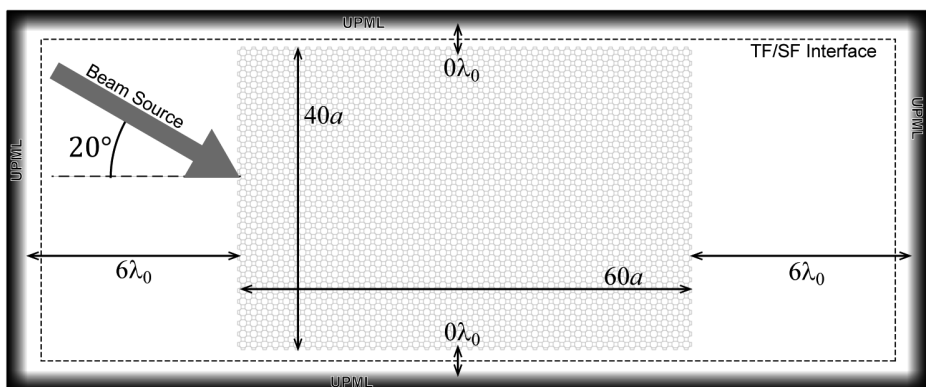


Figure 8.11 Grid strategy for FDFD analysis of a self-collimating photonic crystal.

Lines 15 to 19 define everything that is needed about the source. This includes the free space wavelength λ_{m0} , the free space wavenumber k_0 , the width of the Gaussian beam bw , and the angle of incidence θ . λ_{m0} is set to a value of 1. Since no length units were used, it may not be clear what this value of 1 represents. It represents whatever length scale you wish! That could be meters, micrometers, inches, or even Martian units. It is only necessary to ensure that any other length parameters be defined with the same units. See Chapter 2 for a discussion on scalability in electromagnetics. An extremely narrow beam was chosen to get a nicely diverging beam. The angle of incidence of the Gaussian beam was set to 20° . Lines 21 to 29 define everything that is needed about the photonic crystal. The normalized frequency wn is set equal to 0.217 as identified from the IFCs in Chapter 7. From this, the lattice constant a is calculated as the normalized frequency wn times the operating wavelength λ_{m0} . The radius of the hole is defined relative to the lattice constant as $0.48*a$. $er1$ is the dielectric constant of the fill medium and $er2$ is the dielectric constant of the photonic crystal. For this simulation, the fill medium is set to air, but it is best to define it this way in the dashboard instead of hard-coding the device to be air. The size of the overall lattice is L_x in the x -direction and L_y in the y -direction. These were set to produce a lattice roughly $60a \times 40a$. Last, lines 30 to 35 define all numerical parameters. The parameter $NRES$ controls the grid resolution and is the number of grid cells per minimum wavelength. The array `SPACER` defines four numbers that control the amount of physical space placed around the outside of the photonic crystal. The first number in `SPACER` is the space on the left, the second number is the space on the right, the third number is the space on the top, and the fourth number is the space on the bottom. In this example, all four dimensions are defined in terms of wavelength, which is often the best way to do this. The same ordering of numbers appears in the array `NPML` to define the size of the UPML at each boundary in terms of the number of grid points. A good nominal value for the size of a UPML is around 20 cells and that is what is used here for all boundaries. If this simulation was to be optimized for efficiency, consider reducing the size of the UPMLs at the top and bottom boundaries knowing that very little wave power will be incident on those boundaries. The last parameter is n_{max} , the maximum refractive index found anywhere on the grid. It is used later to determine the minimum wavelength in the simulation. Here, n_{max} is calculated through the intermediate parameter er_{max} because all of the material properties were defined in terms of relative permittivity.

After the dashboard, the real work can begin. Lines 37 to 71 perform the first major task of calculating a grid that is optimized for the simulation defined in the dashboard. Lines 42 and 43 calculate the first guess at the grid resolution parameters dx and dy that define the size of the cells on the Yee grid. This is calculated to be the minimum wavelength divided by $NRES$. The minimum wavelength is the free space wavelength λ_{m0} divided by the maximum refractive index n_{max} . Next, the grid resolution parameters are adjusted so as to resolve the unit cell with an exact integer number of cells. This will allow a single unit cell to be arrayed across the grid to build the photonic crystal. The unit cell used in Chapter 7 was the primitive unit cell with lattice spacing a . An alternative unit cell is defined here for a lattice that is rotated by 45° about the z -axis. This alternative unit cell has dimension b

calculated as $b = a\sqrt{2}$. The grid resolution parameter dx is adjusted in lines 47 to 48. nx is calculated to be the integer number of cells that should be used to represent the dimension b exactly. To ensure this, dx is recalculated as b/nx . The same procedure on lines 50 and 51 is used to adjust the grid resolution parameter dy . After this, the grid resolution is fixed and will not be changed again. Therefore, the next step is to calculate both the physical and numerical sizes of the grid. Sx is the physical size of the grid in the x -direction. It is calculated as the width of the photonic crystal Lx plus the spacer regions to the left and the right of the photonic crystal. Note the order of the terms on line 54 is written in the same order they reside on the grid. This practice helps to remember and visualize all of the dimensions that must be included. Nx is the numerical size of the grid, or the number of cells, in the x -direction. This is calculated as the physical size divided by the grid resolution Sx/dx , rounding this ratio up to the next largest integer, and adding the UPML regions on the left and right sides of the grid. The order of the terms on line 55 reflects the order they appear on the grid as was done when calculating Sx . Since the UPML regions were not considered in the original calculation of Sx , Sx is recalculated as $Nx*dx$ for consistency and to be the true physical size of the grid. Lines 58 to 60 repeat the same steps to calculate Sy and Ny to represent the size of the grid in the y -direction. At this point, an optimized grid is calculated. Lines 62 to 64 go on to calculate the $2\times$ grid parameters $Nx2$, $Ny2$, $dx2$, and $dy2$. Lines 66 to 75 calculate the axis arrays for both the Yee grid and the $2\times$ grid. Observe how the coordinates are adjusted so that $x = y = 0$ at the center of the input face of the photonic crystal. This is done so that when the Gaussian beam is rotated, it will be rotated about this origin and will always illuminate the photonic crystal at the center of the input face. Observe on line 68 that $NPML(1)$ is multiplied by dx to convert it to a physical length. On line 73, $2*NPML(1)$ is multiplied by $dx2$ to convert these $2\times$ grid parameters to physical length.

Now that the grid is calculated correctly to fit the entire photonic crystal, lines 77 to 131 build the materials onto this grid. First, lines 81 to 86 calculate a small $2\times$ meshgrid for just a single unit cell of dimension b . $Nxuc$ and $Nyuc$ are the number of points on this small $2\times$ grid, $xauc$ and $yauc$ are the axis vectors for this grid, and X and Y are the meshgrid arrays for this grid. Lines 88 to 95 build the unit cell on the small $2\times$ grid following techniques described in Chapter 1. Line 89 builds a circle to the middle of the unit cell and lines 90 to 93 add circles at each corner of the unit cell and puts this information into the array $ERuc$. Line 94 converts the 0's and 1's in the array $ERuc$ to the relative permittivity values defined by $er1$ and $er2$ in the dashboard. Line 95 defines the relative permeability throughout the unit cell as all 1's because this device has only the vacuum permeability. With the unit cell constructed, it is time to array the unit cell across the full grid. Lines 97 and 99 initialize the arrays $ER2$ and $UR2$ to all 1's to correspond to vacuum. These arrays are the relative permittivity and relative permeability, respectively, across the full $2\times$ grid. The $2\times$ grid is especially useful here due to the curved geometry of the photonic crystal. The double loop from lines 101 to 113 arrays the unit cell to form the whole photonic crystal. In this loop, $nx1$, $nx2$, $ny1$, and $ny2$ are the start and stop indices of where a unit cell is to be placed onto the $2\times$ grid. Lines 110 and 111 incorporate the unit cell at the location defined by these array indices. After the double loop,

the device has been constructed and is visualized to the figure window on lines 115 to 120. Line 123 incorporates the UPML using the function `addupml2d()` discussed in Chapter 5. Last, lines 125 to 131 form the diagonal materials matrices that will be used to calculate the wave matrix A .

It is now time to perform FDFD! Lines 137 to 141 build the derivative matrices by calling the function `yeeDer2d()` discussed in Chapter 4. Dirichlet boundary conditions are defined at all boundaries in the array `BC` since there is a UPML at each of the boundaries. Line 144 builds the wave matrix A for the E mode using the materials matrices and the derivative matrices.

Next, the source field is calculated on lines 146 to 152. The concept of calculating the source field `fsrc` is shown in Figure 8.12. Think of the source field as the field that would result if there was no device built onto the grid. Figure 8.12(a) shows the Gaussian beam across the grid without the angle of incidence built in. Figure 8.12(b) shows the same Gaussian beam but with the 20° angle of incidence incorporated. To incorporate the angle of incidence, the meshgrid is rotated on lines 148 to 150. Line 148 calculates the ordinary meshgrid, line 149 converts the Cartesian meshgrid to polar coordinates, and line 150 converts the polar coordinates back to Cartesian after adding the angle of incidence `theta` to the angle meshgrid term. This allows the simple Gaussian beam equation to be used on line 151 to calculate the source field, but using the rotated meshgrid parameters. A real Gaussian beam diverges, but is a more complicated function to calculate. A simple Gaussian beam can be used as long as the source function is prevented from crossing the far-right TF/SF interface. On the far-right side, the source function would not match the simulated beam because the simulated beam diverges. Line 152 accomplishes this by setting the source function to zero at all points to the right of the input face of the photonic crystal. The meshgrid was set up so that $x = 0$ at the input face of the photonic crystal. The final source field is shown in Figure 8.12(c). Figure 8.12(d) shows the field calculated from a simulation without a photonic crystal to fully illustrate that the source beam is diverging. It is a good practice to run a simulation without the

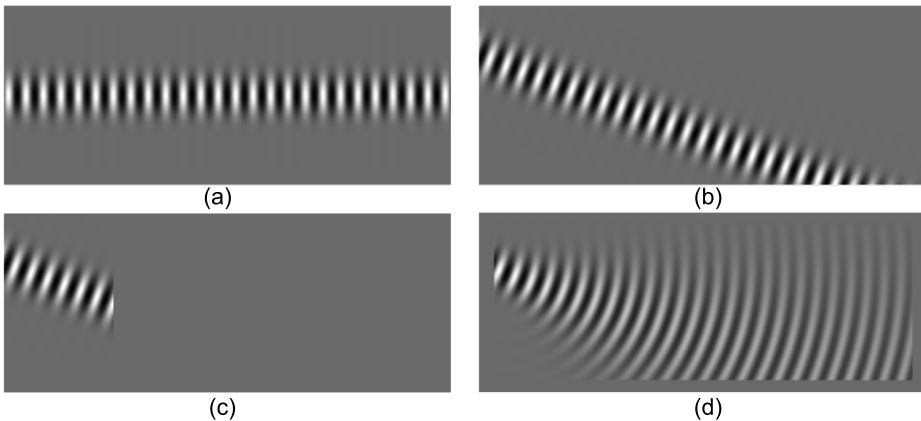


Figure 8.12 Calculation of the source field. (a) Source field with no rotation. (b) Source field with 20° rotation. (c) Truncated source field with 20° rotation. (d) Simulated source field without photonic crystal showing divergence.

device present to ensure that the code is working correctly up to that point. Problems with the code can be more difficult to diagnose when the simulation results are complicated by the presence of a device.

Next, the SF masking matrix Q is constructed on lines 154 to 161. A single TF/SF interface running vertically through the left part of the grid is not sufficient for this Gaussian beam source. If a large enough angle of incidence were used, the Gaussian beam would run more vertically causing the beam to overlap the UPML and move away from the TF/SF interface. Both of these will cause problems in the FDFD simulation. To prevent these problems, the TF/SF interface is made into a complete perimeter around the grid just outside of the UPML regions. This is shown in Figure 8.11. Given Q , the source vector b is calculated using the QAAQ equation on line 164. Do not confuse the source vector b with the unit cell dimension b . It is an unfortunate reuse of the same variable name. Last, the matrix equation is solved on line 167 to calculate the field f . It is reshaped back to the two-dimensional Yee grid on line 168 so that it can be visualized and postprocessed.

The field calculated from the simulation is visualized from lines 170 to 190, and the result is shown in Figure 8.13. Line 183 uses MATLAB's `contour()` function to superimpose the photonic crystal onto the fields. It draws the contour lines at the edges of the photonic crystal lattice. It is a very good practice to combine the device and simulated field into the same diagram in some manner. Much can be learned by observing how the field interacts with the device and material interfaces. For this example, the lattice is illuminated with a diverging beam incident at 20° . Despite the beam's divergence and angle of incidence, the beam inside of the lattice is self-collimated and forced to propagate straight along the x -axis without diverging. When the beam exits the lattice, the beam continues with essentially the same behavior as it entered the lattice. In this simulation, nothing was done at the edges of the photonic crystal to minimize reflections and the scattered waves can be observed in the simulation. A reflected wave is observed at the input face of the lattice as well as a self-collimated beam inside the lattice reflected from the output face.

To ensure the most reliable simulation possible, a convergence study should be performed. It was observed that reasonable convergence was achieved at $NRES = 20$.

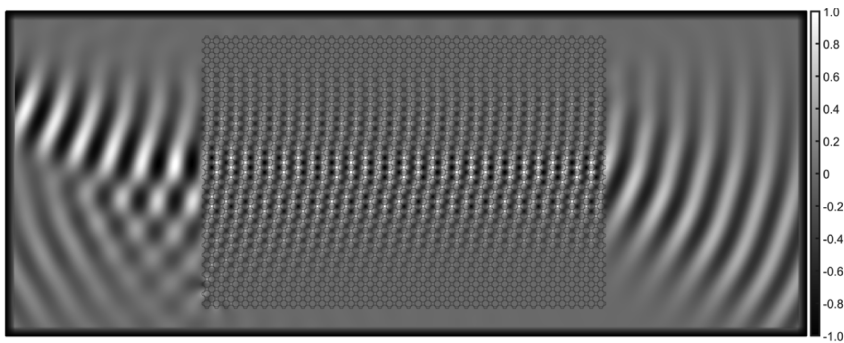


Figure 8.13 Simulation results from a self-collimating photonic crystal.

It is a mistake to make any conclusions about a simulation without first ensuring proper convergence.

8.4.4 FDFD Analysis of an OIC Directional Coupler

To demonstrate how FDFD can be used to simulate OICs, the directional coupling device shown in Figure 8.14 will be analyzed. A directional coupler is formed when two waveguides are brought close enough together to be electromagnetically coupled [6]. When this happens, coupled-mode theory [7–10] shows that there is a periodic exchange of power between the waveguides. Directional couplers are commonly used in microwave circuits [11, 12], OICs [6], and other frequency bands [13].

For best computational efficiency, the three-dimensional OIC will be reduced to a two-dimensional representation using the effective index method (EIM) discussed in Chapter 6. The rib waveguide will be the same design discussed in Chapter 6, but with smaller dimensions in order that the waveguide support only the fundamental mode. The revised dimensions are $h = 300$ nm, $t = 100$ nm, and $w = 400$ nm. The total length of the input waveguide along the x -direction will be 8.525 μm . The second waveguide will start at a distance of 775 nm from the start of the input waveguide. Both waveguides end at the same position on the right. The gap between the waveguides is 155 nm. By experimenting with the simulation, it can be observed that smaller gaps will increase the coupling between the waveguides and shorten the distance it takes to exchange power. The *coupling length* is the distance required to transfer the maximum amount of power from the first waveguide into the second waveguide. While a directional coupler is simulated here, it should be clear that any type of circuit can be analyzed including coupled-line filters, ring resonators, waveguide bends, and more.

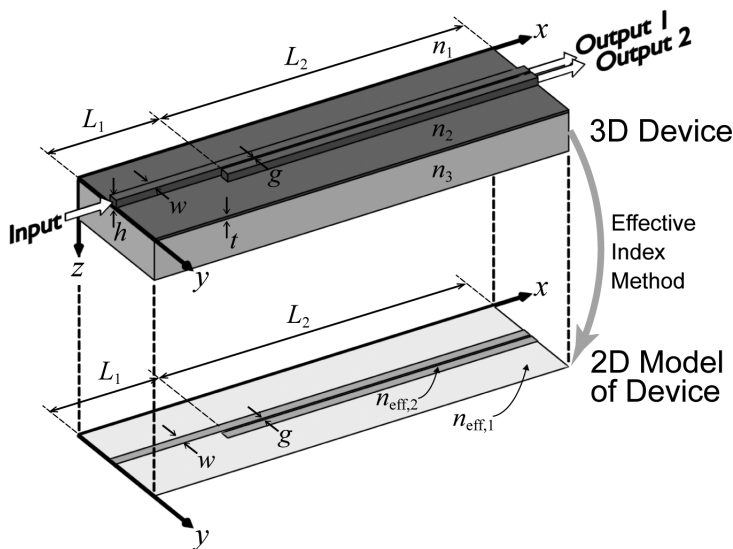


Figure 8.14 Reducing a three-dimensional directional coupler to two dimensions.

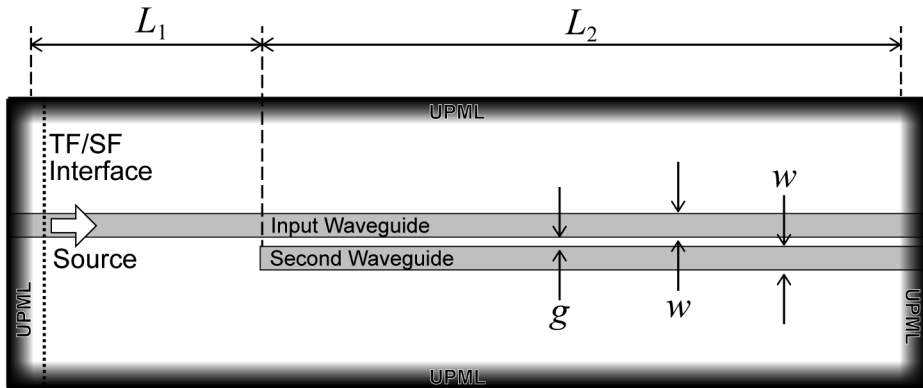


Figure 8.15 Grid strategy for simulating an OIC directional coupler.

The grid strategy for this simulation is depicted in Figure 8.15. A UPML is placed at all grid boundaries and will be 20 cells large. The input waveguide will run the entire length of the grid in the x -direction. The second waveguide will run parallel to the first waveguide, but its starting point will be some distance to the right. The TF/SF interface will cross vertically through the grid just to the right of the leftmost UPML. This is where the source will be injected into the input waveguide using the QAAQ technique.

A block diagram of the steps to simulate this OIC is provided in Figure 8.16. The first three steps follow the ordinary FDFD algorithm summarized in Figure 8.5. Before the OIC can be represented on a two-dimensional grid using the EIM, the two effective refractive indices must be calculated. A first slab waveguide analysis is performed for a slab waveguide defined by the vertical cross section off of the rib waveguide. Given that the slab is illuminated with a vertically polarized mode, the slab waveguide analysis must be H mode. The effective refractive index of the fundamental mode from this analysis is recorded as $n_{\text{eff},1}$. A second slab waveguide analysis is performed for the H mode of a slab waveguide defined by the vertical cross section on the rib waveguide. The effective refractive index of the fundamental mode from this analysis is recorded as $n_{\text{eff},2}$. Now the OIC is constructed onto the two-dimensional grid using $n_{\text{eff},1}$ for locations off of the rib waveguide and $n_{\text{eff},2}$ for locations on the rib waveguide. It is now that a third slab waveguide analysis is performed in the cross section of the grid perpendicular to the input waveguide. This is calculated for the E mode and the fundamental mode is selected for the source. The source function is created by calculating the fundamental mode across the entire grid, adding phase at each position x according to $\exp(-jk_0 n_{\text{eff}} x)$. Otherwise, the FDFD analysis follows that of Figure 8.5 and is performed for the E mode. When the simulation is finished, the field at the output is analyzed by weighing it against all of the modes in the cross section of the waveguide. From this, transmission and reflection are calculated. The last step is to visualize the fields calculated by the simulation.

The MATLAB code to simulate this device can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter8_directionalcoupler.m`. The

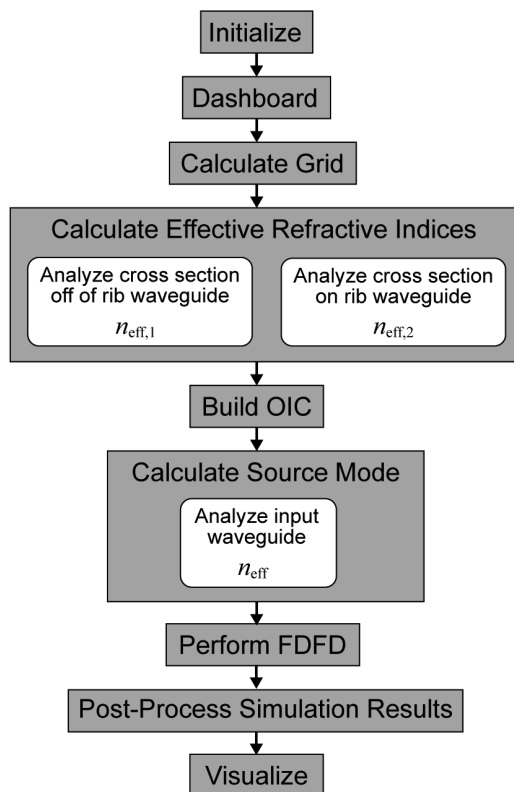


Figure 8.16 Block diagram of FDFD analysis of an OIC.

header for the function extends from lines 1 to 9. The first line is the name of the MATLAB program. Lines 4 to 6 initialize MATLAB by closing any open figure window, clearing the command window, and clearing all variables from memory. Lines 8 and 9 define the units of the program. Being a photonics simulation, the only unit defined here is `micrometers`, but it is often useful to also define `nanometers`.

Lines 11 to 36 comprise the dashboard of the program where all parameters are defined that control the simulation. The source parameters are defined on lines 16 and 17 to include the free space wavelength λ_{am0} and the free space wavenumber k_0 . The parameters describing the dimensions and material properties of the rib waveguide are defined from lines 19 to 25 just how they were in Chapter 6. Lines 27 to 30 define parameters specific to the directional coupler. This includes the length of space L_1 between the start of the input waveguide and the start of the second waveguide, the length of the second waveguide L_2 , and the gap distance g between the two waveguides. Observe that these are defined in terms of the source wavelength λ_{am0} instead of absolute dimensions. Using this approach, the OIC will behave the same regardless of the chosen wavelength. If a parameter sweep is to be performed, these parameters should be defined in terms of a design wavelength, not the wavelength of the simulation. Last, lines 32 to 38 define all of the parameters related to the numerical aspects of the simulation. `NRES` defines the resolution of the simulation as the number of points per minimum wavelength. `SPACER` contains four

numbers identifying how much physical space there should be on all four sides of the OIC. NPML contains a similar set of four numbers defining the size of the UPML at each grid boundary in terms of the number of points. n_{\max} is the maximum refractive index occurring anywhere in the simulation. This is needed to later determine the minimum wavelength that must be resolved by N_{RES} cells.

With all of the parameters controlling the simulation defined, the first task is to calculate a grid that is optimized for the device to be simulated. The grid is calculated from lines 38 to 69 and starts by calculating the grid resolution parameters dx and dy . They are calculated as the smallest wavelength in the simulation λ_{\min}/n_{\max} divided by N_{RES} . This calculation does not consider at all the device dimensions so it is entirely probable that critical dimensions are not represented exactly by an integer number of cells on the grid. Having critical dimensions represented exactly this way improves the accuracy and convergence rate of the simulation. To address this, lines 46 to 60 adjust the grid resolution parameters dx and dy so that the width of the rib waveguide is represented exactly by an integer number of cells. To adjust dx , first n_x is calculated as the number of cells that should exactly represent the width of the rib. This is calculated as the width of the rib waveguide divided by dx and then rounding up to the nearest integer. At this point, dx is adjusted by recalculating it as the width of the rib divided by n_x . Lines 49 and 50 repeat this same calculation to adjust dy .

At this point, the grid resolution parameters dx and dy are fixed and will not be changed again. The next step is to calculate both the physical size of the grid and the number of points. Lines 53 to 55 do this for the x -direction. S_x is the physical size of the grid in the x -direction and is calculated as the spacer regions plus the two lengths of waveguide L_1 and L_2 defined in the dashboard. Observe that the order of the terms to calculate S_x on line 53 matches where the dimensions lie on the grid. This practice helps to ensure all dimensions are included and nothing is forgotten. N_x is the number of cells of the grid in the x -direction and is calculated as S_x/dx plus the UPML regions on the left and right sides of the grid. Line 55 recalculates the physical size of the grid S_x because the UPMLs were not included in the first calculation of S_x . The same three steps are repeated to calculate S_y and N_y that represent the physical and numerical sizes of the grid in the y -direction.

With the grid parameters calculated and optimized for simulating the device defined in the dashboard, the $2\times$ grid parameters are calculated. This includes the number of points N_{x2} and N_{y2} as well as the grid resolution parameters $dx2$ and $dy2$. The last step in the grid calculation is calculating the axis arrays for both the Yee grid, x_a and y_a , and the $2\times$ grid, x_{a2} and y_{a2} .

Before the device can be constructed onto the grid, the effective refractive indices must be calculated that accurately represent the three-dimensional OIC in two dimensions. Lines 71 to 175 perform two-slab waveguide analyses to determine these. Following Figure 6.6, the first slab waveguide is constructed as a vertical cross section through the OIC away from the rib waveguide. Lines 75 to 124 perform this analysis following the same procedure discussed in Chapter 6 for slab waveguides. The fundamental mode of the rib waveguide is vertically polarized indicating that it is the H modes of the slab waveguide that must be calculated. Here the slab waveguides have modes propagating in the $+x$ -direction and the uniform direction

is y . Line 124 records the effective refractive index $neff1$ of the fundamental mode away from the rib waveguide. Similarly, lines 126 to 175 perform the same analysis of the vertical cross section through the rib waveguide. Line 175 records the effective refractive index $neff2$ of the fundamental mode on the rib waveguide. The output of this entire section of code is the two effective refractive indices $neff1$ and $neff2$.

With the grid calculated and the two effective indices calculated for the EIM, lines 177 to 215 build the OIC onto the grid, incorporate the UPMLs and form the diagonal materials matrices. Lines 182 and 183 initialize the relative permittivity and relative permeability arrays on the $2\times$ grid. The relative permittivity array $ER2$ is initialized to all values of $neff1^2$ so that it is only the ribs described by $neff2^2$ that must be incorporated. The effective refractive index is squared to convert the refractive index to relative permittivity. The relative permeability array $UR2$ is set to all ones. Lines 185 to 188 add the input waveguide to the array $ER2$. To do this, the array indices of the vertical start and stop positions are calculated as $ny1$ and $ny2$. Given these, the input waveguide is added across the grid in the x -direction on line 188. A similar procedure is used to add the second waveguide in lines 190 to 194. One difference for the second waveguide is that the array index nx where the waveguide starts in the x -direction is also calculated. The second waveguide is added to $ER2$ on line 194. At this point, the array $ER2$ is finished and lines 196 to 201 visualize the two-dimensional model of the OIC. It is always a good practice to visualize the arrays $UR2$ and $ER2$ to verify they represent the device correctly. With the $2\times$ grid arrays $UR2$ and $ER2$ constructed correctly, line 204 incorporates the UPML and returns the materials arrays on the Yee grid $ERxx$, $ERYy$, $ERzz$, $URxx$, $URyy$, and $URzz$. Lines 206 to 212 form the diagonal materials matrices that will be used to calculate the wave matrix A in a later step.

With the device modeled onto the grid, the next step is to analyze the input waveguide in order to calculate the source mode. This calculation is necessary because the source is a guided mode. This happens on lines 214 to 243 where a third slab waveguide analysis is performed of the cross section of the input waveguide. Line 219 calculates the array index nx where the cross section is to be obtained. Lines 220 to 222 do several things all at once. They extract the cross section of the waveguide, extract the points from the $2\times$ grid to the Yee grid, and form the diagonal materials matrices for the slab waveguide analysis. Lines 224 to 228 build the derivative matrices for the slab waveguide analysis. Dirichlet boundary conditions are used. Lines 230 to 232 build the eigenvalue problem which is then solved on lines 234 to 238. To be consistent with polarization, this slab is analyzed for the E mode. Observe on line 235 that the A and B matrices of the generalized eigenvalue problem are converted to full matrices, the command `eig()` is called instead of `eigs()`, and no other input arguments are given. This is done because it is necessary to calculate all of the eigenmodes from the eigenvalue problem to correctly analyze reflected and transmitted fields. Since all of the modes are being calculated, there is no need to specify the number of modes to calculate or give an estimate for the eigenvalue. The eigenmodes are sorted in descending order of the real part of the effective refractive indices. This places the fundamental mode in the first position. Lines 241 to 243 extract both the field profile and the effective refractive index of the fundamental mode. Observe that the eigenvector matrix Ez is recorded so that it can be used later to calculate reflection and transmission.

It is finally time to perform FDFD analysis of the directional coupler circuit. This happens on lines 245 to 275. Lines 249 to 253 build the derivative matrices using Dirichlet boundary conditions. The wave matrix A is then calculated on line 256 from the derivative matrices and diagonal materials matrices. The source field is calculated on lines 258 to 262. A two-dimensional array f_{src} on the Yee grid is initialized to all zeros on line 259. The loop on lines 260 to 262 fills in the array f_{src} one vertical slice at time. Each vertical slice is filled in with the field profile of the source and the phase of the source. The phase of the source is calculated using the effective refractive index n_{eff} of the source mode. Figure 8.17 shows the final source field calculated for this example. Recall that the source field is the field that would result if no scattering occurred. That means, this is the field that would result if there were only the input waveguide across the grid and no second waveguide or anything else.

Lines 264 to 268 build the SF masking matrix Q . For this problem, there is only a single TF/SF interface running vertically through the grid of two cells to the right of the leftmost UPML, as illustrated in Figure 8.15. The interface is two cells to the right so that there is a cell in the SF region from which to observe reflection. Line 271 calculates the source vector b using the QAAQ equation. Last, the matrix equation is solved on line 274 and the calculated field is reshaped back to the two-dimensional Yee grid on line 275. Line 274 is the most computationally intensive step despite being the shortest line of code in the entire program.

At this point, the FDFD simulation is complete and postprocessing and visualization of the results can begin. Lines 277 to 293 analyze the reflected and transmitted fields to determine how much power is reflected and transmitted in the fundamental mode. To do this, lines 282 to 285 extract the fields in the cross sections of the grid on the far left and right sides of the grid but not in the UPMLs. These are stored in the arrays f_{ref} and f_{trn} . The simulated field along with the fields recorded in f_{ref} and f_{trn} are shown in Figure 8.18. This directional coupler produced near-zero reflections and split the field evenly between the two output waveguides. This is called a 3-dB directional coupler. By adjusting the length over which the waveguides are parallel, different split ratios can be realized.

Given the reflected and transmitted fields, the amplitudes of all possible modes in the cross section of the grid are calculated in lines 288 and 289, but it is only the amplitude of the fundamental mode that is of interest here. The columns in an eigenvector matrix V represent an orthogonal set of configurations that the field is allowed to have. Whatever the field actually looks like, it must be a linear



Figure 8.17 Visualization of the source field f_{src} for a guided mode.

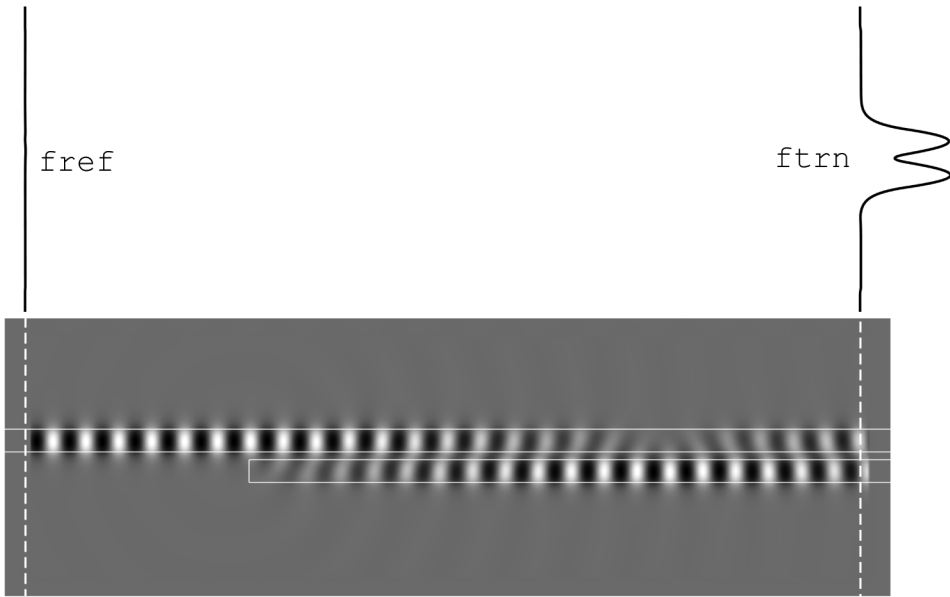


Figure 8.18 Simulation results of the OIC directional coupler.

combination of the eigenvectors in \mathbf{V} . If the amplitudes of all the modes are stored in a column vector \mathbf{a} , then the overall field is $\mathbf{f} = \mathbf{V}\mathbf{a}$. In the case of the FDFD simulation, the overall field \mathbf{f} is found from the simulation and it is the amplitudes stored in \mathbf{a} that need to be calculated. This is found by solving the matrix equation for \mathbf{a} to get $\mathbf{a} = \mathbf{V}^{-1}\mathbf{f}$. The fraction of power reflected into the fundamental mode R of the input waveguide is calculated on line 292. The fraction of power transmitted in the fundamental mode T of the input waveguide is calculated on line 293. These are somewhat analogous to the reflectance and transmittance terms calculated in the diffraction grating example. Depending on the circuit being analyzed, this R and T will rarely obey conservation because power is often scattered out of the waveguides or coupled into higher-order modes. The semicolons were left off of lines 292 and 293 so that the results are reported to the command window.

The very last step is to visualize the simulated results. This happens on lines 295 to 317. The field is visualized using the `pcolor()` function to obtain nice-looking smooth fields. The device is superimposed using the `contour()` function.

References

- [1] Rumpf, R. C., "Simple Implementation of Arbitrarily Shaped Total-Field/Scattered-Field Regions in Finite-Difference Frequency-Domain," *Progress in Electromagnetics Research*, Vol. 36, 2012, pp. 221–248.
- [2] Merewether, D., R. Fisher, and F. Smith, "On Implementing a Numeric Huygen's Source Scheme in a Finite Difference Program to Illuminate Scattering Bodies," *IEEE Trans. on Nuclear Science*, Vol. 27, No. 6, 1980, pp. 1829–1833.

- [3] Umashankar, K., and A. Taflove, "A Novel Method to Analyze Electromagnetic Scattering of Complex Objects," *IEEE Trans. on Electromagnetic Compatibility*, Vol. EMC-24, No. 4, 1982, pp. 397–405.
- [4] Saleh, B. E., and M. C. Teich, *Fundamentals of Photonics*, Hoboken, NJ: John Wiley & Sons, 2019.
- [5] Taflove, A., and S. C. Hagness, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, Norwood, MA: Artech House, 2005.
- [6] Hunsperger, R. G., *Integrated Optics*, New York: Springer, 1995.
- [7] Hardy, A., and W. Streifer, "Coupled Mode Theory of Parallel Waveguides," *Journal of Lightwave Technology*, Vol. 3, No. 5, 1985, pp. 1135–1146.
- [8] Haus, H. A., and W. Huang, "Coupled-Mode Theory," *Proceedings of the IEEE*, Vol. 79, No. 10, 1991, pp. 1505–1518.
- [9] Huang, W.-P., "Coupled-Mode Theory for Optical Waveguides: An Overview," *Journal of the Optical Society of America A*, Vol. 11, No. 3, 1994, pp. 963–983.
- [10] Yariv, A., "Coupled-Mode Theory for Guided-Wave Optics," *IEEE J. of Quantum Electronics*, Vol. 9, No. 9, 1973, pp. 919–933.
- [11] Cohn, S. B., and R. Levy, "History of Microwave Passive Components with Particular Attention to Directional Couplers," *IEEE Trans. on Microwave Theory and Techniques*, Vol. 32, No. 9, 1984, pp. 1046–1054.
- [12] Levy, R., "Directional Couplers," *Advances in Microwaves*, Vol. 1, 1966, pp. 115–209.
- [13] Lu, J.-T., *et al.*, "Terahertz Pipe-Waveguide-Based Directional Couplers," *Optics Express*, Vol. 19, No. 27, 2011, pp. 26883–26890.

Parameter Sweeps with FDFD

Chapter 9 discusses how to modify the MATLAB code for finite-difference frequency-domain (FDFD) in order to perform parameter sweeps. Different types of parameter sweeps are discussed along with best practices. As a first example, the response of a guided-mode resonance filter (GMRF) [1–4] is simulated as a function of wavelength in the range $1.4\text{ }\mu\text{m} < \lambda_0 < 1.5\text{ }\mu\text{m}$. For the second example, the geometry of a terahertz polarizer will be optimized and its response will be plotted as a function of frequency.

9.1 Introduction to Parameter Sweeps

Parameter sweeps are probably the most powerful tool in existence for analyzing and designing electromagnetic and photonic devices. The concept is simple. The results of a simulation are plotted as some variable is varied over a range of values. The most common parameter sweep is probably transmission and reflection plotted as a function of frequency. From parameter sweeps, trends can be determined, device dimensions can be optimized, and discoveries can be made!

Parameter sweeps are as simple as placing a big `for` loop around the FDFD code and recording the results as the loop iterates. This is incredibly easy to do if proper dashboard discipline is being practiced. Careful thought should be given about what steps in the FDFD algorithm are included inside of the loop. Steps that produce the same result throughout the parameter sweep should be moved outside of the loop if at all possible.

Block diagrams of common parameter sweeps are shown in Figure 9.1. First, Figure 9.1(a) shows a block diagram summarizing the major steps in ordinary FDFD. This is provided so that the modifications required for different parameter sweeps can be more easily identified. Figure 9.1(b) shows a block diagram for a wavelength (or frequency) parameter sweep that is by far the most common type of parameter sweep. While it is possible to rebuild the grid and device for each wavelength, it is not recommended because this practice will lead to a jagged response. Instead, the grid and device should be constructed prior to the loop that is based on the worst case requiring the finest resolution. This is usually the shortest wavelength or highest frequency in the sweep. All other steps to implement the FDFD method appear inside of the loop.

Another type of parameter sweep involves changing a dimension associated with the device being simulated. This type of parameter sweep is used very often to optimize the performance of a device. Figure 9.1(c) shows the flow of steps for

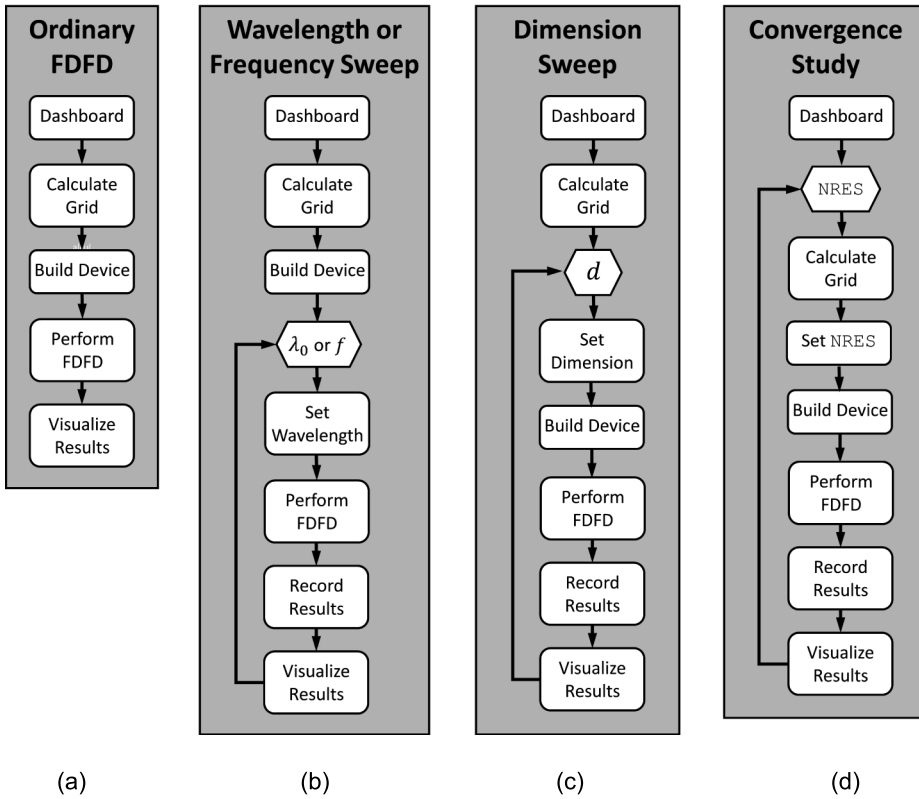


Figure 9.1 Block diagrams of common parameter sweeps in FDFD.

this type of parameter sweep. Sometimes it may be necessary to recalculate a grid for each new dimension, but it is recommended to calculate a grid that will accommodate the full range of dimensions to be simulated. This can be accomplished by calculating a grid with fine enough resolution to resolve the smallest dimensions while also having enough points on the grid to accommodate the largest dimensions. Calculating the grid this way will avoid getting a jagged response. In this sweep, the device has to be rebuilt every iteration so the build step appears inside of the parameter sweep loop.

The last parameter sweep, shown in Figure 9.1(d), is a convergence study where the simulation is repeated as the resolution of the grid is increased. This requires essentially all steps for FDFD to be placed inside of the loop. A convergence study is the primary means to determine proper grid resolution, large enough spacer regions, and large enough perfectly matched layers (PMLs). Convergence studies must be a standard practice in computational electromagnetics.

Observe that each parameter sweep includes a visualization step inside of the loop. Parameter sweeps can be time-consuming, so it is useful to visualize the results as they are being calculated. Many times, errors can be caught early in the parameter sweep. Execution can be stopped, errors corrected, and then the parameter sweep restarted without wasting time waiting for an entire parameter sweep to finish. The key to computation is visualization.

9.2 Modifying FDFD for Parameter Sweeps

The biggest trick to make parameter sweeps easy and utilize simpler code is to write a generic function that performs the FDFD analysis. By putting as much of the FDFD code into the generic function as possible, the parameter sweep itself becomes very short and simple. Having a generic function to implement the FDFD method is useful for other things because programs can concentrate on building devices and analyzing the results.

An excellent device to perform a parameter sweep for is a GMRF [1–4]. A GMRF is both a diffraction grating [5] and a slab waveguide [6] at the same time. For most frequencies, a GMRF behaves as an ordinary dielectric slab and exhibits a slow and weak response in terms of transmission and reflection. Then, over a very narrow range of frequencies, one of the diffraction orders from the diffraction grating is able to couple power from the applied wave into a mode guided in the slab and the transmission and reflection from the GMRF change abruptly. GMRFs are incredibly sensitive to dimensions and material properties so if anything is wrong with the FDFD code, a GMRF simulation will tend to amplify it. For this reason, GMRFs are excellent devices for benchmarking and practicing FDFD simulations. GMRFs are also very interesting and useful devices to study!

The GMRF that will be simulated is illustrated in Figure 9.2. The device itself is a slab waveguide with a diffraction grating at its core. There is no requirement where the diffraction grating has to reside relative to the slab waveguide as long as they are close enough to be electromagnetically coupled. For a wavelength sweep, the spacer regions are set to half of the maximum wavelength in the sweep. This is a good choice if nothing else is known. However, if there is any apriori knowledge of the resonant wavelength, the spacer regions could be set to half of the resonant wavelength. It is the resonant wavelength that will have the largest evanescent fields because this is the wavelength of the guided mode. Regardless, no evanescent fields should be allowed to extend into the UPMLs. The simulation will calculate reflectance and transmittance in the range $1.4 \mu\text{m} < \lambda_0 < 1.5 \mu\text{m}$.

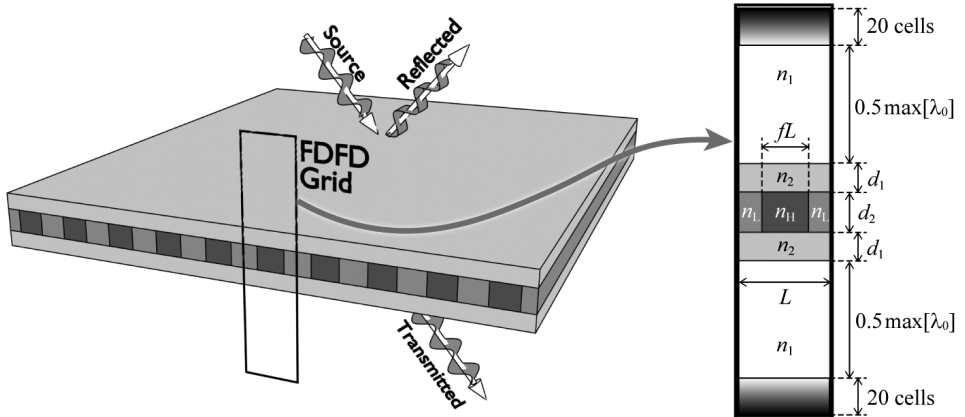


Figure 9.2 GMRF and representation on a two-dimensional FDFD grid. This device has $f = 0.5$, $n_1 = 1.0$, $n_2 = 1.4$, $n_L = 1.9$, $n_H = 2.1$, $d_1 = 265 \text{ nm}$, $d_2 = 380 \text{ nm}$, and $L = 870 \text{ nm}$. The product fL is the physical width of the high-index region.

9.2.1 Generic MATLAB Function to Simulate Periodic Structures

The first step to perform a parameter sweep of the device in Figure 9.2 is to write a generic function that performs FDFD analysis of any periodic structure. The recommended way of doing this is to first write a standard FDFD program to simulate the device at one frequency, just like the diffraction grating example in Chapter 8. Save this program with a name like `fdfd2d_proto.m`. Then save this same file as two additional files, and give one the name `fdfd2d.m` and the other the name `demo_gmrf.m`. In the `fdfd2d()` function, delete everything before the FDFD analysis so that it only contains the steps specific to performing FDFD. In the `demo_gmrf.m` file, delete everything after the device is built onto the grid. Then, add some code to `demo_gmrf.m` to call the `fdfd2d()` function to perform the simulation. Rather than having a large number of input arguments, wrap all parameters defining the device to be simulated in the MATLAB structure `DEV` and all the parameters defining the source in the MATLAB structure `SRC`. Then, all output data will be put into the MATLAB structure `DAT`. When done correctly, the call to the `fdfd2d()` function will be

```
DAT = fdfd2d(DEV,SRC);
```

In a sense, `demo_gmrf.m` will be the first half of the original program `fdfd2d_proto.m` while `fdfd2d()` will be the second half. Most of the work in creating the two new MATLAB programs entails changing variable names to accommodate the use of structures.

The overall grid strategy for simulating periodic structures that `fdfd2d()` will assume is illustrated in Figure 9.3. Periodic boundary conditions (PBCs) are used at the x -axis boundaries to model a device that is infinitely periodic in the x -direction. UPMLs are located at the y -axis boundaries to absorb outgoing waves. The device itself is located near the center of the grid vertically. A top spacer region is inserted above the device to allow sufficient space for the field to be visualized and to prevent evanescent fields from entering the top PML. A bottom spacer region is inserted below the device for the same reasons. The source is injected into the simulation in the $+y$ -direction at the TF/SF interface. The TF/SF interface can be placed anywhere between the top PML and the device, but cannot overlap either of these. For this example, the TF/SF interface is placed just a few cells below the top PML but it can be located lower than this if it is desired to visualize more of the SF. Immediately below the top PML is the *reflection plane* where the reflected waves are analyzed. It must be located in the SF region so it can be placed anywhere above the TF/SF interface and below the top PML. Immediately above the bottom PML is the *transmission plane* where the transmitted waves are analyzed. The location of the transmission plane can be anywhere between the bottom of the device and the bottom PML but not overlapping either of these.

The MATLAB code for the generic function `fdfd2d()` can be downloaded at <https://empossible.net/fdfdbook/>. The first line is the function heading required by MATLAB where the input arguments and output arguments are defined. The commented section from lines 2 to 27 is what will be displayed to the command window when `help fdfd2d` is typed at the command prompt. It is strongly recommended to include a header like this in all functions in order to remember

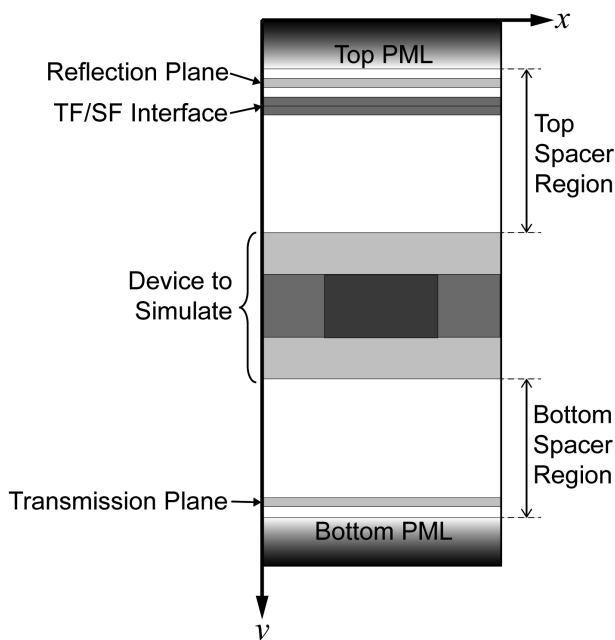


Figure 9.3 Grid strategy assumed by the `fdfd2d()` function for simulating periodic structures with FDFD.

how to use them at a later time. In this header, the input and output arguments along with their fields are listed in the header. The structure `DEV` will contain the arrays `ER2` and `UR2` to define what is to be simulated in a generic way. `DEV` will also contain `RES` with the grid resolution parameters `dx` and `dy` and `NPML` with the size of the UPML at the top and bottom of the grid. The function will always simulate periodic structures so no UPML is used on the left and right sides of the grid. The structure `SRC` will contain the free space wavelength `lam0`, the angle of incidence `theta`, and the mode `MODE` to be simulated that will be either an “E” or an “H.”

It is strongly recommended to avoid having redundant input arguments. For example, it would be a huge mistake to make the grid size parameters `Nx` and `Ny` be input arguments. That is because these can be determined from the inside of the `fdfd2d()` function from the size of either `ER2` or `UR2`. Having redundant input arguments creates extra work and the potential for mistakes.

The section from lines 29 to 56 is a new section that is needed to extract all of the required FDFD parameters from the input arguments. Line 34 determines the size of the $2\times$ grid from the size of the `ER2` array. Lines 36 to 45 calculate the remaining grid parameters and the meshgrid for the Yee grid. Line 48 calculates the free space wavenumber. Lines 51 to 56 extract the material properties of the external regions. These are needed to calculate the source and to calculate reflection and transmission.

The FDFD method is implemented from lines 62 to 105. This code is essentially unchanged from Chapter 8. The simulated field is then postprocessed to analyze reflection and transmission. This analysis appears from lines 107 to 135. Other than

using the variables inside of the structures, the code is identical to that discussed in Chapter 8.

Observe that this `fdfd2d()` function is generic in the sense that it does not have to know anything about the device being simulated. As long as the device is to be simulated with PBCs on the left and right boundaries, this same code can simulate virtually any periodic structure. Not only does this function allow for easy parameter sweeps, it can also simplify code intended to simulate devices at just a single frequency by eliminating all of the FDFD steps. It is a very versatile and useful function to have!

9.2.2 Main MATLAB Program to Simulate the GMRF

The MATLAB code to build the GMRF and run the wavelength sweep can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter9_GMRF.m`. Other than some variables being put into data structures, the code up to line 107 is nearly identical to the original code that simulated a diffraction grating. The header goes from lines 1 to 11 and gives the name of the file, initializes MATLAB, and defines some units. Being a photonics simulation, `micrometers` is set to 1.

The dashboard goes from lines 13 to 39. There are some additional variables defined for the source because a wavelength sweep is being performed. The variable `lam1` defines the shortest wavelength of the sweep, `lam2` defines the longest wavelength of the sweep, and `NLAM` defines how many wavelength points there will be over the range from `lam1` to `lam2`. Line 21 calculates the array of wavelengths from `lam1` to `lam2`. A separate FDFD simulation will be performed for each wavelength in this array so it is important to use the minimum number of points `NLAM` that still resolves the reflection and transmission response sufficiently. Lines 22 and 23 define the angle of incidence `theta` and mode `MODE` to be simulated as was done in Chapter 8. Lines 25 to 33 define all of the parameters needed to build the GMRF onto a grid, as illustrated in Figure 9.2. These are not made input arguments into `fdfd2d()` because all of the information about the device that FDFD needs to know will be built into the arrays `ER2` and `UR2`. The parameters defining the GMRF include the duty cycle `f`, refractive index `n1` above and below the GMRF, refractive index `n2` of the top and bottom layers of the GMRF, the low refractive index `nL` in the grating region, the high refractive index `nH` in the grating region, thickness `d1` of the top and bottom layers, thickness `d2` of the grating region, and the period `L` of the grating.

Lines 41 to 72 calculate a grid that is optimized for simulating this GMRF. It follows the same flow as discussed in Chapter 8. The grid resolution parameter `dx` was adjusted to represent the grating period `L` with an exact integer number of cells. The grid resolution parameter `dy` was adjusted to represent the grating thickness `d2` with an exact integer number of cells.

Lines 74 to 105 build the GMRF on the $2\times$ grid and display the `ER2` array to the figure window. Lines 79 and 80 initialize the relative permeability array `UR2` to vacuum and the relative permittivity array `ER2` to n_1^2 . Lines 82 to 92 calculate the start and stop array indices of all of the features of the GMRF. The variable `nx1` is calculated to be the array index of the left side of the high-index region of the grating, while the variable `nx2` is the array index of the right side. The variables `ny1`

and `ny2` are the array indices of the top and bottom of the top layer of the GMRF. The variables `ny3` and `ny4` are the array indices of the top and bottom of the grating layer. Last, the variables `ny5` and `ny6` are the array indices of the top and bottom of the bottom layer of the GMRF. Given all of these array indices, lines 94 to 98 build the GMRF into the `ER2` array. The array `ER2` is visualized from lines 100 to 105 to the first subplot of a figure window that is five subplots wide. The field calculated during the simulation will be displayed to the second subplot and the reflection and transmission spectra will be displayed in the third to fifth subplot spaces.

The wavelength sweep is implemented in the next section of the code from lines 107 to 152. Line 112 defines the structure field `RES` that contains the grid resolution parameters `dx` and `dy`. Lines 115 to 117 initialize the three arrays where the response of the device will be stored. `REF` will contain the overall reflectance calculated for each wavelength in the array `LAMBDA`, `TRN` will contain the overall transmittance, and `CON` will contain the power conservation that is reflectance plus transmittance. It is always best practice to plot the conservation in addition to reflectance and transmittance because this is a great indicator of the health and accuracy of the simulation. It should be within 1% of 100% for all frequencies. Sometimes larger errors arise around wavelengths that correspond to the cutoff of a diffraction order where waves are propagating nearly parallel to the x -axis. The wavelength sweep itself occurs from lines 119 to 152 inside of a `for` loop. For important loops, consider using a bit more significant comment like what is on lines 119 to 121. Line 122 starts the `for` loop to progress the variable `n1am` from a value of 1 at the start of the sweep to a value of `NLAM` at the end of the sweep. The first step inside of the loop is on line 125 where the free space wavelength `lam0` is set for the current simulation. Line 128 then performs the FDFD simulation at this wavelength. Lines 131 to 133 record the result. This is very simple code that demonstrates how easy parameter sweeps become when there exists a generic function to perform the simulation! The rest of the code in the loop simply visualizes the results. Lines 135 to 140 visualize the field calculated at each wavelength in the second subplot. Lines 143 to 151 plot the reflection and transmission spectrum as they are calculated. It is very good practice to visualize the device, field, and spectra during the sweep. These are the key things to visualize to catch errors and monitor the status of the simulation.

As discussed in previous chapters, having code that runs without error is not sufficient to simulate a device. A convergence study must be performed. If nothing is known about the device, convergence can be checked at the shortest wavelength that will typically converge last. If anything is known about the device, convergence can be checked at the resonant wavelength that usually exhibits the slowest convergence. It is even possible to track the center position of resonance of the GMRF and plot that as a function of `NRES`. Regardless of how it is done, make performing a convergence study standard practice!

This device was found to converge somewhere between `NRES=20` and `NRES=40`. Figure 9.4 shows the reflection, transmission, and conservation lines for a wavelength sweep performed with `NRES=40` for the E mode at two different angles of incidence. A narrow resonance is observed around $\lambda_0 = 1.5 \mu\text{m}$ for normal incidence. For the very small angle of incidence of 2.5° , the resonance split into two resonances observed around $\lambda_1 = 1.465 \mu\text{m}$ and $\lambda_2 = 1.53 \mu\text{m}$. GMRFs are very sensitive devices!

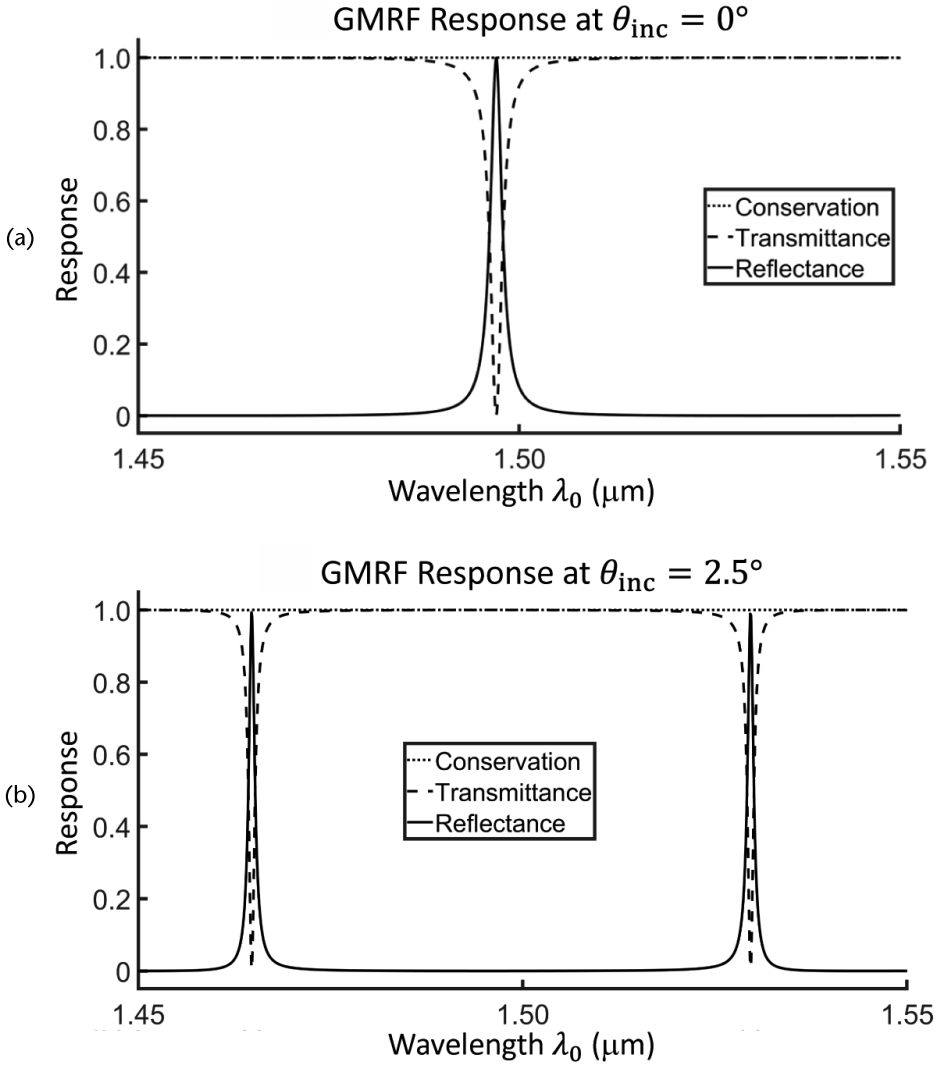


Figure 9.4 Spectral response of the GMRF in Figure 9.2 for the E mode at two different angles of incidence.

While plotting the conservation line should be standard practice in day-to-day simulation work, it is generally ignored when generating graphics for presentations and publications. The audience will assume the results presented are correct and were generated after convergence was obtained.

9.2.3 Main MATLAB Programs to Analyze a Metal Polarizer

To demonstrate a different type of parameter sweep, the wire grid terahertz polarizer presented in [7] will be analyzed here using FDFD. The basic structure of the polarizer is illustrated in Figure 9.5 and is composed of gold lines deposited onto the surface of a grating etched into a silicon (Si) substrate. It is a wire grid polarizer because it is composed of an array of long continuous wires. It is a bilayer polarizer

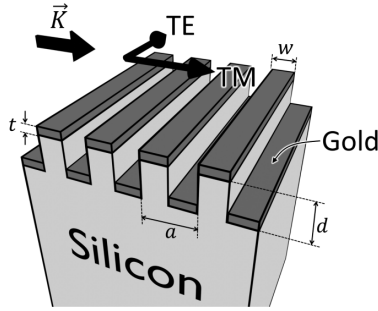


Figure 9.5 Geometry of the polarizer presented in [7] along with the direction of TE and TM polarization. The optimized design has $a = 4.0 \mu\text{m}$, $t = 200 \text{ nm}$, $w = 2.0 \mu\text{m}$, and $d = 1.5 \mu\text{m}$. The dielectric constant of the substrate is $\epsilon_r = 11.8$ and the gold is made a perfect electric conductor (PEC).

because there are two layers of wires at different heights, and the interaction between the two produces the broadband and high extinction ratio that the device exhibits. When the electric field is polarized parallel to the wires, the field is able to push the free charges along the wires and the device acts much like a solid metal surface, reflecting the wave. This is called the transverse electric (TE) polarization because the electric field is transverse to the grating vector \vec{K} that defines the period and direction of the grooves. When the electric field is polarized perpendicular to the wires [transverse magnetic (TM) polarization], the free charges in the wires are constrained in the perpendicular direction and the device acts more like a dielectric, allowing the wave to transmit. This is called the TM polarization because the magnetic field is transverse to the grating vector \vec{K} .

The extinction ratio ξ is the transmittance T_{TM} of the TM wave that transmits through the polarizer divided by the transmittance T_{TE} of the TE wave blocked by the polarizer.

$$\xi = \frac{T_{\text{TM}}}{T_{\text{TE}}} \quad (9.1)$$

For polarizers, it is common to define TE and TM relative to the grating vector \vec{K} instead of the plane of incidence (POI). This is because polarizers are often used at normal incidence where no POI can be defined. In the context of a two-dimensional FDFD simulation, the E mode is the TE polarization in this case because it has the electric field transverse to the grating vector \vec{K} while the H mode is the TM polarization because it has the magnetic field transverse to the grating vector \vec{K} . It is important to note that this definition of TE and TM assigns the E and H modes opposite to how they have been assigned everywhere else in this book. Be cautious about the definition of TE and TM when reading the literature!

There are four key parameters in Figure 9.5 to design in order to produce an optimized design. For brevity, only the grating depth parameter d will be considered here. To choose a value of d that optimizes the extinction ratio, the parameter sweep shown in Figure 9.6 was performed using FDFD. The MATLAB code that performs this parameter sweep can be downloaded at <https://empossible.net/fdfdbook/> and

is called `Chapter9_polarizer_d.m`. The header extends from lines 1 to 23. It was chosen to make `micrometers` equal to 1 based on the dimensions of the polarizer and the wavelength of the terahertz waves.

The all-important dashboard extends from lines 25 to 52 and defines all the parameters that specify the source, device, and numerical parameters that control the simulation. Lines 29 to 32 define the source parameters. To design a polarizer operating in the range from 0.6 THz to 3.0 THz, the frequency in the parameter sweep is set to 2.0 THz, which is roughly in the middle of the range of frequencies of interest. The angle of incidence `SRC.theta` is set to 0° . The fixed dimensions of the polarizer are defined first from lines 35 to 37, consistent with that in Figure 9.5. The material properties are defined on lines 39 and 40. Lines 42 to 45 define the parameter sweep to be performed. The variable `d1` is the starting value for the grating depth and `d2` is the ending value. These extreme cases are illustrated as insets in Figure 9.6. `NDAT` is the number of points to use in the sweep. `d_DAT` is an array of values for grating depth where a simulation will be performed for each to determine the extinction ratio. The grid parameters are defined from lines 47 to 52. `NRES` is set to a very high value of 200 where good convergence was observed. This is typical for simulations containing metals.

Lines 54 to 71 calculate the initial guess for the grid resolution parameters, but the final calculations for all of the grid parameters have been moved to the main loop. This section calculates the initial guess based on resolving the minimum wavelength and then refines the parameters in order to resolve the minimum dimension. For this design, the metal layer thickness `t` is in the z -direction.

The parameter sweep is performed on lines 73 to 180. This is a large section of code inside of the loop because most of the steps in FDFD must be performed inside of the loop for this type of parameter sweep. Lines 77 to 85 initialize the arrays where the results of the simulations are stored. The reflectance, transmittance, and power conservation for the E mode (TE polarization) are stored in the arrays `REF_E`, `TRN_E`, and `CON_E`, respectively. The reflectance, transmittance, and power conservation for the H mode (TM polarization) are stored in the arrays `REF_H`, `TRN_H`, and `CON_H`, respectively. The last two arrays, `ER` and `ER_dB`, store the extinction ratio on a linear and decibel scale, respectively. Do not confuse these arrays with relative permittivity.

The main loop for the parameter sweep extends from lines 86 to 180. The loop begins on line 90. The first step in the loop on line 93 grabs the next value of grating depth `d` that should be simulated. Lines 95 to 117 finish calculating the grid that the device will be built onto. Lines 119 to 121 initialize the relative permittivity array `DEV.ER2` and relative permeability array `DEV.UR2` to all ones, corresponding to air. The device is built onto the grid using start and stop indices that are calculated on lines 123 to 131. `nx` is calculated to be the number of cells wide for the tall grating tooth. It is calculated in a way that ensures `nx` will be an even number by dividing the argument of the `round()` function by two and then multiplying the `round()` function on the outside by two. The variable `nx1` is the starting index of the tall grating tooth in the x -direction and is calculated using the centering algorithm from Chapter 1. However, the centering algorithm has been modified in a way that ensures `nx1` will be an odd number so that the tangential electric field components will be at the inside edge of the tooth. This is done by dividing the argument of

`floor()` by two, multiplying the result of `floor()` by two, and then adding one. `nx2` is the ending index which should also be an odd number for the same reason. Since `nx` was calculated to be an even number, `nx2` is guaranteed to be odd. In the y -direction, `ny1` and `ny2` are the start and stop array indices of the top metal layer and `ny3` and `ny4` are the start and stop array indices of the bottom metal layer. These are also calculated in a way that ensures they are odd numbers. Given the start and stop array indices, lines 133 to 137 build the polarizer onto the $2\times$ grid.

Just prior to simulation, line 140 defines the remaining field `DEV.RES` that specifies the resolution parameters `dx` and `dy` to the function `fdfd2d()`. From there, the E mode (TE polarization) is simulated on lines 142 and 143 and the results are recorded on lines 145 to 147. Similarly, the H mode (TM polarization) is simulated on lines 150 and 151 and the results are recorded on lines 152 to 154. Lines 156 to 158 calculate the extinction ratio on a linear scale in the array `ER` and also calculate it on a decibel (dB) scale in the array `ER_dB`. Lines 160 to 179 visualize the fields for each mode and plot the extinction ratio as the simulation is running. It is always a good practice to visualize the intermediate results of a parameter sweep as it is running because errors can often be caught early on without having to wait for the parameter sweep to finish.

The results of the parameter sweep are shown in Figure 9.6 and were obtained with `NRES=200`. It is typical for simulations of subwavelength metallic devices to require high values of `NRES`. In [7], a value of $d = 1.5\ \mu\text{m}$ was chosen for the grating depth.

Given a design, a second parameter sweep was performed for this device to plot transmittance and extinction ratio as a function of frequency from 0.6 THz up to 3.0 THz. The simulation results shown in Figure 9.7 for this sweep were obtained at `NRES=200`. The MATLAB code for this parameter sweep can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter9_polarizer_freq.m`.

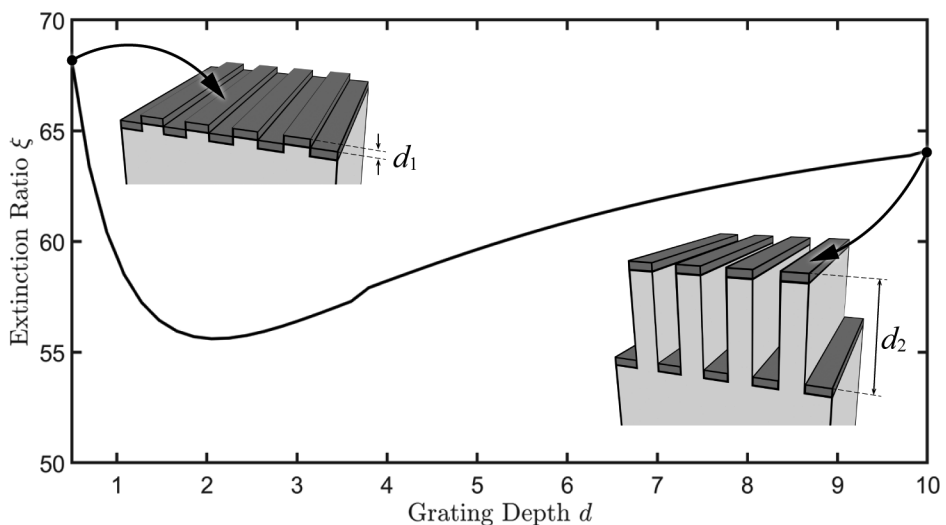


Figure 9.6 Extinction ratio ξ as a function of grating depth d .

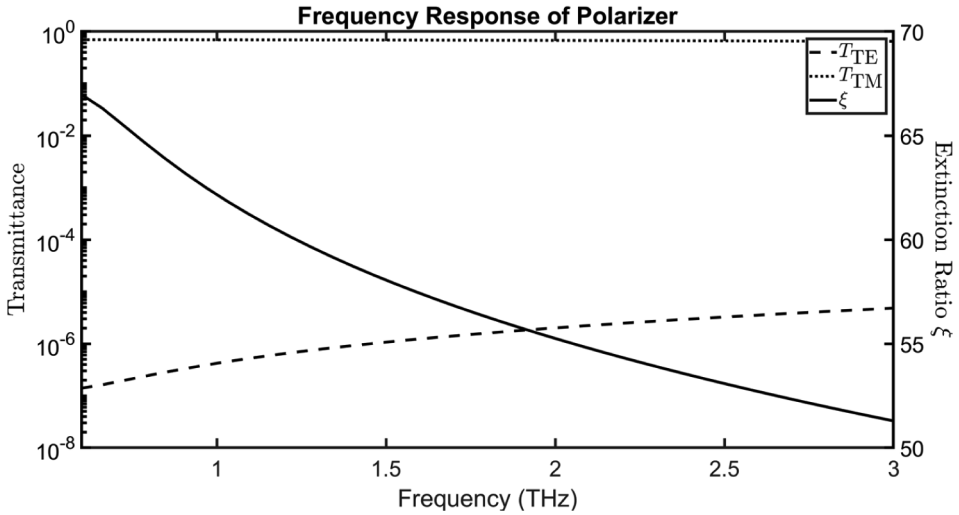


Figure 9.7 Frequency response of the terahertz polarizer.

9.3 Identifying Common Problems in FDFD

Most simulation problems are more easily diagnosed when performing parameter sweeps than they are when simulating a single case at a single frequency. Figure 9.8 illustrates three common problems in FDFD that can be identified through the power conservation versus frequency response. The first common problem is when the absorbing boundaries are not working correctly. To demonstrate this, a vacuum was simulated and the PMLs were forced to produce some reflections by making them only four cells large. Reflections from the boundaries cause standing waves that tend to produce a slow-rolling response in the power conservation. The solid black line in Figure 9.8 shows the power conservation line for an FDFD simulation of a vacuum where the bottom PML is partially reflecting.

A second common problem is insufficient grid resolution. This tends to produce a power conservation that steadily worsens with increasing frequency, as demonstrated by the line with long dashes in Figure 9.8 that is also a simulation of a vacuum. This happens because the lower frequencies have longer wavelengths that are resolved with less numerical error by the grid. To mitigate this, it is not recommended to adjust the resolution of the grid for different frequencies because this will lead to jagged response lines. Instead, calculate the grid based on worst cases, such as the smallest dimensions and highest frequencies, and use the same grid for all frequencies. It is always best in parameter sweeps to keep the simulations as similar as possible to avoid jagged response lines.

A third common problem happens when the spacer regions are made too small. This tends to produce localized spikes in the power conservation, usually around the cutoff frequencies of diffraction orders. The dotted line in Figure 9.8 shows the frequency response of the sawtooth diffraction grating presented in Chapter 8. The solid vertical lines in the figure identify where the diffraction orders have cutoff frequencies. Observe that the spikes in the conservation line correspond to the cutoff frequencies. Diffraction orders near cutoff can produce large evanescent fields or

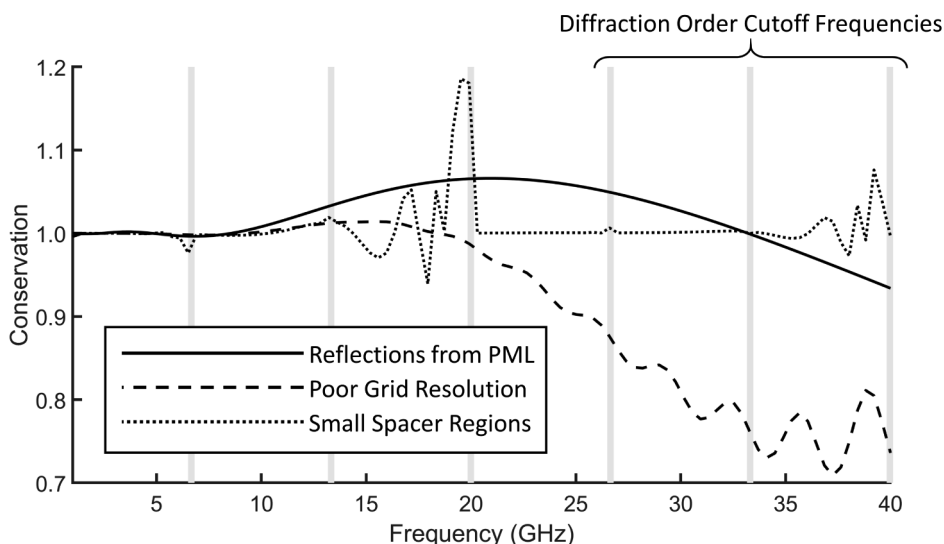


Figure 9.8 Common problems in FDFD identified through power conservation.

waves propagating near parallel to the x -axis. Neither of these are handled well by diffraction order calculations or by the PMLs at the y -axis boundaries. Virtually all numerical methods have similar problems resolving the response of periodic structures near cutoff frequencies. Making the spacer regions too large will not hurt the accuracy of the simulation, but the simulations will require more memory and take a longer time to run.

References

- [1] Barton, J. H., *et al.*, “All-Dielectric Frequency Selective Surface for High Power Microwaves,” *IEEE Trans. on Antennas and Propagation*, Vol. 62, No. 7, 2014, pp. 3652–3656.
- [2] Magnusson, R., and Y. H. Ko, “Guided-Mode Resonance Nanophotonics: Fundamentals and Applications” p. 992702.
- [3] Mehta, A. A., *et al.*, “Guided Mode Resonance Filter as a Spectrally Selective Feedback Element in a Double-Cladding Optical Fiber Laser,” *IEEE Photonics Technology Letters*, Vol. 19, No. 24, 2007, pp. 2030–2032.
- [4] Rumpf, R. C., *et al.*, “Guided-Mode Resonance Filter Compensated to Operate on a Curved Surface,” *Progress in Electromagnetics Research*, Vol. 40, 2013, pp. 93–103.
- [5] Loewen, E. G., and E. Popov, *Diffraction Gratings and Applications*: CRC Press, 2018.
- [6] Okamoto, K., *Fundamentals of Optical Waveguides*: Academic Press, 2006.
- [7] Deng, L., *et al.*, “Extremely High Extinction Ratio Terahertz Broadband Polarizer Using Bilayer Subwavelength Metal Wire-Grid Structure,” *Applied Physics Letters*, Vol. 101, No. 1, 2012, p. 011101.

FDFD Analysis of Three-Dimensional and Anisotropic Devices

When general anisotropy is introduced into finite-difference frequency-domain (FDFD), all electromagnetic field components are coupled and no simplifications can be made to the math even if the device is one or two-dimensional [1]. For this reason, anisotropy will be introduced at the same time as three dimensions. Two-dimensional simulations are still possible using a three-dimensional implementation simply by setting one of the grid dimensions equal to one. Regardless, three-dimensional FDFD becomes much more computationally intensive and several modifications to the algorithm are needed to make it solvable on ordinary desktop computers. Most significantly, the matrix equation $\mathbf{f} = \mathbf{A}^{-1}\mathbf{b}$ will be solved iteratively instead of directly to conserve memory. The FDFD method as presented so far in this book is not easily solvable by iteration due to poor conditioning of the wave matrix \mathbf{A} [2]. Conditioning will be improved by replacing the uniaxial perfectly matched layer (UPML) with a stretched-coordinate PML (SCPML) absorbing boundary. This is an advanced chapter that requires a strong understanding of all other chapters in this book. It can be argued that FDFD is not well suited for three-dimensional simulations, but some things are still possible. This chapter will demonstrate how FDFD can be used to simulate crossed gratings, simulate frequency selective surfaces, do parameter retrieval for metamaterials, and simulate devices designed by transformation optics (TO) like invisibility cloaks.

10.1 Formulation of Three-Dimensional FDFD

Formulation of three-dimensional FDFD starts by making Maxwell's equations discrete. The field functions are made discrete and distributed throughout space according to the Yee grid scheme [3]. The partial derivatives are approximated using finite differences. When anisotropy is introduced, however, a problem arises in the discrete equations. Not all of the terms are defined at the same points. Interpolations must be performed in the discrete equations to ensure all terms are defined at the same points. This has the unfortunate consequence of complicating the discrete equations and adding to the computational burden of performing simulations with anisotropy. After the interpolations are added, the discrete equations are written for every cell on the grid and the large sets of equations are expressed in matrix form. New matrices arise in this formulation to perform the interpolations and incorporate the SCPML terms. Finally, the wave matrix is derived as a surprisingly simple matrix equation. This is an excellent example of how calculating small and simple

matrices can make calculating big and complicated matrices much simpler. To finish the formulation, incorporating sources using the TF/SF method and solving the final matrix equation by iteration is discussed.

10.1.1 Finite-Difference Approximation of Maxwell's Curl Equations

From Chapter 5, Maxwell's curl equations with an SCPML expand into the following set of six coupled partial differential equations. In these equations, all nine tensor elements are retained for both permeability and permittivity.

$$\frac{1}{s_y} \frac{\partial E_z}{\partial y} - \frac{1}{s_z} \frac{\partial E_y}{\partial z} = k_0 (\mu_{xx} \tilde{H}_x + \mu_{xy} \tilde{H}_y + \mu_{xz} \tilde{H}_z) \quad (10.1)$$

$$\frac{1}{s_z} \frac{\partial E_x}{\partial z} - \frac{1}{s_x} \frac{\partial E_z}{\partial x} = k_0 (\mu_{yx} \tilde{H}_x + \mu_{yy} \tilde{H}_y + \mu_{yz} \tilde{H}_z) \quad (10.2)$$

$$\frac{1}{s_x} \frac{\partial E_y}{\partial x} - \frac{1}{s_y} \frac{\partial E_x}{\partial y} = k_0 (\mu_{zx} \tilde{H}_x + \mu_{zy} \tilde{H}_y + \mu_{zz} \tilde{H}_z) \quad (10.3)$$

$$\frac{1}{s_y} \frac{\partial \tilde{H}_z}{\partial y} - \frac{1}{s_z} \frac{\partial \tilde{H}_y}{\partial z} = k_0 (\epsilon_{xx} E_x + \epsilon_{xy} E_y + \epsilon_{xz} E_z) \quad (10.4)$$

$$\frac{1}{s_z} \frac{\partial \tilde{H}_x}{\partial z} - \frac{1}{s_x} \frac{\partial \tilde{H}_z}{\partial x} = k_0 (\epsilon_{yx} E_x + \epsilon_{yy} E_y + \epsilon_{yz} E_z) \quad (10.5)$$

$$\frac{1}{s_x} \frac{\partial \tilde{H}_y}{\partial x} - \frac{1}{s_y} \frac{\partial \tilde{H}_x}{\partial y} = k_0 (\epsilon_{zx} E_x + \epsilon_{zy} E_y + \epsilon_{zz} E_z) \quad (10.6)$$

In the above equations, μ_{mn} and ϵ_{mn} are the tensor elements of permeability and permittivity, respectively. The terms s_x , s_y , and s_z are the complex stretching parameters associated with the SCPML. From here, the grid is normalized according to $x' = k_0 x$, $y' = k_0 y$, and $z' = k_0 z$ to arrive at the final form of the analytical equations.

$$\frac{1}{s_y} \frac{\partial E_z}{\partial y'} - \frac{1}{s_z} \frac{\partial E_y}{\partial z'} = \mu_{xx} \tilde{H}_x + \mu_{xy} \tilde{H}_y + \mu_{xz} \tilde{H}_z \quad (10.7)$$

$$\frac{1}{s_z} \frac{\partial E_x}{\partial z'} - \frac{1}{s_x} \frac{\partial E_z}{\partial x'} = \mu_{yx} \tilde{H}_x + \mu_{yy} \tilde{H}_y + \mu_{yz} \tilde{H}_z \quad (10.8)$$

$$\frac{1}{s_x} \frac{\partial E_y}{\partial x'} - \frac{1}{s_y} \frac{\partial E_x}{\partial y'} = \mu_{zx} \tilde{H}_x + \mu_{zy} \tilde{H}_y + \mu_{zz} \tilde{H}_z \quad (10.9)$$

$$\frac{1}{s_y} \frac{\partial \tilde{H}_z}{\partial y'} - \frac{1}{s_z} \frac{\partial \tilde{H}_y}{\partial z'} = \epsilon_{xx} E_x + \epsilon_{xy} E_y + \epsilon_{xz} E_z \quad (10.10)$$

$$\frac{1}{s_z} \frac{\partial \tilde{H}_x}{\partial z'} - \frac{1}{s_x} \frac{\partial \tilde{H}_z}{\partial x'} = \epsilon_{yx} E_x + \epsilon_{yy} E_y + \epsilon_{yz} E_z \quad (10.11)$$

$$\frac{1}{s_x} \frac{\partial \tilde{H}_y}{\partial x'} - \frac{1}{s_y} \frac{\partial \tilde{H}_x}{\partial y'} = \varepsilon_{zx} E_x + \varepsilon_{zy} E_y + \varepsilon_{zz} E_z \quad (10.12)$$

Now (10.7) to (10.12) must be made discrete. Writing these equations in discrete form is trickier than it was in previous chapters due to the PML terms and the incorporation of general anisotropy. Start by taking a guess at expressing (10.7) in a discrete form where the partial derivatives are approximated using central finite differences.

$$\begin{aligned} & \frac{1}{s_y|_{i,j,k}^{Hx}} \frac{E_z|_{i,j+1,k} - E_z|_{i,j,k}}{\Delta y} - \frac{1}{s_z|_{i,j,k}^{Hx}} \frac{E_y|_{i,j,k+1} - E_y|_{i,j,k}}{\Delta z} \\ & \neq \mu_{xx}|_{i,j,k} \tilde{H}_x|_{i,j,k} + \mu_{xy}|_{i,j,k} \tilde{H}_y|_{i,j,k} + \mu_{xz}|_{i,j,k} \tilde{H}_z|_{i,j,k} \end{aligned} \quad (10.13)$$

This equation may appear to be correct at first glance, but recall that each term in a finite-difference equation must be defined at the same point in space. The majority of the terms in (10.13) are defined at the same location as $\tilde{H}_x|_{i,j,k}$. The superscripts on the PML terms indicate where the PML terms are defined. Unfortunately, according to the Yee grid scheme, the terms $\mu_{xy}|_{i,j,k} \tilde{H}_y|_{i,j,k}$ and $\mu_{xz}|_{i,j,k} \tilde{H}_z|_{i,j,k}$ are located at entirely different locations than $\tilde{H}_x|_{i,j,k}$. To correct this, the products $\mu_{xy} \tilde{H}_y$ and $\mu_{xz} \tilde{H}_z$ are interpolated at the location of $\tilde{H}_x|_{i,j,k}$. These interpolations are essentially the average value from the surrounding four points.

$$\mu_{xy} \tilde{H}_y \approx \frac{\mu_{xy}|_{i-1,j,k} \tilde{H}_y|_{i-1,j,k} + \mu_{xy}|_{i,j,k} \tilde{H}_y|_{i,j,k} + \mu_{xy}|_{i-1,j+1,k} \tilde{H}_y|_{i-1,j+1,k} + \mu_{xy}|_{i,j+1,k} \tilde{H}_y|_{i,j+1,k}}{4} \quad (10.14)$$

$$\mu_{xz} \tilde{H}_z \approx \frac{\mu_{xz}|_{i-1,j,k} \tilde{H}_z|_{i-1,j,k} + \mu_{xz}|_{i,j,k} \tilde{H}_z|_{i,j,k} + \mu_{xz}|_{i-1,j,k+1} \tilde{H}_z|_{i-1,j,k+1} + \mu_{xz}|_{i,j,k+1} \tilde{H}_z|_{i,j,k+1}}{4} \quad (10.15)$$

The correct discrete form of (10.7) can now be written and the same procedure applied to all of (10.8) to (10.12). These are

$$\begin{aligned} & \frac{1}{s_y|_{i,j,k}^{Hx}} \frac{E_z|_{i,j+1,k} - E_z|_{i,j,k}}{\Delta y'} - \frac{1}{s_z|_{i,j,k}^{Hx}} \frac{E_y|_{i,j,k+1} - E_y|_{i,j,k}}{\Delta z'} \\ & = \mu_{xx}|_{i,j,k} \tilde{H}_x|_{i,j,k} \\ & + \frac{\mu_{xy}|_{i-1,j,k} \tilde{H}_y|_{i-1,j,k} + \mu_{xy}|_{i,j,k} \tilde{H}_y|_{i,j,k} + \mu_{xy}|_{i-1,j+1,k} \tilde{H}_y|_{i-1,j+1,k} + \mu_{xy}|_{i,j+1,k} \tilde{H}_y|_{i,j+1,k}}{4} \\ & + \frac{\mu_{xz}|_{i-1,j,k} \tilde{H}_z|_{i-1,j,k} + \mu_{xz}|_{i,j,k} \tilde{H}_z|_{i,j,k} + \mu_{xz}|_{i-1,j,k+1} \tilde{H}_z|_{i-1,j,k+1} + \mu_{xz}|_{i,j,k+1} \tilde{H}_z|_{i,j,k+1}}{4} \end{aligned} \quad (10.16)$$

$$\begin{aligned}
& \frac{1}{s_z|_{i,j,k}^{Hy}} \frac{E_x|_{i,j,k+1} - E_x|_{i,j,k}}{\Delta z'} - \frac{1}{s_x|_{i,j,k}^{Hy}} \frac{E_z|_{i+1,j,k} - E_z|_{i,j,k}}{\Delta x'} \\
&= \frac{\mu_{yx}|_{i,j,k} \tilde{H}_x|_{i,j,k} + \mu_{yx}|_{i+1,j,k} \tilde{H}_x|_{i+1,j,k} + \mu_{yx}|_{i,j-1,k} \tilde{H}_x|_{i,j-1,k} + \mu_{yx}|_{i+1,j-1,k} \tilde{H}_x|_{i+1,j-1,k}}{4} \\
&+ \mu_{yy}|_{i,j,k} \tilde{H}_y|_{i,j,k} \\
&+ \frac{\mu_{yz}|_{i,j-1,k} \tilde{H}_z|_{i,j-1,k} + \mu_{yz}|_{i,j,k} \tilde{H}_z|_{i,j,k} + \mu_{yz}|_{i,j-1,k+1} \tilde{H}_z|_{i,j-1,k+1} + \mu_{yz}|_{i,j,k+1} \tilde{H}_z|_{i,j,k+1}}{4}
\end{aligned} \tag{10.17}$$

$$\begin{aligned}
& \frac{1}{s_x|_{i,j,k}^{Hz}} \frac{E_y|_{i+1,j,k} - E_y|_{i,j,k}}{\Delta x'} - \frac{1}{s_y|_{i,j,k}^{Hz}} \frac{E_x|_{i,j+1,k} - E_x|_{i,j,k}}{\Delta y'} \\
&= \frac{\mu_{zx}|_{i,j,k} \tilde{H}_x|_{i,j,k} + \mu_{zx}|_{i+1,j,k} \tilde{H}_x|_{i+1,j,k} + \mu_{zx}|_{i,j,k-1} \tilde{H}_x|_{i,j,k-1} + \mu_{zx}|_{i+1,j,k-1} \tilde{H}_x|_{i+1,j,k-1}}{4} \\
&+ \frac{\mu_{zy}|_{i,j,k} \tilde{H}_y|_{i,j,k} + \mu_{zy}|_{i,j+1,k} \tilde{H}_y|_{i,j+1,k} + \mu_{zy}|_{i,j,k-1} \tilde{H}_y|_{i,j,k-1} + \mu_{zy}|_{i,j+1,k-1} \tilde{H}_y|_{i,j+1,k-1}}{4} \\
&+ \mu_{zz}|_{i,j,k} \tilde{H}_z|_{i,j,k}
\end{aligned} \tag{10.18}$$

$$\begin{aligned}
& \frac{1}{s_y|_{i,j,k}^{Ex}} \frac{\tilde{H}_z|_{i,j,k} - \tilde{H}_z|_{i,j-1,k}}{\Delta y'} - \frac{1}{s_z|_{i,j,k}^{Ex}} \frac{\tilde{H}_y|_{i,j,k} - \tilde{H}_y|_{i,j,k-1}}{\Delta z'} \\
&= \varepsilon_{xx}|_{i,j,k} E_x|_{i,j,k} \\
&+ \frac{\varepsilon_{xy}|_{i,j-1,k} E_y|_{i,j-1,k} + \varepsilon_{xy}|_{i,j,k} E_y|_{i,j,k} + \varepsilon_{xy}|_{i+1,j-1,k} E_y|_{i+1,j-1,k} + \varepsilon_{xy}|_{i+1,j,k} E_y|_{i+1,j,k}}{4} \\
&+ \frac{\varepsilon_{xz}|_{i,j,k} E_z|_{i,j,k} + \varepsilon_{xz}|_{i+1,j,k} E_z|_{i+1,j,k} + \varepsilon_{xz}|_{i,j,k-1} E_z|_{i,j,k-1} + \varepsilon_{xz}|_{i+1,j,k-1} E_z|_{i+1,j,k-1}}{4}
\end{aligned} \tag{10.19}$$

$$\begin{aligned}
& \frac{1}{s_z|_{i,j,k}^{Ey}} \frac{\tilde{H}_x|_{i,j,k} - \tilde{H}_x|_{i,j,k-1}}{\Delta z'} - \frac{1}{s_x|_{i,j,k}^{Ey}} \frac{\tilde{H}_z|_{i,j,k} - \tilde{H}_z|_{i-1,j,k}}{\Delta x'} \\
&= \frac{\varepsilon_{yx}|_{i-1,j,k} E_x|_{i-1,j,k} + \varepsilon_{yx}|_{i,j,k} E_x|_{i,j,k} + \varepsilon_{yx}|_{i-1,j+1,k} E_x|_{i-1,j+1,k} + \varepsilon_{yx}|_{i,j+1,k} E_x|_{i,j+1,k}}{4} \\
&+ \varepsilon_{yy}|_{i,j,k} E_y|_{i,j,k} \\
&+ \frac{\varepsilon_{yz}|_{i,j,k} E_z|_{i,j,k} + \varepsilon_{yz}|_{i,j+1,k} E_z|_{i,j+1,k} + \varepsilon_{yz}|_{i,j,k-1} E_z|_{i,j,k-1} + \varepsilon_{yz}|_{i,j+1,k-1} E_z|_{i,j+1,k-1}}{4}
\end{aligned} \tag{10.20}$$

$$\begin{aligned}
& \frac{1}{s_x|_{i,j,k}^{Ez}} \frac{\tilde{H}_y|_{i,j,k} - \tilde{H}_y|_{i-1,j,k}}{\Delta x'} - \frac{1}{s_y|_{i,j,k}^{Ez}} \frac{\tilde{H}_x|_{i,j,k} - \tilde{H}_x|_{i,j-1,k}}{\Delta y'} \\
&= \frac{\epsilon_{zx}|_{i-1,j,k} E_x|_{i-1,j,k} + \epsilon_{zx}|_{i,j,k} E_x|_{i,j,k} + \epsilon_{zx}|_{i-1,j,k+1} E_x|_{i-1,j,k+1} + \epsilon_{zx}|_{i,j,k+1} E_x|_{i,j,k+1}}{4} \\
&+ \frac{\epsilon_{zy}|_{i,j-1,k} E_y|_{i,j-1,k} + \epsilon_{zy}|_{i,j,k} E_y|_{i,j,k} + \epsilon_{zy}|_{i,j-1,k+1} E_y|_{i,j-1,k+1} + \epsilon_{zy}|_{i,j,k+1} E_y|_{i,j,k+1}}{4} \\
&+ \epsilon_{zz}|_{i,j,k} E_z|_{i,j,k}
\end{aligned} \tag{10.21}$$

Inspection of (10.16) to (10.18) shows that μ_{xx} , μ_{yx} , and μ_{zx} should be defined at the same points as \tilde{H}_x . Similarly, μ_{xy} , μ_{yy} , and μ_{zy} should be defined at the same points as \tilde{H}_y and μ_{xz} , μ_{yz} , and μ_{zz} should be defined at the same points as \tilde{H}_z . Inspection of (10.19) to (10.21) shows that ϵ_{xx} , ϵ_{yx} , and ϵ_{zx} should be defined at the same points as E_x . Similarly, ϵ_{xy} , ϵ_{yy} , and ϵ_{zy} should be defined at the same points as E_y and ϵ_{xz} , ϵ_{yz} , and ϵ_{zz} should be defined at the same points as E_z .

10.1.2 Maxwell's Equations in Matrix Form

Each of (10.16) to (10.21) is written once for every cell on the grid. Each set of discrete equations can be written in matrix form leading to a set of six matrix equations. These are

$$\mathbf{S}_y^{\text{Hx}} \mathbf{D}_y^e \mathbf{e}_z - \mathbf{S}_z^{\text{Hx}} \mathbf{D}_z^e \mathbf{e}_y = \boldsymbol{\mu}_{xx} \tilde{\mathbf{h}}_x + \mathbf{R}_x^- \mathbf{R}_y^+ \boldsymbol{\mu}_{xy} \tilde{\mathbf{h}}_y + \mathbf{R}_x^- \mathbf{R}_z^+ \boldsymbol{\mu}_{xz} \tilde{\mathbf{h}}_z \tag{10.22}$$

$$\mathbf{S}_z^{\text{Hy}} \mathbf{D}_z^e \mathbf{e}_x - \mathbf{S}_x^{\text{Hy}} \mathbf{D}_x^e \mathbf{e}_z = \mathbf{R}_y^- \mathbf{R}_x^+ \boldsymbol{\mu}_{yx} \tilde{\mathbf{h}}_x + \boldsymbol{\mu}_{yy} \tilde{\mathbf{h}}_y + \mathbf{R}_y^- \mathbf{R}_z^+ \boldsymbol{\mu}_{yz} \tilde{\mathbf{h}}_z \tag{10.23}$$

$$\mathbf{S}_x^{\text{Hz}} \mathbf{D}_x^e \mathbf{e}_y - \mathbf{S}_y^{\text{Hz}} \mathbf{D}_y^e \mathbf{e}_x = \mathbf{R}_z^- \mathbf{R}_x^+ \boldsymbol{\mu}_{zx} \tilde{\mathbf{h}}_x + \mathbf{R}_z^- \mathbf{R}_y^+ \boldsymbol{\mu}_{zy} \tilde{\mathbf{h}}_y + \boldsymbol{\mu}_{zz} \tilde{\mathbf{h}}_z \tag{10.24}$$

$$\mathbf{S}_y^{\text{Ex}} \mathbf{D}_y^h \tilde{\mathbf{h}}_z - \mathbf{S}_z^{\text{Ex}} \mathbf{D}_z^h \tilde{\mathbf{h}}_y = \boldsymbol{\epsilon}_{xx} \mathbf{e}_x + \mathbf{R}_x^+ \mathbf{R}_y^- \boldsymbol{\epsilon}_{xy} \mathbf{e}_y + \mathbf{R}_x^+ \mathbf{R}_z^- \boldsymbol{\epsilon}_{xz} \mathbf{e}_z \tag{10.25}$$

$$\mathbf{S}_z^{\text{Ey}} \mathbf{D}_z^h \tilde{\mathbf{h}}_x - \mathbf{S}_x^{\text{Ey}} \mathbf{D}_x^h \tilde{\mathbf{h}}_z = \mathbf{R}_y^+ \mathbf{R}_x^- \boldsymbol{\epsilon}_{yx} \mathbf{e}_x + \boldsymbol{\epsilon}_{yy} \mathbf{e}_y + \mathbf{R}_y^+ \mathbf{R}_z^- \boldsymbol{\epsilon}_{yz} \mathbf{e}_z \tag{10.26}$$

$$\mathbf{S}_x^{\text{Ez}} \mathbf{D}_x^h \tilde{\mathbf{h}}_y - \mathbf{S}_y^{\text{Ez}} \mathbf{D}_y^h \tilde{\mathbf{h}}_x = \mathbf{R}_z^+ \mathbf{R}_x^- \boldsymbol{\epsilon}_{zx} \mathbf{e}_x + \mathbf{R}_z^+ \mathbf{R}_y^- \boldsymbol{\epsilon}_{zy} \mathbf{e}_y + \boldsymbol{\epsilon}_{zz} \mathbf{e}_z \tag{10.27}$$

Two new types of matrices have appeared in these equations compared to what was presented in Chapter 4 due to incorporating the SCPML and performing interpolations. The \mathbf{S} terms are diagonal matrices containing the inverse of the PML terms across the Yee grid along their diagonal. The term \mathbf{S}_y^{Hx} is a diagonal matrix containing the $s_y^{-1}(x,y,z)$ PML function at the locations of $\tilde{H}_x|_{i,j,k}$. The other \mathbf{S} matrices are defined similarly. The \mathbf{R} terms are matrices that perform interpolations across the Yee grid and look very similar to the derivative matrices. The term \mathbf{R}_x^+ interpolates a function by averaging the function with the value in the next cell in the positive x -direction. The term \mathbf{R}_x^- interpolates a function by averaging the

function with the value in the previous cell in the positive x -direction. The term \mathbf{R}_y^+ interpolates a function by averaging the function with the value in the next cell in the positive y -direction. The term \mathbf{R}_y^- interpolates a function by averaging the function with the value in the previous cell in the positive y -direction. The term \mathbf{R}_z^+ interpolates a function by averaging the function with the value in the next cell in the positive z -direction. The term \mathbf{R}_z^- interpolates a function by averaging the function with the value in the previous cell in the positive z -direction.

10.1.3 Interpolation Matrices

Interpolation matrices are almost identical to derivative matrices [1]. The basic finite-difference approximation of a derivative with respect to x' is

$$\frac{\partial E_z}{\partial x'} \approx \frac{E_z|_{i+1,j,k} - E_z|_{i,j,k}}{\Delta x'} \rightarrow \mathbf{D}_{x'}^e \mathbf{e}_z \quad (10.28)$$

An interpolation operation utilizes the same terms as (10.28), but adds the terms instead of subtracting them, and divides by two instead of dividing by $\Delta x'$.

$$\langle E_z \rangle_{i+0.5,j,k} \approx \frac{E_z|_{i+1,j,k} + E_z|_{i,j,k}}{2} \rightarrow \mathbf{R}_x^+ \mathbf{e}_z \quad (10.29)$$

In fact, when Dirichlet boundary conditions are used, the interpolation matrices can be calculated directly from the derivative matrices. First, the absolute value of every element in the derivative matrix is calculated to convert the subtraction to an addition. Second, the new matrix is multiplied by the resolution parameter $\Delta x'$ and then divided by two. From this, \mathbf{R}_x^+ , \mathbf{R}_y^+ , and \mathbf{R}_z^+ can be calculated directly from $\mathbf{D}_{x'}^e$, $\mathbf{D}_{y'}^e$, and $\mathbf{D}_{z'}^e$ as

$$\mathbf{R}_x^+ = \frac{\Delta x'}{2} \cdot |\mathbf{D}_{x'}^e| \quad \mathbf{R}_y^+ = \frac{\Delta y'}{2} \cdot |\mathbf{D}_{y'}^e| \quad \mathbf{R}_z^+ = \frac{\Delta z'}{2} \cdot |\mathbf{D}_{z'}^e| \quad (10.30)$$

If periodic boundary conditions (PBCs) are used, the interpolation matrices cannot be calculated from the derivative matrices because the absolute value operation alters any complex numbers. A separate function must be written in MATLAB to build the interpolation matrices when PBCs are used. The code to do this is nearly identical to building derivative matrices. To make things easier, after \mathbf{R}_x^+ , \mathbf{R}_y^+ , and \mathbf{R}_z^+ are calculated, the remaining interpolation matrices can be calculated directly from them as follows.

$$\mathbf{R}_x^- = (\mathbf{R}_x^+)^H \quad \mathbf{R}_y^- = (\mathbf{R}_y^+)^H \quad \mathbf{R}_z^- = (\mathbf{R}_z^+)^H \quad (10.31)$$

Equation (10.31) is valid for both Dirichlet boundary conditions and PBCs. If any other boundary conditions are used, be cautious as (10.31) may no longer be valid. While (10.31) may look like how the derivative matrices were related, the relation for interpolation matrices has no negative sign. Be careful!

10.1.4 Three-Dimensional Matrix Wave Equation

Equations (10.22) to (10.24) can be written as a single block matrix equation as well as (10.25) to (10.27) to get the following two block matrix equations.

$$\begin{bmatrix} \mathbf{0} & -\mathbf{S}_z^{\text{Hx}}\mathbf{D}_{z'}^{\text{e}} & \mathbf{S}_y^{\text{Hx}}\mathbf{D}_{y'}^{\text{e}} \\ \mathbf{S}_z^{\text{Hy}}\mathbf{D}_{z'}^{\text{e}} & \mathbf{0} & -\mathbf{S}_x^{\text{Hy}}\mathbf{D}_{x'}^{\text{e}} \\ -\mathbf{S}_y^{\text{Hz}}\mathbf{D}_{y'}^{\text{e}} & \mathbf{S}_x^{\text{Hz}}\mathbf{D}_{x'}^{\text{e}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \\ \mathbf{e}_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu}_{xx} & \mathbf{R}_x^-\mathbf{R}_y^+\boldsymbol{\mu}_{xy} & \mathbf{R}_x^-\mathbf{R}_z^+\boldsymbol{\mu}_{xz} \\ \mathbf{R}_y^-\mathbf{R}_x^+\boldsymbol{\mu}_{yx} & \boldsymbol{\mu}_{yy} & \mathbf{R}_y^-\mathbf{R}_z^+\boldsymbol{\mu}_{yz} \\ \mathbf{R}_z^-\mathbf{R}_x^+\boldsymbol{\mu}_{zx} & \mathbf{R}_z^-\mathbf{R}_y^+\boldsymbol{\mu}_{zy} & \boldsymbol{\mu}_{zz} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \\ \tilde{\mathbf{h}}_z \end{bmatrix} \quad (10.32)$$

$$\begin{bmatrix} \mathbf{0} & -\mathbf{S}_z^{\text{Ex}}\mathbf{D}_{z'}^{\text{h}} & \mathbf{S}_y^{\text{Ex}}\mathbf{D}_{y'}^{\text{h}} \\ \mathbf{S}_z^{\text{Ey}}\mathbf{D}_{z'}^{\text{h}} & \mathbf{0} & -\mathbf{S}_x^{\text{Ey}}\mathbf{D}_{x'}^{\text{h}} \\ -\mathbf{S}_y^{\text{Ez}}\mathbf{D}_{y'}^{\text{h}} & \mathbf{S}_x^{\text{Ez}}\mathbf{D}_{x'}^{\text{h}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \\ \tilde{\mathbf{h}}_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{\epsilon}_{xx} & \mathbf{R}_x^+\mathbf{R}_y^-\boldsymbol{\epsilon}_{xy} & \mathbf{R}_x^+\mathbf{R}_z^-\boldsymbol{\epsilon}_{xz} \\ \mathbf{R}_y^+\mathbf{R}_x^-\boldsymbol{\epsilon}_{yx} & \boldsymbol{\epsilon}_{yy} & \mathbf{R}_y^+\mathbf{R}_z^-\boldsymbol{\epsilon}_{yz} \\ \mathbf{R}_z^+\mathbf{R}_x^-\boldsymbol{\epsilon}_{zx} & \mathbf{R}_z^+\mathbf{R}_y^-\boldsymbol{\epsilon}_{zy} & \boldsymbol{\epsilon}_{zz} \end{bmatrix} \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \\ \mathbf{e}_z \end{bmatrix} \quad (10.33)$$

These equations can be written in an even more compact block matrix form as follows.

$$\mathbf{C}^{\text{e}}\tilde{\mathbf{e}} = [\boldsymbol{\mu}]\tilde{\mathbf{h}} \quad (10.34)$$

$$\mathbf{C}^{\text{h}}\tilde{\mathbf{h}} = [\boldsymbol{\epsilon}]\tilde{\mathbf{e}} \quad (10.35)$$

where

$$\tilde{\mathbf{e}} = \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \\ \mathbf{e}_z \end{bmatrix} \quad (10.36)$$

$$\tilde{\mathbf{h}} = \begin{bmatrix} \tilde{\mathbf{h}}_x \\ \tilde{\mathbf{h}}_y \\ \tilde{\mathbf{h}}_z \end{bmatrix} \quad (10.37)$$

$$\mathbf{C}^{\text{e}} = \begin{bmatrix} \mathbf{0} & -\mathbf{S}_z^{\text{Hx}}\mathbf{D}_{z'}^{\text{e}} & \mathbf{S}_y^{\text{Hx}}\mathbf{D}_{y'}^{\text{e}} \\ \mathbf{S}_z^{\text{Hy}}\mathbf{D}_{z'}^{\text{e}} & \mathbf{0} & -\mathbf{S}_x^{\text{Hy}}\mathbf{D}_{x'}^{\text{e}} \\ -\mathbf{S}_y^{\text{Hz}}\mathbf{D}_{y'}^{\text{e}} & \mathbf{S}_x^{\text{Hz}}\mathbf{D}_{x'}^{\text{e}} & \mathbf{0} \end{bmatrix} \quad (10.38)$$

$$\mathbf{C}^{\text{h}} = \begin{bmatrix} \mathbf{0} & -\mathbf{S}_z^{\text{Ex}}\mathbf{D}_{z'}^{\text{h}} & \mathbf{S}_y^{\text{Ex}}\mathbf{D}_{y'}^{\text{h}} \\ \mathbf{S}_z^{\text{Ey}}\mathbf{D}_{z'}^{\text{h}} & \mathbf{0} & -\mathbf{S}_x^{\text{Ey}}\mathbf{D}_{x'}^{\text{h}} \\ -\mathbf{S}_y^{\text{Ez}}\mathbf{D}_{y'}^{\text{h}} & \mathbf{S}_x^{\text{Ez}}\mathbf{D}_{x'}^{\text{h}} & \mathbf{0} \end{bmatrix} \quad (10.39)$$

$$[\epsilon] = \begin{bmatrix} \epsilon_{xx} & \mathbf{R}_x^+ \mathbf{R}_y^- \epsilon_{xy} & \mathbf{R}_x^+ \mathbf{R}_z^- \epsilon_{xz} \\ \mathbf{R}_y^+ \mathbf{R}_x^- \epsilon_{yx} & \epsilon_{yy} & \mathbf{R}_y^+ \mathbf{R}_z^- \epsilon_{yz} \\ \mathbf{R}_z^+ \mathbf{R}_x^- \epsilon_{zx} & \mathbf{R}_z^+ \mathbf{R}_y^- \epsilon_{zy} & \epsilon_{zz} \end{bmatrix} \quad (10.40)$$

$$[\mu] = \begin{bmatrix} \mu_{xx} & \mathbf{R}_x^- \mathbf{R}_y^+ \mu_{xy} & \mathbf{R}_x^- \mathbf{R}_z^+ \mu_{xz} \\ \mathbf{R}_y^- \mathbf{R}_x^+ \mu_{yx} & \mu_{yy} & \mathbf{R}_y^- \mathbf{R}_z^+ \mu_{yz} \\ \mathbf{R}_z^- \mathbf{R}_x^+ \mu_{zx} & \mathbf{R}_z^- \mathbf{R}_y^+ \mu_{zy} & \mu_{zz} \end{bmatrix} \quad (10.41)$$

The column vectors \mathbf{e}_x , \mathbf{e}_y , and \mathbf{e}_z for the electric field components have been assembled into the single-column vector $\vec{\mathbf{e}}$. The column vectors \mathbf{h}_x , \mathbf{h}_y , and \mathbf{h}_z for the magnetic field components have been assembled into the single-column vector $\vec{\mathbf{h}}$. The curl matrix operating on the electric fields is now written as \mathbf{C}^e and includes the SCPML terms. The curl matrix operating on the magnetic fields is written as \mathbf{C}^h and also includes the SCPML terms. The material tensors now contain the interpolation matrices and are written as $[\epsilon]$ and $[\mu]$.

From here, two different matrix wave equations can be derived and either can be used to analyze a device. Unlike two-dimensional FDFD, these two wave equations are not calculating two different modes. They are performing the very same analysis, just in terms of different fields. To start, (10.34) is solved for $\vec{\mathbf{h}}$ and (10.35) is solved for $\vec{\mathbf{e}}$.

$$\vec{\mathbf{h}} = [\mu]^{-1} \mathbf{C}^e \vec{\mathbf{e}} \quad (10.42)$$

$$\vec{\mathbf{e}} = [\epsilon]^{-1} \mathbf{C}^h \vec{\mathbf{h}} \quad (10.43)$$

A matrix wave equation in terms of just the electric fields $\vec{\mathbf{e}}$ is derived by substituting (10.42) into (10.35).

$$(\mathbf{C}^h [\mu]^{-1} \mathbf{C}^e - [\epsilon]) \vec{\mathbf{e}} = 0 \quad (10.44)$$

The matrix division by $[\mu]$ in (10.44) can be very slow when calculating the wave matrix, especially when the permeability is anisotropic. To speed computation, it is helpful to describe the device from the beginning in terms of its impermeability $[\psi(x,y,z)]$ instead of its permeability $[\mu(x,y,z)]$. These are related through $[\psi(x,y,z)] = [\mu(x,y,z)]^{-1}$. If for some reason the permeability must be calculated first, the impermeability can be calculated from the permeability using (10.45) [1]. This equation is best implemented in MATLAB when the μ_{mn} terms are arrays.

$$[\psi] = \frac{\begin{bmatrix} \mu_{yy}\mu_{zz} - \mu_{yz}\mu_{zy} & \mu_{xz}\mu_{zy} - \mu_{xy}\mu_{zz} & \mu_{xy}\mu_{yz} - \mu_{xz}\mu_{yy} \\ \mu_{yz}\mu_{zx} - \mu_{yx}\mu_{zz} & \mu_{xx}\mu_{zz} - \mu_{xz}\mu_{zx} & \mu_{xz}\mu_{yx} - \mu_{xx}\mu_{yz} \\ \mu_{yx}\mu_{zy} - \mu_{yy}\mu_{zx} & \mu_{xy}\mu_{zx} - \mu_{xx}\mu_{zy} & \mu_{xx}\mu_{yy} - \mu_{xy}\mu_{yx} \end{bmatrix}}{\mu_{xx}\mu_{yy}\mu_{zz} - \mu_{xx}\mu_{yz}\mu_{zy} - \mu_{xz}\mu_{yy}\mu_{zx} - \mu_{xy}\mu_{yx}\mu_{zz} + \mu_{xy}\mu_{yz}\mu_{zx} + \mu_{xz}\mu_{yx}\mu_{zy}} \quad (10.45)$$

Using impermeability $[\psi(x,y,z)]$, the wave matrix contains only matrix multiplications that are much faster and more efficient to calculate. The revised matrix wave equation is

$$(C^h[\psi]C^e - [\epsilon])\vec{e} = 0 \quad (10.46)$$

Following a similar procedure, a matrix equation in terms of just the magnetic fields \vec{h} is derived by substituting (10.43) into (10.34).

$$(C^e[\epsilon]^{-1}C^h - [\mu])\vec{h} = 0 \quad (10.47)$$

The matrix division in (10.47) is also very slow to calculate, especially when the permittivity is anisotropic. In this case, it is helpful to describe the device in terms of its impermittivity $[\zeta(x,y,z)]$ instead of its permittivity $[\epsilon(x,y,z)]$. These are related through $[\zeta(x,y,z)] = [\epsilon(x,y,z)]^{-1}$. If for some reason the permittivity must be calculated first, the impermittivity can be calculated from the permittivity using (10.48). This equation is best implemented in MATLAB when the ϵ_{mn} terms are arrays.

$$[\zeta] = \frac{\begin{bmatrix} \epsilon_{yy}\epsilon_{zz} - \epsilon_{yz}\epsilon_{zy} & \epsilon_{xz}\epsilon_{zy} - \epsilon_{xy}\epsilon_{zz} & \epsilon_{xy}\epsilon_{yz} - \epsilon_{xz}\epsilon_{yy} \\ \epsilon_{yz}\epsilon_{zx} - \epsilon_{yx}\epsilon_{zz} & \epsilon_{xx}\epsilon_{zz} - \epsilon_{xz}\epsilon_{zx} & \epsilon_{xz}\epsilon_{yx} - \epsilon_{xx}\epsilon_{yz} \\ \epsilon_{yx}\epsilon_{zy} - \epsilon_{yy}\epsilon_{zx} & \epsilon_{xy}\epsilon_{zx} - \epsilon_{xx}\epsilon_{zy} & \epsilon_{xx}\epsilon_{yy} - \epsilon_{xy}\epsilon_{yx} \end{bmatrix}}{\epsilon_{xx}\epsilon_{yy}\epsilon_{zz} - \epsilon_{xx}\epsilon_{yz}\epsilon_{zy} - \epsilon_{xz}\epsilon_{yy}\epsilon_{zx} - \epsilon_{xy}\epsilon_{yx}\epsilon_{zz} + \epsilon_{xy}\epsilon_{yz}\epsilon_{zx} + \epsilon_{xz}\epsilon_{yx}\epsilon_{zy}} \quad (10.48)$$

Using impermittivity $[\zeta(x,y,z)]$, the revised matrix wave equation contains only multiplications.

$$(C^h[\zeta]C^e - [\epsilon])\vec{e} = 0 \quad (10.49)$$

Neither of the matrix wave equations derived above are yet solvable because a source has not been incorporated.

10.2 Incorporating Sources into Three-Dimensional FDFD

The matrix wave equations derived above contain all three vector components of the electric field (or magnetic field). Therefore, all three vector components of the source must be calculated. When the electric field wave equation is being solved, the components of the source field are $E_{x,\text{src}}(x,y,z)$, $E_{y,\text{src}}(x,y,z)$, and $E_{z,\text{src}}(x,y,z)$. It is important to remember that the field components are at physically different locations due to the staggering of the Yee grid. This must be taken into account when calculating the source components because they will all have a slightly different phase. After calculation, the three discrete functions are reshaped into the column vectors $\mathbf{e}_{x,\text{src}}$, $\mathbf{e}_{y,\text{src}}$, and $\mathbf{e}_{z,\text{src}}$ and assembled into a block column vector $\vec{\mathbf{e}}_{\text{src}}$ according to

$$\vec{\mathbf{e}}_{\text{src}} = \begin{bmatrix} \mathbf{e}_{x,\text{src}} \\ \mathbf{e}_{y,\text{src}} \\ \mathbf{e}_{z,\text{src}} \end{bmatrix} \quad (10.50)$$

Each source field component has its own TF/SF regions. \mathbf{Q}_x is the SF masking matrix for $\mathbf{e}_{x,\text{src}}$, \mathbf{Q}_y is the SF masking matrix for $\mathbf{e}_{y,\text{src}}$, and \mathbf{Q}_z is the SF masking matrix for $\mathbf{e}_{z,\text{src}}$. It is standard practice to build one matrix \mathbf{Q}_{xyz} and use this same matrix for all field components ($\mathbf{Q}_x = \mathbf{Q}_y = \mathbf{Q}_z = \mathbf{Q}_{xyz}$). Given these, the overall SF masking matrix \mathbf{Q} is still a diagonal matrix as it was for two-dimensional FDFD.

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}_x & 0 & 0 \\ 0 & \mathbf{Q}_y & 0 \\ 0 & 0 & \mathbf{Q}_z \end{bmatrix} = \begin{bmatrix} \mathbf{Q}_{xyz} & 0 & 0 \\ 0 & \mathbf{Q}_{xyz} & 0 \\ 0 & 0 & \mathbf{Q}_{xyz} \end{bmatrix} \quad (10.51)$$

After calculating the source field $\bar{\mathbf{e}}_{\text{src}}$, the wave matrix \mathbf{A} , and the SF masking matrix \mathbf{Q} , the source vector \mathbf{b} is calculated using the standard QAAQ equation introduced in Chapter 8.

$$\mathbf{b} = (\mathbf{Q}\mathbf{A} - \mathbf{A}\mathbf{Q})\bar{\mathbf{e}}_{\text{src}} \quad (10.52)$$

10.3 Iterative Solution for FDFD

Direct solution of the matrix wave equation $\mathbf{A}\mathbf{f} = \mathbf{b}$ by methods like LU decomposition [4] is by far the simplest and most robust solution method. It is even faster than an iterative solver when the matrices are small. As matrix size grows, however, the efficiency and accuracy degrade and eventually iterative solvers become superior. It is difficult to quantify exactly when iterative solvers become superior, but they are superior for the vast majority of three-dimensional FDFD simulations. The drawback of an iterative solver is that solutions can be slow to calculate and there is no guarantee that an iterative solver will converge to a solution [4–6]. Even when iterative solvers are slower, they can still offer significant memory savings.

In order to optimize the efficiency of an iterative solver, the conditioning of the matrix equation to be solved must be considered [7, 8]. The *condition number* of a matrix \mathbf{A} is a measure of how much the answer to a linear algebra problem changes due to small changes in \mathbf{A} . Matrices with large condition numbers are said to be *ill-conditioned* and are not handled well by iterative solvers. Matrices with small condition numbers are said to be *well-conditioned* and are ideal for iterative solvers. To improve the conditioning of the wave matrix \mathbf{A} in FDFD, the UPML is replaced with the SCPML. Another option is *preconditioning* where a matrix equation $\mathbf{A}\mathbf{f} = \mathbf{b}$ is premultiplied (or postmultiplied, or both) by a preconditioner \mathbf{P} such that the equation $\mathbf{P}^{-1}\mathbf{A}\mathbf{f} = \mathbf{P}^{-1}\mathbf{b}$ is better conditioned [5, 6, 8]. The ideal preconditioner transforms \mathbf{A} into the identity matrix so that only a single iteration is required to solve the problem. A simple preconditioner that works well for FDFD is the Jacobi preconditioner where \mathbf{P} is the diagonal of the wave matrix [8].

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{f} = \mathbf{P}^{-1}\mathbf{b} \quad (10.53)$$

$$\mathbf{P} = \text{diag}(\mathbf{A}) = \begin{bmatrix} A_{11} & 0 & 0 \\ 0 & A_{22} & \\ 0 & & \ddots & 0 \\ & & 0 & A_{MM} \end{bmatrix} \quad (10.54)$$

Given adequate conditioning, MATLAB offers a variety of options for solving matrix equations iteratively. The quasi-minimal residual method `qmr()` seems to be a slower iterative solver but more robust, whereas the biconjugate gradient method `bicg()` is faster but less robust. The *robustness* of a solver is its dependability to converge to a solution despite the conditioning or other properties that could make a matrix equation difficult to solve. Experiment with the various solvers in MATLAB to see what works best for your simulation. Representative MATLAB code to solve the FDFD matrix equation $\mathbf{A}\mathbf{f} = \mathbf{b}$ using the `qmr()` solver is

```
tol    = 1e-10;
maxit  = 15000;
f0     = zeros(M,1); %M is number of rows in A
f      = qmr(A,b,tol,maxit,[],[],f0);
```

In the above code, `tol` is the tolerance that controls how well converged the final answer should be. The variable `maxit` sets the maximum number of iterations in case the solver is failing to converge to a solution. The number of iterations required to solve an FDFD problem can range from several hundred to many thousands, depending on the size and complexity of the simulation. `f0` is an initial guess for the solution. Since the solution is completely unknown, it is set to all zeros in the code above. When performing parameter sweeps, there may be some benefit to seeding the current simulation with the results from the previous if the changes to the problem are small from one step to the next in the sweep. The last line in the code above solves the matrix equation using the `qmr()` solver.

If a graphical processing unit (GPU) is available to MATLAB, significant speed gains can be achieved by performing the iteration on the GPU. MATLAB makes this as easy as assigning the `A` and `b` terms to the GPU, calling the iterative solver, and then gathering the solution `f` back from the GPU after the solution is obtained. When the arrays are assigned to the GPU, the iterative solver in MATLAB automatically knows to solve them on the GPU. Representative code to solve $\mathbf{A}\mathbf{f} = \mathbf{b}$ iteratively on a GPU in MATLAB is

```
tol    = 1e-10;
maxit  = 15000;
f0     = zeros(M,1); %M is number of rows in A
A      = gpuArray(A);
b      = gpuArray(B);
f      = qmr(A,b,tol,maxit,[],[],f0);
f      = gather(f);
```

10.4 Calculating Reflection and Transmission for Doubly Periodic Structures

When a structure is periodic in both the x - and y -directions, the structure can be analyzed as a crossed grating as discussed in Chapter 2. Calculating reflection and transmission will be performed through the diffraction orders. It is a straightforward extension of how this calculation was done for two-dimensional simulations in Chapter 8. For three-dimensional simulations, the source wave vector \vec{k}_{inc} is described by the angles θ_{inc} and ϕ_{inc} as

$$k_{x,\text{inc}} = k_0 n_{\text{ref}} \sin \theta_{\text{inc}} \cos \phi_{\text{inc}} \quad (10.55)$$

$$k_{y,\text{inc}} = k_0 n_{\text{ref}} \sin \theta_{\text{inc}} \sin \phi_{\text{inc}} \quad (10.56)$$

$$k_{z,\text{inc}} = k_0 n_{\text{ref}} \cos \theta_{\text{inc}} \quad (10.57)$$

When the incident wave encounters a periodic structure, it leads to an infinite expansion for both x and y components of the wave vectors associated with the diffraction orders. Boundary conditions require that this expansion is the same for both reflected and transmitted diffraction orders so

$$k_x(m, n) = k_{x,\text{ref}}(m, n) = k_{x,\text{trn}}(m, n) = k_{x,\text{inc}} - m \frac{2\pi}{\Lambda_x} \quad (10.58)$$

$$k_y(m, n) = k_{y,\text{ref}}(m, n) = k_{y,\text{trn}}(m, n) = k_{y,\text{inc}} - n \frac{2\pi}{\Lambda_y} \quad (10.59)$$

In these equations, Λ_x is the period of the structure in the x -direction, Λ_y is the period of the structure in the y -direction, and m and n are the diffraction order numbers. The longitudinal components of the wave vectors associated with the diffraction orders may be different on the reflection side and transmission side because the medium can be different on each side. For this reason, they must be calculated separately from the dispersion relation written for both sides.

$$k_{z,\text{ref}}(m, n) = -\sqrt{(k_0 n_{\text{ref}})^2 - k_x^2(m, n) - k_y^2(m, n)} \quad (10.60)$$

$$k_{z,\text{trn}}(m, n) = \sqrt{(k_0 n_{\text{trn}})^2 - k_x^2(m, n) - k_y^2(m, n)} \quad (10.61)$$

When the simulation is finished, the field is analyzed to calculate the complex amplitudes of the diffraction orders in the same three steps it was for two-dimensional simulations. First, the fields are extracted from the cross sections of the grid at both the reflection and transmission planes. These are $\langle E_{x,\text{ref}}(x, y) \rangle$, $\langle E_{y,\text{ref}}(x, y) \rangle$, $\langle E_{z,\text{ref}}(x, y) \rangle$, $\langle E_{x,\text{trn}}(x, y) \rangle$, $\langle E_{y,\text{trn}}(x, y) \rangle$ and $\langle E_{z,\text{trn}}(x, y) \rangle$. The x , y , and z field components reside at different locations on the grid so they must be interpolated at a common point. The angled-bracket operation $\langle \rangle$ represents the field components

interpolated at the origins of the Yee cells. Second, the phase tilt due to an oblique angle of incidence is removed to isolate the envelope term of the Bloch modes. A cross section of the source wave will be used as the phase tilt in this calculation so that $\exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)]$ does not have to be calculated again. The envelope functions are calculated according to

$$\begin{aligned} a_{x,\text{ref}}(x, y) &= \langle E_{x,\text{ref}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \\ a_{y,\text{ref}}(x, y) &= \langle E_{y,\text{ref}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \\ a_{z,\text{ref}}(x, y) &= \langle E_{z,\text{ref}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \end{aligned} \quad (10.62)$$

$$\begin{aligned} a_{x,\text{trn}}(x, y) &= \langle E_{x,\text{trn}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \\ a_{y,\text{trn}}(x, y) &= \langle E_{y,\text{trn}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \\ a_{z,\text{trn}}(x, y) &= \langle E_{z,\text{trn}}(x, y) \rangle \div \exp[-j(k_{x,\text{inc}}x + k_{y,\text{inc}}y)] \end{aligned} \quad (10.63)$$

Third, the complex amplitudes of the diffraction orders are calculated from the discrete Fourier transform of the amplitude functions calculated in (10.62) and (10.63). Using a two-dimensional fast Fourier transform (FFT) algorithm, these are calculated as

$$\begin{aligned} E_{x0,\text{ref}}(m, n) &= \text{FFT}_{2\text{D}}[a_{x,\text{ref}}(x, y)] \\ E_{y0,\text{ref}}(m, n) &= \text{FFT}_{2\text{D}}[a_{y,\text{ref}}(x, y)] \\ E_{z0,\text{ref}}(m, n) &= \text{FFT}_{2\text{D}}[a_{z,\text{ref}}(x, y)] \end{aligned} \quad (10.64)$$

$$\begin{aligned} E_{x0,\text{trn}}(m, n) &= \text{FFT}_{2\text{D}}[a_{x,\text{trn}}(x, y)] \\ E_{y0,\text{trn}}(m, n) &= \text{FFT}_{2\text{D}}[a_{y,\text{trn}}(x, y)] \\ E_{z0,\text{trn}}(m, n) &= \text{FFT}_{2\text{D}}[a_{z,\text{trn}}(x, y)] \end{aligned} \quad (10.65)$$

Given the complex amplitudes of the diffraction orders, the diffraction efficiencies are calculated using the equations derived in Chapter 2. These are

$$R_{\text{DE}}(m, n) = \frac{|E_{0,\text{ref}}(m, n)|^2}{|E_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m, n)}{k_{z,\text{inc}}} \right] \quad (10.66)$$

$$T_{\text{DE}}(m, n) = \frac{|E_{0,\text{trn}}(m, n)|^2}{|E_{0,\text{inc}}|^2} \text{Re} \left[\frac{\mu_{r,\text{ref}}}{\mu_{r,\text{trn}}} \frac{k_{z,\text{trn}}(m, n)}{k_{z,\text{inc}}} \right] \quad (10.67)$$

Equations (10.66) and (10.67) are used when it is the electric field being solved by FDFD. If instead it is the magnetic field being solved, the diffraction efficiency calculations are

$$R_{\text{DE}}(m, n) = \frac{|\tilde{H}_{0,\text{ref}}(m, n)|^2}{|\tilde{H}_{0,\text{inc}}|^2} \text{Re} \left[-\frac{k_{z,\text{ref}}(m, n)}{k_{z,\text{inc}}} \right] \quad (10.68)$$

$$T_{\text{DE}}(m, n) = \frac{|\tilde{H}_{0,\text{trn}}(m, n)|^2}{|\tilde{H}_{0,\text{inc}}|^2} \text{Re} \left[\frac{\epsilon_{r,\text{ref}} k_{z,\text{trn}}(m, n)}{\epsilon_{r,\text{trn}} k_{z,\text{inc}}} \right] \quad (10.69)$$

10.5 Implementation of Three-Dimensional FDFD and Examples

10.5.1 Standard Sequence of Simulations for a Newly Written Three-Dimensional FDFD Code

Before any new three-dimensional devices are simulated, there should be a standard sequence of simulations performed to identify and troubleshoot problems with a newly written code. The recommended sequence for three-dimensional simulations follows essentially the same sequence used for two-dimensional simulations. This sequence is illustrated in Figure 10.1 where PBCs are used at the x - and y -axis boundaries to simulate a periodic structure. The first simulation is a complete vacuum with the TF/SF interface positioned at the center of the grid vertically launching a linearly polarized wave at normal incidence, as depicted in Figure 10.1(a). The TF/SF interface is centered vertically to better see if there are any backward waves or reflections that should not be present. It would be more difficult to see if the TF/SF interface were tight against the top PML. If the FDFD code is working correctly, the simulation will calculate 100% transmittance, 0% reflectance, and no wave should be visible above the TF/SF interface. A correct simulation for this case is shown in Figure 10.2(a) for a linear polarization (LP). The specific values for reflectance and transmittance should be within a fraction of a percent. At $N_{\text{RES}}=20$, a working three-dimensional FDFD simulation may calculate numbers like $R = 4.9 \times 10^{-6}$ and $T = 0.996$. If a correct simulation is achieved, a different LP should be tried. If anything incorrect is detected with such a simple simulation, the mistake will be easier to find, and better educated guesses for values of intermediate parameters can be made.

Next, an angle of incidence is incorporated along with a circular polarization to excite all field components at once. In Figure 10.2(b), the angles were set to $\theta = \phi = 30^\circ$. This is mostly to test that the PBCs are working correctly. The source wave should be visible in the TF region below the TF/SF interface and no wave should be visible in the SF region above the TF/SF interface. If there are any mistakes in the code, the source wave may not look like a uniform plane wave in the TF region below the TF/SF interface, and waves may be observed in the SF region above the TF/SF interface. This was discussed in Chapter 8. Keep an open mind when troubleshooting.

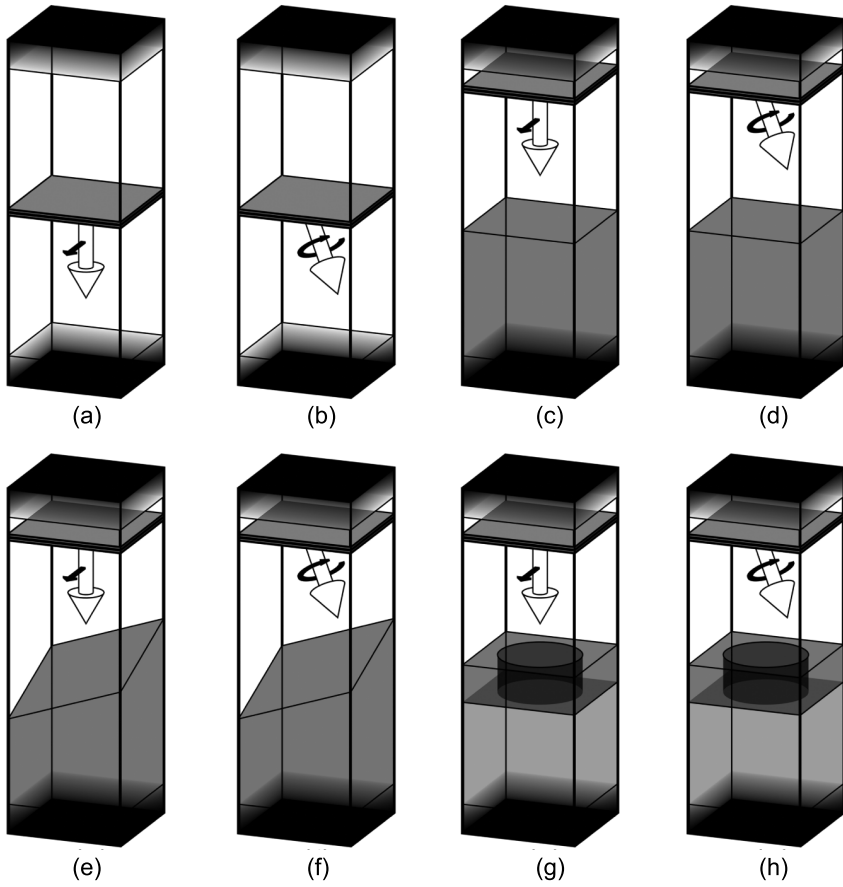


Figure 10.1 Standard sequence of simulations for a newly written 3D FDFD code. (a) and (b) Simulations in a vacuum. (c) and (d) Simulations of a single interface. (e) and (f) Simulations of an asymmetric diffraction grating. (g) and (h) Simulations of a guided-mode resonance filter (GMRF).

After the FDFD code can simulate vacuum correctly, the next thing is to move the TF/SF interface five or so cells below the top PML and then build a single material interface onto the grid. This is illustrated in Figure 10.1(c) that also shows an LP wave at normal incidence. In this case, the Fresnel equations discussed in Chapter 2 can be used to verify the simulation results are correct. First, fill the bottom half of the grid with $\epsilon_r = 9.0$ and $\mu_r = 1.0$. For normal incidence, this causes exactly 25% reflectance and 75% transmittance. The most common mistake here is not calculating the diffraction efficiency of the transmitted diffraction orders correctly because that equation contains extra terms that are easily missed. After this, swap the permittivity and permeability such that $\epsilon_r = 1.0$ and $\mu_r = 9.0$ and ensure the simulation still gives 25% reflectance and 75% transmittance. Next, set $\epsilon_r = \mu_r = 3.0$ and verify reflectance is near 0% and transmittance is near 100%. This case produces no reflections because the impedance is constant throughout the grid.

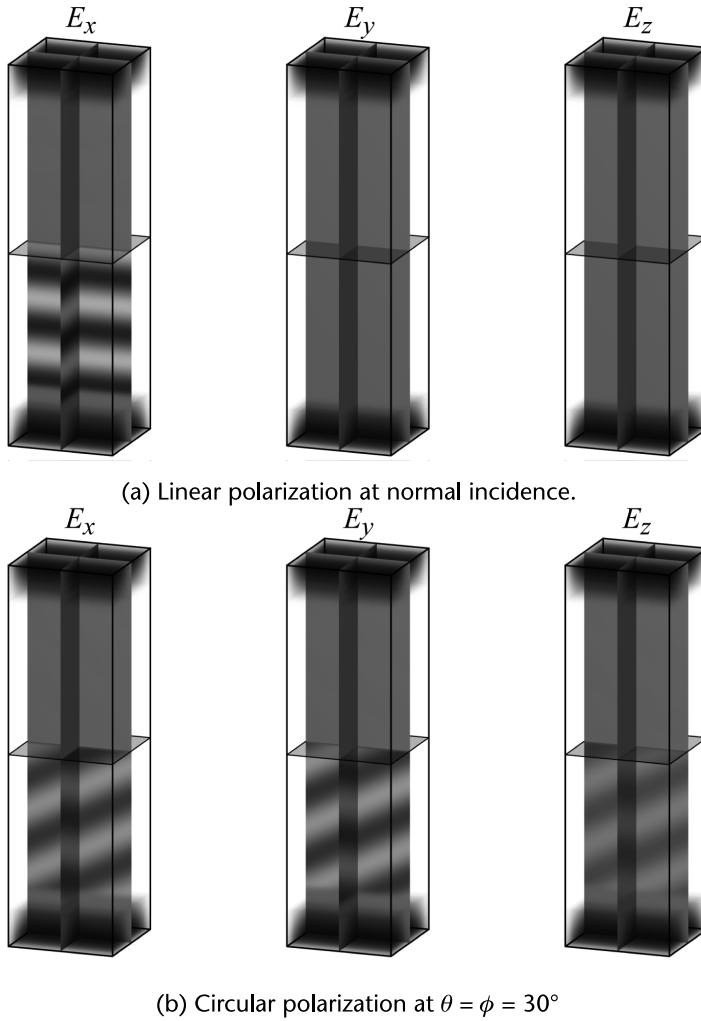


Figure 10.2 (a) Correct simulation where source wave is linearly polarized and at normal incidence. (b) Correct simulation where source wave is circularly polarized and incident at $\theta = \phi = 30^\circ$.

Next, incorporate an angle of incidence along with circular polarization and repeat various combinations of ϵ_r and μ_r , as illustrated in Figure 10.1(d). Using the Fresnel equations, an infinite number of simple simulations are possible. Simulate a few simple cases and then move on.

Next, it is best to simulate an asymmetric diffraction grating. An asymmetric diffraction grating is illustrated in Figure 10.1(e) and is an excellent device to verify that the diffraction orders are being handled correctly in the FDFD code. The same asymmetric diffraction grating simulated in Chapter 8 can be repeated for three-dimensional simulations as well. The grating can be rotated by 90° to verify the diffraction orders are being handled correctly in the x - and y - directions independently. The same asymmetric diffraction grating is simulated again, but with an angle of incidence incorporated, as illustrated in Figure 10.1(f). The last simulation for testing new codes is a wavelength (or frequency) sweep of a GMRF,

as illustrated in Figure 10.1(g) and (h). GMRFs are extremely sensitive devices. If anything is wrong in the code, a GMRF will tend to amplify the problem so that even small mistakes can be identified and corrected. This type of simulation will be performed in Section 10.5.3 for a crossed-grating GMRF.

10.5.2 Generic Three-Dimensional FDFD Function to Simulate Periodic Structures

In Chapter 9, it was argued that it is easiest to perform parameter sweeps by creating a generic FDFD function for simulating periodic structures. This will be done for three-dimensional FDFD as well. This new function will be called `fd3d()` and will be programmed to simulate virtually any periodic structure. The overall grid strategy for simulating three-dimensional periodic structures that this function will assume is illustrated in Figure 10.3. This is the same basic grid strategy discussed in Chapter 9. PMLs are located at the z -axis boundaries to absorb outgoing waves. PBCs are used at the x - and y -axis boundaries to model an infinitely periodic array. Immediately below the top PML are the two reflection planes where the reflected waves will be analyzed. Two adjacent planes are needed because the z component of the field will have to be interpolated between two values in the z -direction. In fact, all of the field components will be interpolated at the origin of the Yee cells so they can be used together to calculate reflection and transmission. Immediately below this is the TF/SF interface where the source is injected in the $+z$ -direction. Immediately below this

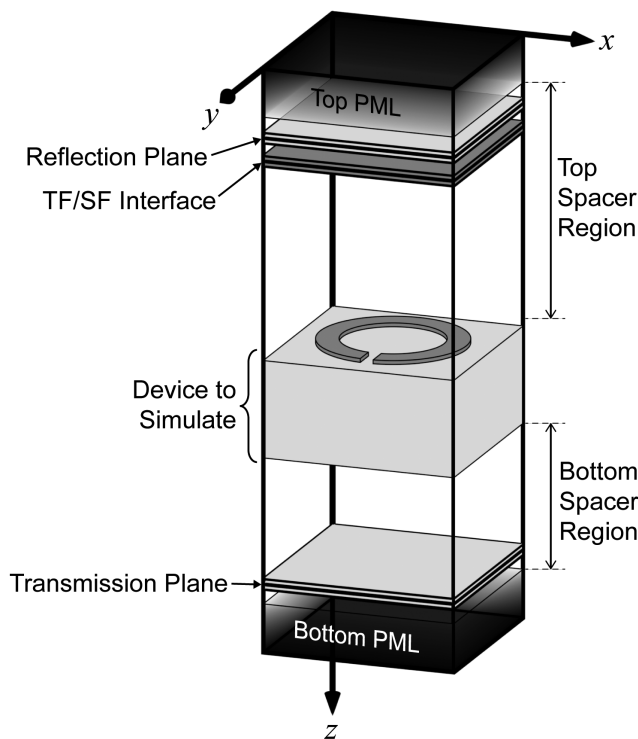


Figure 10.3 Grid strategy assumed by the `fd3d()` function for simulating periodic structures with FDFD.

above the bottom PML are the transmission planes where the transmitted waves will be analyzed. Two planes are needed because the z component of the field will have to be interpolated between two values in the z -direction. The device itself is located near the center of the grid vertically with spacer regions placed above and below the device. It is up to the function calling the `fd3d()` function to add the spacer regions.

The MATLAB code for the generic `fd3d()` function can be downloaded at <https://empossible.net/fdfdbook/>. The header extends from lines 2 to 41 and is what is displayed in the command window if “`help fd3d`” is typed at the command prompt. Lines 47 and 48 define the tolerance and a maximum number of iterations that should be allowed when attempting to obtain a solution by iteration. Line 48 defines an anonymous function that forms a diagonal matrix from an array.

The input arguments are processed from lines 53 to 133. There are two ways the program that calls `fd3d()` can define the device to be simulated. The first option is to build the device into the isotropic $2 \times$ grid arrays `DEV.ER2` and `DEV.UR2`. The second option is to define all of the tensor elements for the permittivity and/or the permeability. Lines 57 to 62 take this option into account to determine the size of the grid. If the field `ER2` exists in the structure `DEV`, the size of the grid is determined from the size of `DEV.ER2`. If the field `ER2` does not exist in the structure `DEV`, the size of the grid is determined from the size of `DEV.ER2xx`. Lines 64 to 73 calculate the rest of the grid parameters. Lines 75 to 79 calculate the axis arrays and the meshgrid for the $2 \times$ grid because the meshgrid is needed to calculate the source field. It is standard practice to center the x - and y -axis arrays at zero, as is done on lines 76 and 77. The z -axis array calculated on line 78 is not centered this way. Line 82 calculates the free space wavenumber k_0 from the free space wavelength `SRC.lam0`. Lines 84 to 87 calculate the matrix size `M` as well as the zeros `ZZ` and identity matrices `I`. Lines 89 to 110 form the diagonal materials matrices for all of the tensor elements of the relative permittivity. If the materials are specified in `DEV.ER2`, the tensor is defined as diagonally anisotropic entirely from `ER2`. Otherwise, the tensor elements are specified and are individually diagonalized. This is repeated for the relative permeability from lines 112 to 133.

The FDFD algorithm is implemented from lines 135 to 242. It starts by extracting the material properties from the top and bottom parts of the grid on lines 139 to 145. Lines 147 to 162 calculate the PML terms and form the diagonal matrices that incorporate the SCPML into the curl matrices. Lines 164 to 173 calculate the wave vectors needed for the PBCs and for calculating reflection and transmission. The incident wave vector is calculated using (10.55) to (10.57). Lines 175 to 179 call the `yee3d()` function to build the derivative matrices for the three-dimensional Yee grid. Lines 181 to 184 calculate the interpolation matrices directly from the derivative matrices using (10.30). Now that the interpolation matrices are calculated, matrices are constructed for the materials tensors according to (10.40) and (10.41) on lines 186 to 192. The curl matrices are calculated from lines 194 to 201 using (10.38) and (10.39). This lets the wave matrix `A` be calculated on line 201.

The next few sections of code lead up to building the source vector `b`. Lines 203 to 213 calculate the electric field polarization vector `P` following Section 2.7

in Chapter 2. From the polarization vector, lines 215 to 220 calculate all three vector components of the source field and assemble them into the column vector `fsrc`. Observe how these calculations are implemented using the $2\times$ grid so that the phase and positions of the field components on the Yee grid are accounted for. Lines 222 to 227 build the SF masking matrix `0`, where the SF region is made to cover the PML at the top of the grid plus an additional three cells. The three cells are to leave room for analyzing the reflected waves. Given all of this, the source vector `b` is calculated on line 230 using the QAAQ equation.

All of this runs relatively quickly. Lines 232 to 237 are where the field is solved and is the most computationally intensive step in the `fdfd3d()` function. Here, the `A` and `b` terms are assigned to a GPU for faster calculation. The iterative solver `bicg()` recognizes the matrices are stored on the GPU and the iterative algorithm is performed on the GPU. If no GPU is present, lines 234, 235, and 237 should be deleted and the iterative solver will run on the CPU. After the field is solved, the x , y , and z components of the field throughout the entire grid are extracted from the solution on lines 239 to 242. The spaces added in lines 240 and 241 are ignored by MATLAB and are inserted simply to align the equations to have clean code.

At this point, the FDFD simulation is finished and the rest of the `fdfd3d()` function postprocesses the fields to calculate reflection and transmission. This occurs on lines 244 to 309. Lines 248 to 257 extract the fields from the top and bottom of the grid, just outside of the SCPML regions. Observe that the z components of the fields are extracted from two slices through the grid. The fields are staggered on the Yee grid so in order to combine them for calculations they must be interpolated to common points on the grid. For this purpose, the origins of the Yee cells are chosen. Lines 259 to 278 perform these interpolations. Lines 280 to 287 remove the phase tilt from the fields following (10.62) and (10.63). This is done because it is the envelope term that should be Fourier transformed following (10.64) and (10.65). Lines 289 to 300 perform the Fourier transforms to calculate the complex amplitudes of the diffraction orders. Observe that the results from the FFTs are shifted using `fftshift()` to move the zero-order term to the center of the array, and then divided by the total number of points $N_x \times N_y$ in the FFT to make the values true Fourier coefficients. Lines 294 and 300 calculate the output arrays `DAT.s11` and `DAT.s21` that contain the complex reflection and transmission coefficients of the diffraction orders. These terms will be used in the parameter retrieval described in Section 10.5.5 but are useful any time the amplitude and phase of the diffraction orders are of interest. Lines 302 and 304 calculate the diffraction efficiencies of all the diffraction orders using (10.66) and (10.67). These are summed on lines 307 and 308 to calculate the overall reflectance and transmittance.

10.5.3 Simulation of a Crossed-Grating GMRF

To demonstrate three-dimensional FDFD and parameter sweeps at the same time, the crossed-grating GMRF illustrated in Figure 10.4 will be simulated. The left part of this figure illustrates the GMRF, the source wave, the reflected wave, the transmitted wave, and a box formed around a single unit cell of the device. Using PBCs, it is only necessary to model a single unit cell on the FDFD grid. This single

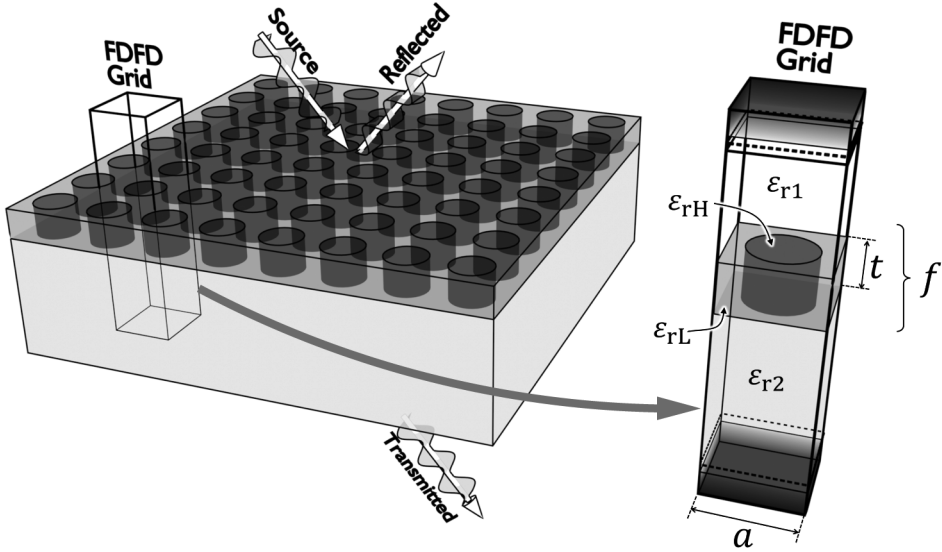


Figure 10.4 Crossed-grating GMRF and how it is represented on an FDFD grid. This device has $a = 3.7$ cm, $t = 1.5$ cm, $f = 0.5$, $\epsilon_{r1} = 1.0$, $\epsilon_{r2} = 2.0$, $\epsilon_{rL} = 3.0$, and $\epsilon_{rH} = 5.0$.

unit cell is illustrated in the right part of Figure 10.4. The GMRF is composed of a core region of relative permittivity $\epsilon_{rL} = 3.0$ that hosts a periodic array of holes filled with relative permittivity $\epsilon_{rH} = 5.0$. The relative permittivity above the crossed grating is $\epsilon_{r1} = 1.0$ and below the crossed-grating is $\epsilon_{r2} = 2.0$. Both mediums above and below the crossed-grating are assumed to be semi-infinite. The thickness of the grating layer is $t = 1.5$ cm and the radius r of the holes is chosen so that they occupy a fraction of the area defined by f . The equation to calculate r from f is

$$r = a\sqrt{\frac{f}{\pi}} \quad (10.70)$$

It is desired to simulate this device using FDFD to calculate and plot its reflectance and transmittance from 4.5 to 5.5 GHz. The device is illuminated with a right circular polarized (RCP) wave at normal incidence. To perform a parameter sweep, the function `fd3d()` will be used so the only tasks left for the main program are calculating the grid, building the device, and controlling the parameter sweep. The generic function makes parameter sweeps very easy!

The MATLAB code that performs the parameter sweep of the GMRF can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter10_GMRF.m`. The header extends from lines 1 to 25 and includes initializing MATLAB, defining the units, and defining the constants. The all-important dashboard extends from lines 27 to 56 and defines all of the parameters that control the simulation. The source parameters are defined on lines 31 to 39. A frequency sweep is defined from 4.5 to 5.5 GHz with 500 frequency points used in the array `FREQ`. It is common to use many frequency points when simulating GMRFs because these devices tend to exhibit very narrowband and abrupt spectral behavior. The source is an RCP wave at normal incidence so lines 38 and 39 calculate the transverse electric (TE) and

transverse magnetic (TM) components. These are made to have equal amplitude but a factor of $1i$ was included to make the components 90° out of phase. Lines 41 to 48 define all of the material properties and dimensions of the GMRF as illustrated in Figure 10.4. Lines 50 to 56 define the grid parameters. `NRES` defines the number of grid cells to resolve the minimum wavelength. `DEV.NPML` is an array of two numbers that define the size of the SCPML at the z -axis boundaries of the grid. `ermax` is the maximum relative permittivity found anywhere on the grid and is calculated by determining the maximum value of the relative permittivity terms defining the GMRF. From this, `nmax` is the maximum refractive index anywhere on the grid and is calculated from `ermax` as `sqrt(ermax)`. `lam_max` is the maximum wavelength in the simulation and is calculated as the speed of light divided by the minimum frequency in the simulation. `SPACER` is an array of two numbers specifying the size of the regions between the device and the PMLs above and below. It was found from trial-and-error that one wavelength of the maximum wavelength was the minimum amount of space needed for this simulation. The guided mode in a dielectric GMRF can extend outside of the device. The evanescent field from a device should not be allowed to touch the PML or it will excite a propagating wave inside of the PML that provides an escape path for power that is not accounted for by the simulation.

The grid is calculated from lines 58 to 101. For the x - and y -directions, the grid resolution was snapped to the period of the GMRF. In the z -direction, the grid resolution was snapped to the height of the core of the GMRF. Lines 103 to 120 build the materials arrays that define the GMRF on the $2\times$ grid. Since PBCs are used, only a single unit cell has to be constructed onto the grid. Lines 107 to 109 initialize the relative permittivity array `DEV.ER2` and the relative permeability array `DEV.UR2` to all ones to represent air. The grating is added throughout the entire $2\times$ grid on lines 111 to 114. Line 112 calculates the radius r of the hole from the fill factor f using (10.70). Lines 116 to 120 overwrite the cells above and below the actual grating layer with the correct values for relative permittivity. The entire grid above the grating is filled with `er1` while the entire grid below the grating is filled with `er2`. Observe that the centering algorithm was not used here to position the grating layer because that would not be correct if the two spacer regions were set to different sizes. Instead, the starting index of the grating layer is set to a position below the top PML and below the top spacer region. The number of cells in the top PML was multiplied by two because the GMRF is being constructed on the $2\times$ grid that contains twice as many cells.

The frequency sweep is performed from lines 122 to 174. Line 127 defines the last remaining field of `DEV`, which is `DEV.RES` that stores the grid resolution parameters `dx`, `dy`, and `dz`. Lines 129 to 132 initialize the arrays that will store the reflectance `REF`, transmittance `TRN`, and power conservation `CON` calculated during the sweep. The loop for the actual frequency sweep extends from line 137 to 174. The first step in the loop is on lines 139 to 141 where the next frequency to simulate is grabbed from the array `FREQ`. From this, the free space wavelength `SRC.lam0` is set to the speed of light divided by the frequency. Line 144 calls the `fdfd3d()` function that performs the simulation. This function was described in detail in Section 10.5.2. When the simulation for the current iteration finishes, lines 146 to 149 record the response in the arrays `REF`, `TRN`, and `CON`. Lines 151 to 159 visualize the field as it is being calculated. Lines 161 to 172 plot the response as the simulation is running.

Visualizing the fields and spectral response while the parameter sweep is being calculated is very good practice. Many times when problems with the code arise during the sweep, they can be caught early without having to wait for the entire sweep to be completed.

The reflectance, transmittance, and power conservation of the GMRF are plotted in Figure 10.5(b). Convergence for this device was found to start at around $N_{RES}=20$. The device exhibits relatively low background reflection with two narrow regions of very high reflection. These occur at frequencies where the grating couples the applied wave into guided modes. From the spectrum, these resonances are observed around 4.9 and 5.2 GHz. The field is visualized at 4.9 GHz in Figure 10.5(a) using MATLAB's `slice()` function along with a wireframe image of the crossed grating. A guided mode is clearly observed.

10.5.4 Simulation of a Frequency Selective Surface

Planar periodic structures are very common in electromagnetics and they include frequency selective surfaces (FSSs) [9–11], metasurfaces [12–19], array antennas [20, 21], artificial ground planes [22], and more. FSSs are planar structures that are designed to reflect, transmit, or absorb different frequencies of electromagnetic waves. In this section, the FSS shown in Figure 10.6 will be simulated using almost identical code to that discussed in Section 10.5.2. In fact, all of the periodic structures mentioned above are simulated the same way.

The element chosen for this FSS is a Jerusalem cross [23–25] because it is a very well-known element and is good practice to build onto the Yee grid. The geometry of

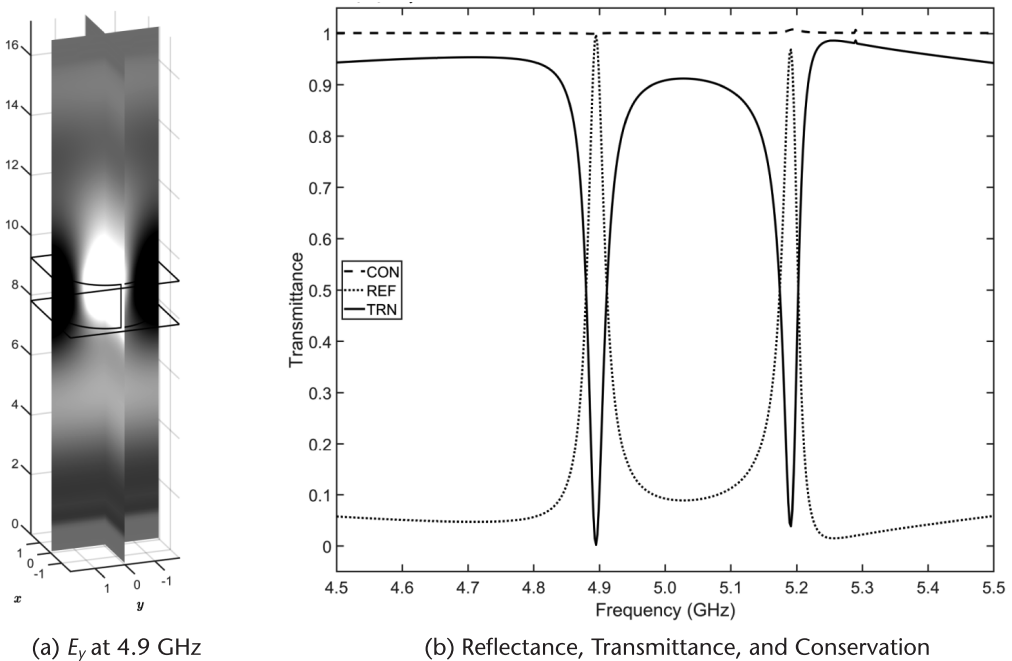


Figure 10.5 (a) Electric field component E_y visualized on resonance at 4.9 GHz. (b) Transmittance, reflectance, and power conservation response for the GMRF.

the Jerusalem cross is shown in Figure 10.7. This FSS will use acrylonitrile butadiene styrene (ABS) plastic as the substrate material and perfect electric conductors (PECs) for the metal elements. It is possible to modify the matrix equation $\mathbf{A}\mathbf{f} = \mathbf{b}$ in order

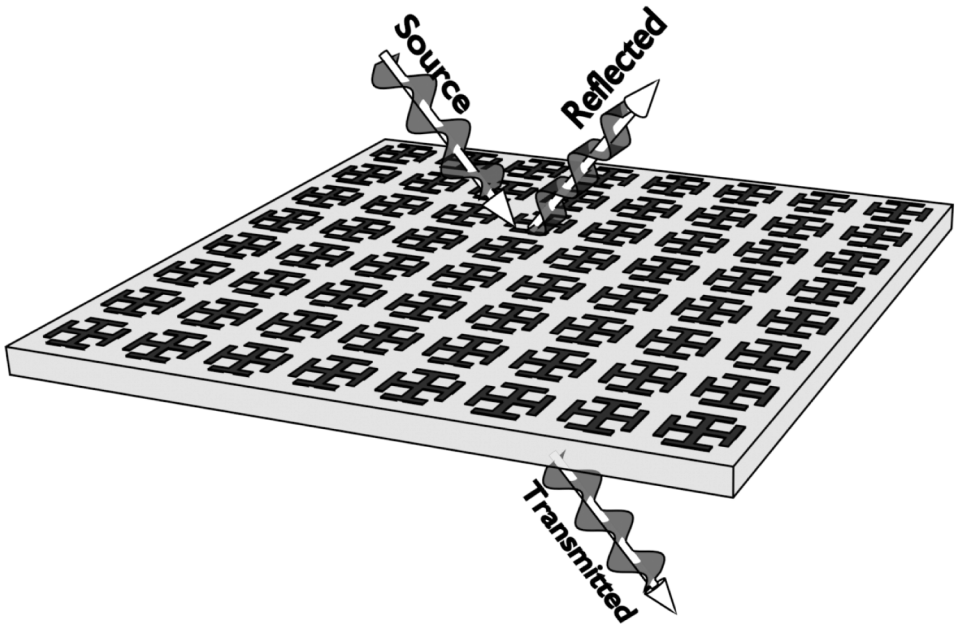


Figure 10.6 Frequency selective surface composed of a periodic array of Jerusalem cross elements.

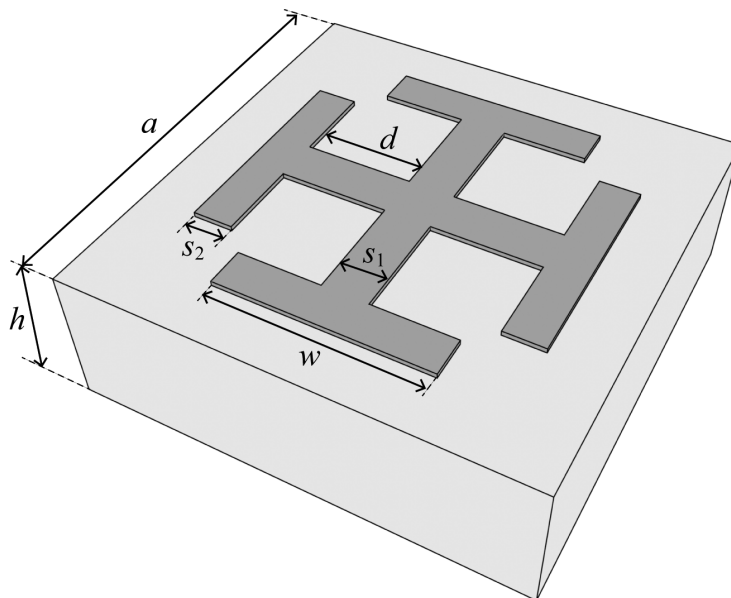


Figure 10.7 Geometry of the frequency selective surface element. This design has $a = 9.95$ mm, $h = 3.16$ mm, $s_1 = 1.0$ mm, $s_2 = 0.9$ mm, $w = 4.5$ mm, and $d = 2.32$ mm. The substrate is ABS plastic with $\epsilon_r = 2.5$.

to force the electric fields in the metals to be exactly zero to implement the PEC conductors. However, it is much easier to simply set the relative permittivity to a very large value, like $\epsilon_{r,PEC} \approx 10^6$, where the metals are located on the grid. The specific dimensions for the element are given in the figure caption.

The MATLAB code to simulate the FSS can be downloaded at <https://empossible.net/fdfdbook/> and is called `Chapter10_FSS.m`. The program is very similar to the code for the crossed grating GMRF. The header extends from lines 1 to 25 and includes initializing MATLAB, defining the units, and defining the constants to be used in the simulation. The all-important dashboard extends from lines 27 to 58 and defines all of the parameters that control the simulation. The source parameters are defined on lines 31 to 39. A frequency sweep is defined from 8.0 to 12.0 GHz with 50 frequency points in the array `FREQ`. The source is a TE-polarized wave and observe that `SRC.theta` is set to 30° and `SRC.phi` is set to 60° in order to practice simulating waves at an oblique angle of incidence. Lines 41 to 47 define all of the dimensions of the FSS element as described in Figure 10.7. The material properties are defined on lines 49 and 50. The dielectric constant of the ABS plastic substrate is `erd=2.5` and the dielectric constant of the metal is set to `erm=1e6`. This simulation could just as easily have accounted for the loss in the materials through a complex permittivity. Lines 52 to 58 define the grid parameters. `NRES` defines the number of grid cells to resolve the minimum wavelength. Convergence was found at around `NRES=80`, but this simulation was performed at `NRES=100`. `DEV.NPML` is an array of two numbers that define the size of the SCPML at the z -axis boundaries of the grid. `nmax` is the maximum refractive index found anywhere on the grid and was simply set to `sqrt(erd)` since the ABS plastic substrate is the only dielectric in this simulation other than air. `lam_max` is the maximum wavelength in the simulation and is calculated as the speed of light divided by the minimum frequency in the simulation. `SPACER` is an array of two numbers specifying the size of the spacer regions between the device and PMLs. It was found from trial-and-error that 0.2 of the largest wavelength was sufficient.

The grid is calculated from lines 60 to 107. For the x - and y -directions, the grid resolution was snapped to the period of the FSS. In the z -direction, the grid resolution was snapped to the thickness of the substrate. Lines 109 to 154 build the material arrays that define the FSS on the $2\times$ grid. Since PBCs are used, only a single unit cell has to be constructed onto the grid. Lines 113 to 115 initialize the relative permittivity array `DEV.ER2` and the relative permeability array `DEV.UR2` to all ones to represent air. The substrate is added on lines 117 to 120. Observe that the centering algorithm was not used here because that would not be correct if the two spacer regions were set to different sizes. Instead, the starting index of the substrate `nz1` was to be at a position below the top PML and below the top spacer region. The number of cells in the top PML was multiplied by two because the FSS is being constructed on the $2\times$ grid that contains twice as many cells. Also, observe that the argument of the `round()` operation is divided by two and then multiplied by two on the outside of the `round()` operation. This ensures the resulting integer is an even number. When the value of 1 is added to this even number, the starting index `nz1` becomes an odd number. This was done to place the tangential field components as the first field components in the ABS plastic. The concept of using

odd and even array indices to place metals onto the Yee grid is discussed in the Appendix. The FSS element is constructed onto the grid in four steps from lines 122 to 151. First, lines 123 to 132 calculate the physical positions of the edges of the FSS element working left to right. Lines 134 to 141 calculate the array indices on the $2\times$ grid of these physical positions. Lines 143 to 149 populate an array `ER2` with zeros and ones in the pattern of the FSS element. Line 151 converts the zeros and ones to the actual relative permittivity values that need to be placed onto the grid. The FSS element is added to the `DEV.ER2` array on line 154.

The frequency sweep is performed from lines 156 to 207. Line 161 defines the last remaining field of `DEV`, which is `DEV.RES`, that stores the grid resolution parameters `dx`, `dy`, and `dz`. Lines 163 to 166 initialize the arrays that will store the reflectance `REF`, transmittance `TRN`, and power conservation `CON` calculated during the sweep. The loop for the actual frequency sweep extends from lines 168 to 207. The first step in the loop is on lines 173 to 175 where the next frequency to simulate is grabbed from the array `FREQ`. From this, the free space wavelength `SRC.lam0` is set to the speed of light divided by the frequency. Line 178 calls the `fd3d3d()` function that performs the simulation at the current frequency. The `fd3d3d()` function is described in detail in Section 10.5.2. When the simulation finishes, lines 180 to 183 record the response in the arrays `REF`, `TRN`, and `CON`. Lines 185 to 205 visualize the response as the simulation is running. This is a very good practice because many problems arising during the sweep can be caught early on without having to wait for the entire sweep to be completed. Line 200 calculates the transmittance on a decibel (dB) scale. This is a very common practice when the response of a device varies over many orders of magnitude.

The transmittance of the FSS is plotted on a decibel (dB) scale as a function of frequency in Figure 10.8. The figure shows transmittance for two different cases. The first case is a TE-polarized wave at normal incidence and the second case is a TE-polarized wave incident at $\theta = 30^\circ$ and $\phi = 60^\circ$. Despite the angle of incidence,

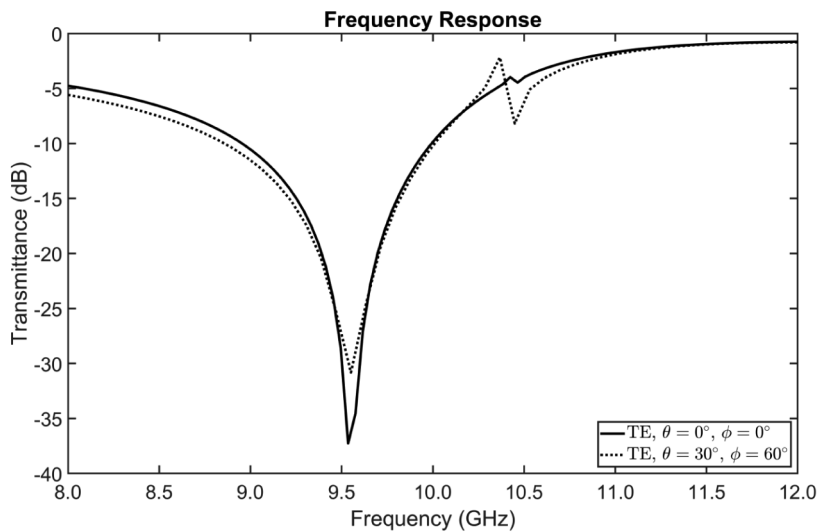


Figure 10.8 Transmittance of a TE-polarized wave from a frequency selective surface at two different angles of incidence.

the center frequency of the dip in transmittance is stable. Convergence for this device was found to start at around $NRES=80$, but the results shown in the figure were obtained at $NRES=100$.

10.5.5 Parameter Retrieval for a Left-Handed Metamaterial

Metamaterials are periodic structures that are engineered to provide effective permittivity and permeability [26–28]. Often, these are permittivity and permeability values that do not exist in ordinary materials. Metamaterials are typically composed of a periodic array of resonant metallic elements that mimic the behavior of atomic-scale resonances to produce effective material properties. All-dielectric metamaterials are also possible [29–31]. The metamaterial that will be demonstrated here is a famous left-handed metamaterial designed by Smith and co-authors [32] and is illustrated in Figure 10.9. It is composed of an FR-4 substrate with a straight wire on the back and a square split-ring resonator on the front. Dimensions and material properties are given in the figure caption. In brief, the wire on the back provides the negative permittivity while the split-ring resonator provides the negative permeability. The geometry is engineered so that these properties exist at the same frequency to make the metamaterial be left-handed and have a negative refractive index.

The results from an FDFD simulation will be used to calculate the effective refractive index n_{eff} , effective impedance η_{eff} , effective relative permittivity $\epsilon_{r,eff}$, and effective relative permeability $\mu_{r,eff}$ of the metamaterial described above. The

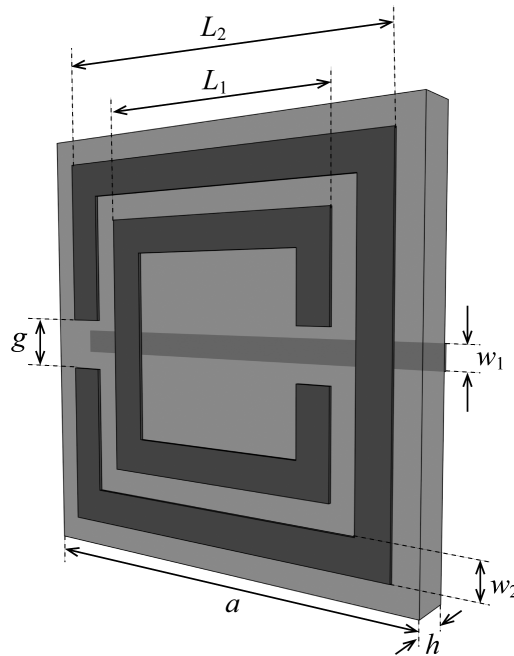


Figure 10.9 Geometry of the left-handed metamaterial in [32]. This design has $a = 2.5$ mm, $h = 0.25$ mm, $w_1 = 0.14$ mm, $w_2 = 0.2$ mm, $L_1 = 1.5$ mm, $L_2 = 2.2$ mm, and $g = 0.3$ mm. The substrate is FR-4 with a dielectric constant $\epsilon_r = 4.4$ and loss tangent $\tan\delta = 0.02$. The metal is copper with conductivity $\sigma = 5.8 \times 10^7 \Omega \cdot m$.

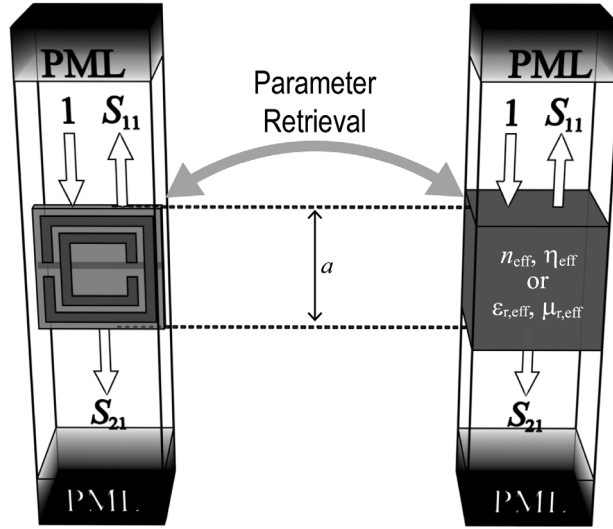


Figure 10.10 Parameter retrieval from a left-handed metamaterial. (a) FDFD simulation of scattering from a metamaterial unit cell of period a . (b) FDFD simulation of scattering from a homogeneous slab of thickness a .

concept of the retrieval process is illustrated in Figure 10.10. From the reflection and transmission simulated using FDFD, the effective properties of the metamaterial will be retrieved using the Nicolson–Ross–Weir (NRW) method [33, 34]. If correct effective properties are retrieved, then a homogeneous slab of material having these same properties and occupying the same space will produce the same reflection and transmission.

Retrieving the effective properties of the metamaterial via the NRW method begins with the analytical analysis of scattering from a homogeneous slab in air. The scattering parameters for a wave normally incident onto this slab are

$$S_{11} = \frac{(1 - t^2)r}{1 - r^2 t^2} \quad (10.71)$$

$$S_{21} = \frac{(1 - r^2)t}{1 - r^2 t^2} \quad (10.72)$$

$$r = \frac{\eta_{\text{eff}} - \eta_0}{\eta_{\text{eff}} + \eta_0} \quad (10.73)$$

$$t = \exp(jk_0 n_{\text{eff}} a) \quad (10.74)$$

In these equations, n_{eff} is the effective refractive index of the metamaterial, so it is also the refractive index of the homogeneous slab. Similarly, η_{eff} is the effective impedance of the metamaterial, so it is also the impedance of the homogeneous slab. Both of these terms are complex quantities to account for the loss. S_{11} is the complex reflection coefficient in the forward direction and S_{21} is the complex transmission

coefficient of the forward direction. The parameter r is the complex reflection coefficient of the first air-to-slab interface and is not the overall reflection coefficient from the slab. The parameter t quantifies the phase accumulated by a wave from one pass through the slab using the positive sign convention.

Calculating the effective properties of the slab starts by simulating a single unit cell to obtain values for the scattering parameters S_{11} and S_{21} . Given the scattering parameters, (10.71) to (10.74) are solved for n_{eff} and η_{eff} . First, (10.71) and (10.72) are solved for r and t to get

$$X = \frac{1 - S_{21}^2 + S_{11}^2}{2S_{11}} \quad (10.75)$$

$$r = X \pm \sqrt{X^2 - 1} \quad (10.76)$$

$$t = \frac{S_{11} + S_{21} - r}{1 - (S_{11} + S_{21})r} \quad (10.77)$$

The parameter X is just an intermediate parameter with no physical meaning. Assuming the slab is composed of a passive material that cannot exhibit gain, the sign of the square-root in (10.76) is chosen so that $|r|^2 \leq 1$. Given r and t , (10.73) and (10.74) are solved for n_{eff} and η_{eff} . The inversion of (10.74) can be difficult because it leads to an infinite number of possible answers. Taking the natural logarithm of (10.74) gives $\ln(t) = j(k_0 n a \pm 2\pi m)$ where any integer m leads to a different solution, called *branches* [35, 36]. For thin slabs, $m = 0$ can be chosen to get

$$\eta_{\text{eff}} = \eta_0 \frac{1 + r}{1 - r} \quad (10.78)$$

$$n_{\text{eff}} = \frac{\ln t}{jk_0 a} \quad (10.79)$$

Recognizing that $\eta_{\text{eff}} = \eta_0 \sqrt{\mu_{\text{r,eff}}/\epsilon_{\text{r,eff}}}$ and $n_{\text{eff}} = \eta_0 \sqrt{\mu_{\text{r,eff}}\epsilon_{\text{r,eff}}}$, the effective relative permittivity $\epsilon_{\text{r,eff}}$ and effective relative permeability $\mu_{\text{r,eff}}$ can be calculated from n_{eff} and η_{eff} according to

$$\epsilon_{\text{r,eff}} = \frac{n_{\text{eff}} \eta_0}{\eta_{\text{eff}}} \quad (10.80)$$

$$\mu_{\text{r,eff}} = \frac{n_{\text{eff}} \eta_{\text{eff}}}{\eta_0} \quad (10.81)$$

Performing parameter retrieval is an excellent way to reduce the numerical complexity of simulating devices composed of metamaterials or other subwavelength structures [29, 30, 32]. It is very difficult or impossible to calculate the effective properties of diffracting structures like photonic crystals and gratings [37]. Diffracting structures involve more physics than just an effective permittivity and permeability. FDFD simulations of devices described well by effective permittivity

and permeability can be performed more efficiently simply by assigning the effective properties to the grid. This avoids having to build complicated structures onto the grid and having to use very fine grid resolution to resolve the small features of the metamaterials themselves. Most often, simulations using just the effective properties can get away with many fewer points on the grid. Performing an FDFD simulation using just the effective properties of metamaterials will be demonstrated in Section 10.5.6 to simulate an invisibility cloak designed by TO.

The MATLAB code that performs the FDFD simulation and NRW parameter retrieval can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter10_metamaterial.m`. The header extends from lines 1 to 25 and nothing is different from the other codes. The dashboard extends from lines 27 to 63. The source parameters are defined in lines 31 to 39. `f1` and `f2` are the starting and ending frequency for the frequency sweep, `NFREQ` is the number of frequency points to use, and `FREQ` is the array containing all of the frequencies to simulate in the parameter sweep. It is good practice to resolve the response with enough frequency points so that the phase can be unwrapped accurately. The simulation is performed at normal incidence so both `SRC.theta` and `SRC.phi` are set to zero. For this simulation, the TM polarization is selected to place the magnetic field parallel to the axis of the loops. Lines 41 to 48 define the dimensions of the metamaterial as described in Figure 10.9. The material properties are specified in terms of the electrical conductivity `sigma` for the metal and the dielectric constant `er` and loss tangent `tand` for the dielectric. From these, the complex relative permittivity for both materials is calculated on lines 50 to 56. Last, the grid parameters are defined from lines 58 to 63. Convergence was found to begin at around `NRES=100`. The results that will be discussed below were obtained at `NRES=120`. That is, 120 grid cells per wavelength.

Lines 65 to 110 calculate a grid optimized for simulating this metamaterial. The grid resolution is snapped to match the period of the metamaterial. Lines 112 to 170 build the unit cell of the metamaterial onto the $2\times$ grid. First, lines 116 to 118 initialize both the relative permittivity array `DEV.ER2` and the relative permeability array `DEV.UR2` to all ones to represent air. The FR-4 substrate is added to the grid on lines 120 to 127 using the centering algorithm covered in Chapter 1. Observe in these calculations that the centering algorithm has been modified so that the arguments inside of the rounding operations are divided by two and then multiplied by two on the outside. This ensures an even number of cells is calculated. When one is added to an even number an odd number is always obtained. This is being done so that the metals are added to cells on the grid at odd array indexes because these correspond to the field components parallel to the metals. This trick is used throughout this section of code. The wire is added at the back edge of the substrate on lines 129 to 132 using the same modified centering algorithm. Lines 134 to 151 add the outer ring in three steps, also making use of the modified centering algorithm. Lines 135 to 142 add a large square of metal covering the entire area of the outer ring. Lines 143 to 147 subtract a smaller square to form the ring. Lines 147 to 151 remove more metal to add the gap to form the split ring. Lines 153 to 170 add the inner ring following the same procedure used to build the outer ring.

The frequency sweep is performed from lines 172 to 234. Line 177 defines the `DEV.RES` data structure that stores the grid resolution parameters `dx`, `dy`, and `dz`. Lines 179 to 184 initialize the arrays where the frequency response will be stored. `S11` is

the complex reflection coefficient, S_{21} is the complex transmission coefficient, REF is reflectance, TRN is transmittance, and CON is the power conservation. The frequency sweep itself is performed from lines 186 to 234. It is a typical parameter sweep, except around lines 200 to 209 where the S_{11} and S_{21} parameters are recorded. It is critical to calculate phase as if the phase of the source is zero immediately on the entrance face of the unit cell and to calculate reflection and transmission with the phase that is immediately on the exit face of the unit cell. The function `fd3d()`, however, launches the wave from some distance away and also calculates reflection and transmission at some distance away. Lines 205 and 206 of the code remove this extra phase by dividing the exponential terms. It is also possible to modify the `fd3d()` function to place the TF/SF interface and reflection and transmission record planes at the edge of the device, but it was desired to use the generic `fd3d()` function as it is without any modifications.

At this point, the FDFD simulation is finished and parameter retrieval can begin. This is considered postprocessing. While not shown in the code, it is a good practice to save the simulation results at this point. This allows the results of the long simulation to be quickly loaded from memory to work on coding the parameter retrieval section. The parameter retrieval extends from lines 236 to 282 and all of the data is plotted from lines 284 to 346. The NRW method in the literature was derived using the positive sign convention for waves, but the FDFD method in this book is based on the negative sign convention. Lines 240 to 242 calculate the complex conjugate of the S_{11} and S_{21} scattering parameters to reverse the phase and be compatible with positive sign convention equations. In MATLAB and many other programming languages, the phase portion ϕ of a complex number is kept in the range $-\pi \leq \phi \leq \pi$. If the phase exceeds these limits, an integer multiple of 2π is added or subtracted so that the phase falls within this range. This is called *wrapping* and leads to nonphysical discontinuities in the phase response that causes problems with the parameter retrieval algorithm. To fix this, lines 244 to 263 unwrap the phase so that it does not exhibit discontinuities caused by wrapping. Line 266 calculates the intermediate variable x using (10.75). Lines 268 to 272 calculate the reflection coefficient r using (10.76) while also resolving the sign so that $|r|^2 \leq 1$. The transmission coefficient t is calculated on line 274 using (10.77). From here, lines 276 to 278 calculate the effective impedance η_{eff} and effective refractive index n_{eff} using (10.78) and (10.79), respectively. The last step in the parameter retrieval happens on lines 280 to 282 where the effective relative permittivity ϵ_r and effective permeability μ_r are calculated using (10.80) and (10.81), respectively. Lines 284 to 346 plot all of the data calculated from the simulation and parameter retrieval.

The results of the simulation and the parameter retrieval are shown in Figure 10.11 for $N_{RES}=120$ and match well with what was obtained in [32]. Fundamentally, FDFD calculates reflection and transmission from the metamaterial unit cell as shown in Figure 10.10(a). Everything else is calculated from these results. The reflectance and transmittance as a function of frequency are plotted in Figure 10.11(a). Observe that conservation is not obeyed due to the loss incorporated into the simulation. Even when the loss is to be incorporated, it is good practice to run a preliminary simulation with loss set to zero to ensure conservation is obeyed. Violation of conservation is a strong indication of a problem with the code, such as the spacer regions being too small or the PML not working correctly. The phase component of the reflection

and transmission coefficients was unwrapped to form continuous lines and plotted in Figure 10.11(b). It is from the S_{11} and S_{21} terms that the effective refractive index n_{eff} and effective impedance η_{eff} of the metamaterial were calculated using (10.75) to (10.79). These parameters as a function of frequency are plotted in Figure 10.11(c, d). Observe that the effective refractive index is negative in the vicinity of 10.6 GHz. A medium with a negative refractive index is said to be left-handed because the electric and magnetic fields obey a left-hand rule instead of a right-hand rule like in ordinary media. Note the imaginary part of the effective refractive index in Figure 10.11(c) is largest in the vicinity of the negative refractive index. This is a critical problem with metamaterials. They often exhibit very high losses at the same frequencies they would be most useful. Figure 10.11(e, f) shows the effective relative permittivity $\epsilon_{r,\text{eff}}$ and effective relative permeability $\mu_{r,\text{eff}}$ respectively. Observe that both of these parameters are simultaneously negative where there is a negative refractive index.

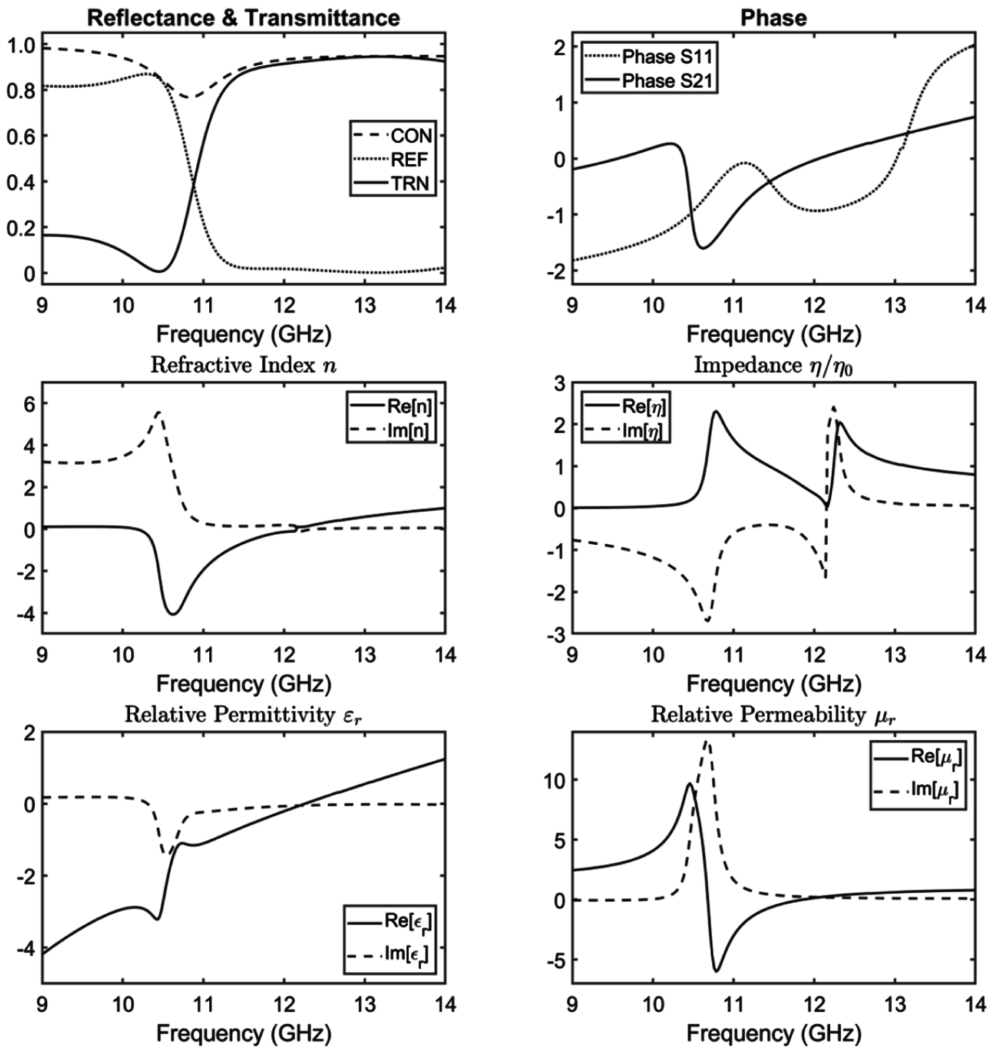


Figure 10.11 Results of parameter retrieval. Metamaterial has a negative index of refraction around 10.6 GHz.

10.5.6 Simulation of an Invisibility Cloak

TO is a technique to design the permittivity and permeability functions so that waves can be made to flow along any desired path [38–41]. A coordinate transform defines the paths the waves should be made to follow. The coordinate transform is applied to Maxwell's equations. The math associated with the transform is moved out of the coordinates and incorporated into the permittivity and permeability functions. The output of TO is a map of permittivity and permeability as a function of position. The permittivity and permeability calculated from TO are typically anisotropic and entail extreme values that are only realizable using exotic materials or metamaterials. The most famous application of TO is invisibility cloaks [42–44], but many other more practical applications exist [38]. Simulating TO devices can be difficult, but the ability to do so will prove to be a fun and powerful tool in your simulation toolbox.

For this example, the classic cylindrical invisibility cloak [44] will be simulated using FDFD. The grid strategy for this simulation is illustrated in Figure 10.12. The cloak has an inner radius of R_1 and an outer radius of R_2 . This is not a device that produces significant evanescent fields so no spacer regions are required. For this simulation, space around the cloak was added simply to visualize the wave. A PML is incorporated around all four boundaries. The TF/SF interface encircles the entire grid just inside of the PML. The PMLs are placed in the SF region while the cloak is placed in the TF region.

The MATLAB code to generate and simulate a cylindrical invisibility cloak can be downloaded at <https://empossible.net/fdfdbook/>. The file is called `Chapter10_cloak.m`. The code uses many of the tools and concepts covered in this chapter. The header extends from lines 1 to 12 and contains all of the same items as previous programs. The dashboard extends from lines 14 to 31 and is where all of the variables

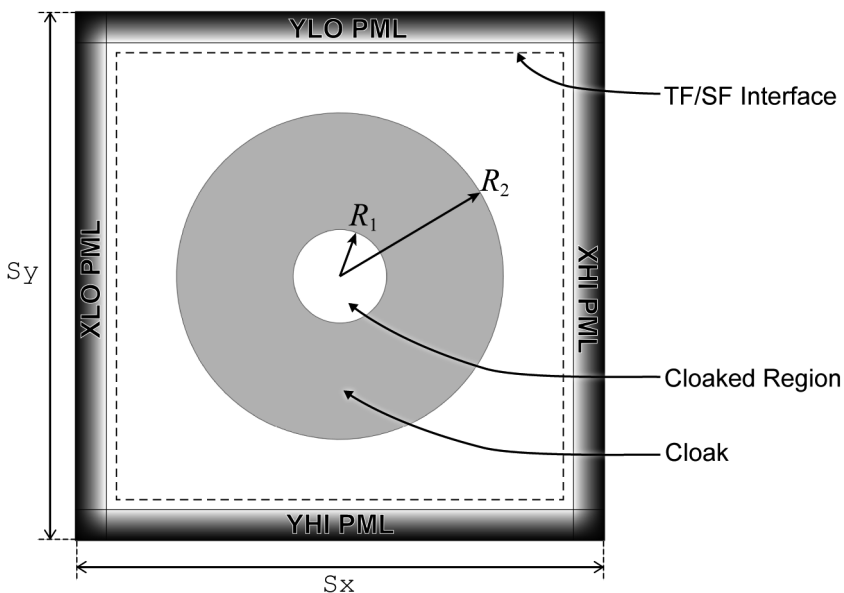


Figure 10.12 Grid strategy to simulate an invisibility cloak designed using TO.

are defined that control the simulation. The source is defined on lines 18 to 21. λ_{am0} is the free space wavelength, θ is the angle of the source wave, and \mathbf{P} is the polarization vector. The polarization is set to the z -direction corresponding to a vertically polarized source.

Lines 33 to 58 calculate the grid. The primary difference in this section compared to previous chapters is that n_{max} is not used to calculate the grid resolution. In general, TO devices can have extreme refractive indices and it is possible it could approach infinity or zero for some devices. For this reason, n_{max} is not used because it could lead to an unfeasible grid.

The invisibility cloak is constructed from lines 60 to 139. Lines 64 to 73 initialize all nine tensor elements for permittivity to free space. There is no need to initialize the permeability because the permeability will just be set equal to the permittivity. Lines 75 to 104 are where the permittivity values for the invisibility cloak are calculated. A double loop is used to iterate through every point on the $2 \times$ grid. For each point, lines 79 and 80 calculate the cylindrical coordinate parameters ρ and ϕ . The condition $R_1 < \rho < R_2$ checks if the current point in the iteration resides inside of the cloak. If so, the permittivity tensor for the cloak is calculated at that point. Otherwise, it moves on to the next iteration of the double loop. Inside of the cloak, the three diagonal elements of the permittivity tensor are calculated in cylindrical coordinates according to [44]

$$[\epsilon_r(\rho, \phi, z)] = \begin{bmatrix} \frac{\rho - R_1}{\rho} & 0 & 0 \\ 0 & \frac{\rho}{\rho - R_1} & 0 \\ 0 & 0 & \left(\frac{R_2}{R_2 - R_1}\right)^2 \frac{\rho - R_1}{\rho} \end{bmatrix} \quad (10.82)$$

The permittivity tensor in Cartesian coordinates $[\epsilon_r(x, y)]$ is calculated by rotating the tensor in (10.82) about the z -axis by the cylindrical coordinate angle ϕ . The rotation is performed on lines 90 to 93. After this, lines 95 to 103 populate the tensor arrays with the elements of $[\epsilon_r(x, y)]$.

Recognizing the anisotropic permeability will lead to a very slow calculation of the wave matrix, the impermeability tensor $[\psi_r(x, y)]$ is calculated directly from the permittivity tensor $[\epsilon_r(x, y)]$ on lines 108 to 120. This section of code uses (10.45) to invert the permittivity tensor at each point on the grid to arrive at the impermeability tensor elements. Given the impermeability, the wave matrix is calculated much more efficiently using (10.46). Lines 122 to 141 form diagonal matrices from the permittivity and impermeability tensor arrays. These are used to calculate the wave matrix in a later part of the code.

The FDFD simulation is performed from lines 143 to 223. The source field is calculated on lines 147 to 157. The incident wave vector \mathbf{k}_{inc} is calculated on line 151. The source field is calculated on line 157 by assembling the source field components E_x , E_y , and E_z into a single column vector. The derivative matrices are constructed on line 163 by calling the `yeeder3d()` function described previously. Since

Dirichlet boundary conditions are used, it is possible to calculate the interpolation matrices directly from the derivative matrices using (10.30). This happens on lines 165 to 168. Lines 170 to 185 calculate the SCPML parameters. Line 171 calls the `calcpml3d()` function that calculates the SCPML terms on the $2\times$ grid. Lines 173 to 185 extract the SCPML terms for the Yee grid and forms the diagonal matrices that are used to calculate the curl matrices. It is actually the inverse of the SCPML terms that are used to build the curl matrices. Rather than use matrix division to do this, the PML terms are inverted in the diagonalization step. Lines 187 to 193 form the permittivity and impermeability tensors. Lines 195 to 203 calculate the curl matrices for both the electric and magnetic fields. This is where the SCPML is incorporated into the simulation. Lines 205 to 213 build the TF/SF masking matrix Q . For this simulation, the TF/SF interface forms a rectangle around the entire grid, just outside of the SCPML regions. The wave matrix A is calculated on line 216 using (10.46). The source vector b is calculated on line 219 using the QAAQ equation and then the field f is solved using backward division on line 223. The backward division was chosen here over iteration because the size of the simulation is generally small enough that backward division is the faster solution method. A message is displayed to the command window on line 222 to convey that the field is being solved because this step can take some time to process.

At this point, the FDFD simulation is finished. Lines 225 to 228 extract the vector components f_x , f_y , and f_z from the field solution f . Lines 230 to 233 reshape the field components back to the Yee grid. Lines 235 to 250 visualize the results.

While the program may run and produce a result without error, a convergence study must be performed before any conclusions can be made about the results. For this simulation, convergence was observed at around $NRES = 60$. The results of this simulation are shown in Figure 10.13. It should be mentioned that this would be a great time to produce a nice animation of the field with a GIF as described in Chapter 1!

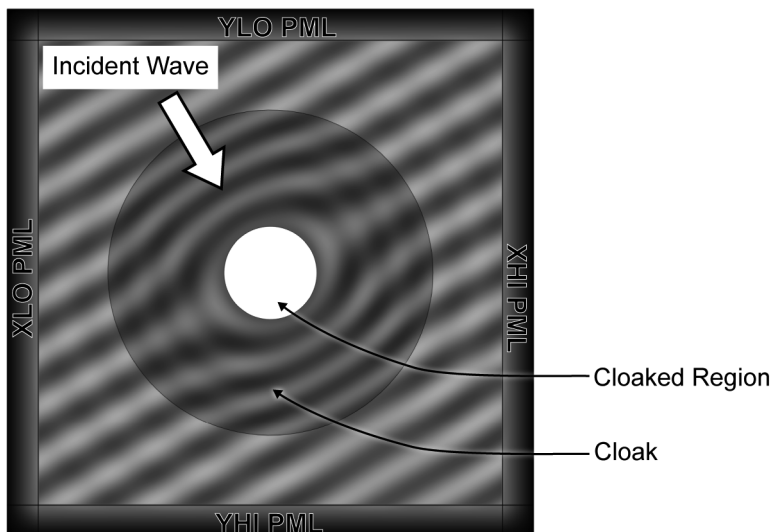


Figure 10.13 Simulation results for a cylindrical invisibility cloak.

References

- [1] Rumpf, R. C., *et al.*, “Finite-Difference Frequency-Domain Algorithm for Modeling Electromagnetic Scattering from General Anisotropic Objects,” *Progress in Electromagnetics Research*, Vol. 61, 2014, pp. 55–67.
- [2] Shin, W., *3D Finite-Difference Frequency-Domain Method for Plasmonics and Nanophotonics*: Stanford University, 2013.
- [3] Yee, K., “Numerical Solution of Initial Boundary Value Problems Involving Maxwell’s Equations in Isotropic Media,” *IEEE Trans. on Antennas and Propagation*, Vol. 14, No. 3, 1966, pp. 302–307.
- [4] Chapra, S. C., and R. P. Canale, *Numerical Methods for Engineers*, Seventh Edition, New York: McGraw-Hill Education, 2015.
- [5] Greenbaum, A., *Iterative Methods for Solving Linear Systems*: SIAM, 1997.
- [6] Saad, Y., *Iterative Methods for Sparse Linear Systems*: SIAM, 2003.
- [7] Strang, G., *et al.*, *Introduction to Linear Algebra*, Wellesley, MA: Wellesley-Cambridge Press, 1993.
- [8] Bertaccini, D., and F. Durastante, *Iterative Methods and Preconditioning for Large and Sparse Linear Systems with Applications*: CRC Press, 2018.
- [9] Anwar, R. S., L. Mao, and H. Ning, “Frequency Selective Surfaces: A Review,” *Applied Sciences*, Vol. 8, No. 9, 2018, p. 1689.
- [10] Farahat, N., and R. Mittra, “Analysis of Frequency Selective Surfaces Using the Finite Difference Time Domain (FDTD) Method,” *IEEE Antennas and Propagation Society International Symposium*, San Antonio, TX, June 16–21, 2002, pp. 568–571.
- [11] Mackay, A., B. Sanz-Izquierdo, and E. A. Parker, “Evolution of Frequency Selective Surfaces,” *FERMAT*, Vol. 2, No. 008, 2014, pp. 1–7.
- [12] Chen, H.-T., A. J. Taylor, and N. Yu, “A Review of Metasurfaces: Physics and Applications,” *Reports on Progress in Physics*, Vol. 79, No. 7, 2016, p. 076401.
- [13] Ding, F., A. Pors, and S. I. Bozhevolnyi, “Gradient Metasurfaces: A Review of Fundamentals and Applications,” *Reports on Progress in Physics*, Vol. 81, No. 2, 2017, p. 026401.
- [14] Glybovski, S. B., *et al.*, “Metasurfaces: From Microwaves to Visible,” *Physics Reports*, Vol. 634, 2016, pp. 1–72.
- [15] Hsiao, H. H., C. H. Chu, and D. P. Tsai, “Fundamentals and Applications of Metasurfaces,” *Small Methods*, Vol. 1, No. 4, 2017, p. 1600064.
- [16] Kildishev, A. V., A. Boltasseva, and V. M. Shalaev, “Planar Photonics with Metasurfaces,” *Science*, Vol. 339, No. 6125, 2013.
- [17] Luo, X., “Principles of Electromagnetic Waves in Metasurfaces,” *Science China Physics, Mechanics & Astronomy*, Vol. 58, No. 9, 2015, pp. 1–18.
- [18] Meinzer, N., W. L. Barnes, and I. R. Hooper, “Plasmonic Meta-Atoms and Metasurfaces,” *Nature Photonics*, Vol. 8, No. 12, 2014, p. 889.
- [19] Vahabzadeh, Y., K. Achouri, and C. Caloz, “Simulation of Metasurfaces in Finite Difference Techniques,” *IEEE Trans. on Antennas and Propagation*, Vol. 64, No. 11, 2016, pp. 4753–4759.
- [20] Hansen, R. C., *Phased Array Antennas*, Hoboken, NJ: John Wiley & Sons, 2009.
- [21] Jensen, M. A., and J. W. Wallace, “A Review of Antennas and Propagation for MIMO Wireless Communications,” *IEEE Trans. on Antennas and Propagation*, Vol. 52, No. 11, 2004, pp. 2810–2824.
- [22] Sievenpiper, D., “Review of Theory, Fabrication, and Applications of High-Impedance Ground Planes,” *Metamaterials: Physics and Engineering Explorations*, 2006, pp. 287–311: Wiley Online Library.
- [23] Melais, S. E., and T. M. Weller, “A Multilayer Jerusalem Cross Frequency Selective Surface,” *2009 IEEE 10th Annual Wireless and Microwave Technology Conference*, Clearwater, FL, April 20–21, 2009, pp. 1–5.

- [24] Sohail, I., *et al.*, “A Linear to Circular Polarization Converter Based on Jerusalem-Cross Frequency Selective Surface,” *2013 7th European Conference on Antennas and Propagation*, Gothenburg, Sweden, April 8–12, 2013, pp. 2141–2143.
- [25] Tsao, C.-H., and R. Mittra, “Spectral-Domain Analysis of Frequency Selective Surfaces Comprised of Periodic Arrays of Cross Dipoles and Jerusalem Crosses,” *IEEE Trans. on Antennas and Propagation*, Vol. 32, No. 5, 1984, pp. 478–486.
- [26] Cai, W., and V. M. Shalaev, *Optical Metamaterials*, New York: Springer, 2010.
- [27] Cui, T. J., D. R. Smith, and R. Liu, *Metamaterials*, New York: Springer, 2010.
- [28] Simovski, C., “Material Parameters of Metamaterials (A Review),” *Optics and Spectroscopy*, Vol. 107, No. 5, 2009, pp. 726–753.
- [29] Avila, J., *et al.*, “Optimization and Characterization of Negative Uniaxial Metamaterials,” *Progress in Electromagnetics Research C*, Vol. 74, 2017, pp. 111–121.
- [30] Garcia, C. R., *et al.*, “3D Printing of Anisotropic Metamaterials,” *Progress in Electromagnetics Research Letters*, Vol. 34, 2012, pp. 75–82.
- [31] Jahani, S., and Z. Jacob, “All-Dielectric Metamaterials,” *Nature Nanotechnology*, Vol. 11, No. 1, 2016, pp. 23–36.
- [32] Smith, D., *et al.*, “Electromagnetic Parameter Retrieval from Inhomogeneous Metamaterials,” *Physical Review E*, Vol. 71, No. 3, 2005, p. 036617.
- [33] Nicolson, A., and G. Ross, “Measurement of the Intrinsic Properties of Materials by Time-Domain Techniques,” *IEEE Trans. on Instrumentation and Measurement*, Vol. 19, No. 4, 1970, pp. 377–382.
- [34] Vicente, A. N., G. M. Dip, and C. Junqueira, “The Step by Step Development of NRW Method,” *2011 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference*, Natal, Brazil, October 29–1 November 2011, pp. 738–742.
- [35] Alu, A., “First-Principles Homogenization Theory for Periodic Metamaterials,” *Physical Review B*, Vol. 84, No. 7, 2011, p. 075153.
- [36] Perrins, W., and R. McPhedran, “Metamaterials and the Homogenization of Composite Materials,” *Metamaterials*, Vol. 4, No. 1, 2010, pp. 24–31.
- [37] Schwartz, B. T., and R. Piestun, “Dynamic Properties of Photonic Crystals and Their Effective Refractive Index,” *Journal of the Optical Society of America B*, Vol. 22, No. 9, 2005, pp. 2018–2026.
- [38] Kundtz, N. B., D. R. Smith, and J. B. Pendry, “Electromagnetic Design with Transformation Pptics,” *Proceedings of the IEEE*, Vol. 99, No. 10, 2010, pp. 1622–1633.
- [39] Kwon, D.-H., and D. H. Werner, “Transformation Electromagnetics: An Overview of the Theory and Applications,” *IEEE Antennas and Propagation Magazine*, Vol. 52, No. 1, 2010, pp. 24–46.
- [40] Ward, A., and J. B. Pendry, “Refraction and Geometry in Maxwell’s Equations,” *Journal of Modern Optics*, Vol. 43, No. 4, 1996, pp. 773–793.
- [41] Eric A. Berry, R. C. R., “Generating Spatially-Variant Metamaterial Lattices Designed from Spatial Transforms,” *Progress in Electromagnetics Research*, Vol. 92, 2020, pp. 103–113.
- [42] Choudhury, B., and R. Jha, “A Review of Metamaterial Invisibility Cloaks,” *Computers, Materials & Continua*, Vol. 33, No. 3, 2013, pp. 275–303.
- [43] Landy, N., and D. R. Smith, “A Full-Parameter Unidirectional Metamaterial Cloak for Microwaves,” *Nature Materials*, Vol. 12, No. 1, 2013, pp. 25–28.
- [44] Schurig, D., *et al.*, “Metamaterial Electromagnetic Cloak at Microwave Frequencies,” *Science*, Vol. 314, No. 5801, 2006, pp. 977–980.

A.1 Best Practices for Building Devices onto Yee Grids

This section will describe some basic best practices for assigning material values to the Yee grid. When metal structures, or structures with high permittivity contrast, are built onto the grid, it is best to have the first electric field component immediately inside of the metal structure be tangential to the edge of the structure. For E mode simulations, all electric fields will always be tangential to the edges because the electric field is always perpendicular to the plane of the structure. More care must be taken when performing H mode simulations with metals. This practice makes it easier for the Yee grid to satisfy the boundary conditions at the interface of the metal. Doing this will improve the rate of convergence of a simulation containing metals and high-permittivity structures. Figure A.1 demonstrates two ways to place a square metal element onto the Yee grid. The ideal placement is shown in Figure A.1(a) where the electric field components immediately inside of the square are tangential to the edge of the square. Figure A.1(b) shows the same metal element placed poorly onto the grid. It has electric field components immediately inside of the square that are normal to the edge of the square. The second case will converge more slowly than the first case. The practice described here is easy for rectangular shapes, but is more difficult or impossible when curved shapes are built onto the grid. When curved shapes are needed, you may be forced to accept slower convergence.

From the above discussion, placing metals on grids might be more intuitively done by directly populating the Yee grid arrays ER_{xx} , ER_{yy} , and ER_{zz} . If the $2\times$ grid technique is to be used, the array indices where structures are placed onto the

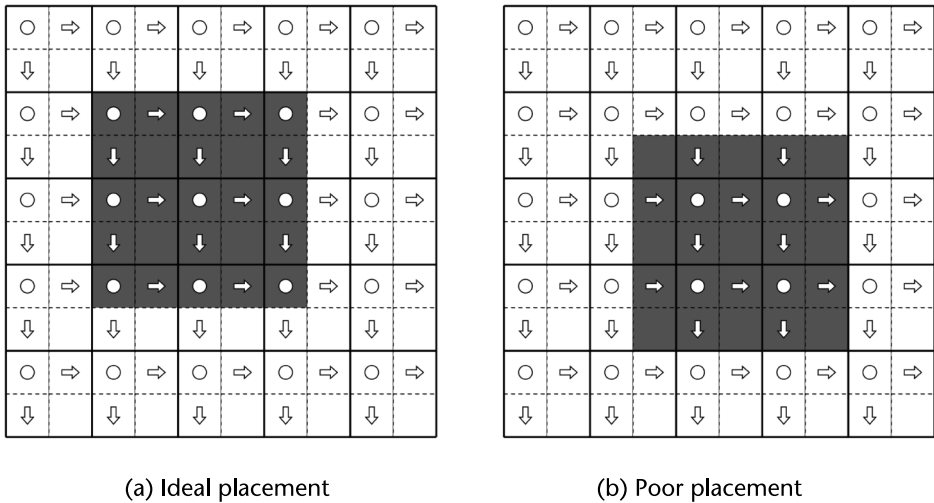


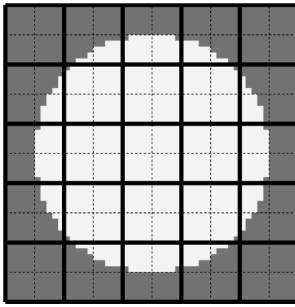
Figure A.1 (a) Ideal placement of a metal square onto a Yee grid where tangential electric field components are immediately inside the metal. (b) Poor placement of a metal square onto a Yee grid where electric field immediately inside metal is not tangential.

grid must be more carefully considered. Observe from Figure A.1(a) that the square patch of metal starts and ends on odd array indices on the $2\times$ grid. The material outside of the square patch starts and stops on even array indices. The array index n_x can be forced to be even or odd using (A.1) and (A.2). The same can be done for any other array index such as n_y or n_z . Equation (A.1) divides n_x by two and then multiplies by two to get essentially the original value of n_x . However, the quantity is rounded after dividing by two to get the nearest integer. Multiplying any integer by two will always give an even number. Equation (A.1) will give the closest even number to the original value of n_x . Equation (A.2) essentially uses the same equation but adds one to always give an odd number. The `floor()` command is used instead of `round()` in order to get the odd number closest to the original value of n_x after one is added. It is common to incorporate these equations into the various techniques described in Chapter 1, such as the centering algorithm.

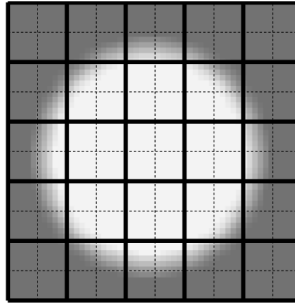
$$n_x = 2 * \text{round}(n_x/2) \quad \text{makes } n_x \text{ even} \quad (\text{A.1})$$

$$n_x = 2 * \text{floor}(n_x/2) + 1 \quad \text{makes } n_x \text{ odd} \quad (\text{A.2})$$

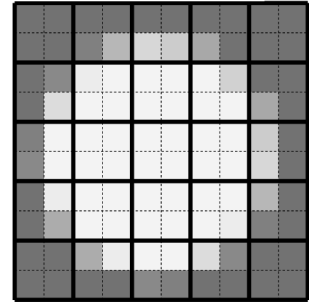
When curved dielectric structures are built onto the grid, the edge of the structure will cut through some of the grid cells. A simple technique to improve convergence rate is to assign the average permittivity (or permeability) to these cells that are partly filled. For example, if 70% of a cell is occupied by relative permittivity 2.5 and 30% is occupied by relative permittivity 6.0, the value that should be assigned to the cell is $(0.70)(2.5) + (0.30)(6.0) = 3.55$. A simple way to accomplish this is illustrated in Figure A.2. The technique involves three separate grids that all represent the same physical space. These are the Yee grid, the $2\times$ grid containing twice as many cells as the Yee grid, and a high-resolution grid containing even more cells than the $2\times$ grid. In this example, the high-resolution grid has five times the number of cells than does the $2\times$ grid. Figure A.2(a) shows the Yee grid, the $2\times$ grid, and a cylinder constructed onto the high-resolution grid. The next step is to blur the high-resolution grid using a rectangular blur function that is 5×5 cells wide. Figure A.2(d) shows the blur function centered on the high-resolution grid. The blurring operation is performed as a convolution using two-dimensional fast Fourier transforms (FFTs). The FFT of the high-resolution grid is multiplied by the FFT of the blurring function. The inverse FFT is calculated from the product to get the blurred function on the high-resolution grid as shown in Figure A.2(b). After the blurring operation, the points on the high-resolution grid that fall at the center of the $2\times$ grid cells are the correct weighted averages. The cells on the $2\times$ grid are assigned these weighted averages.



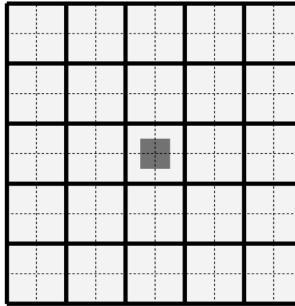
(a) Grids with high-resolution cylinder



(b) Grids after blur operation



(c) Grids after extracting averages



(d) Blur function on high-resolution grid

Figure A.2 Simple method to average the permittivity (or permeability) at the edges of curved boundaries. Thick lines show the Yee grid cells and the dashed lines show the $2\times$ grid cells. (a) Cylinder constructed into a high-resolution grid overlaid onto Yee grid and $2\times$ grid. (b) High-resolution cylinder after blurring operation. (c) Materials assigned to $2\times$ grid after being extracted from the high-resolution grid. (d) The blur function on the high-resolution grid.

A.2 Method Summaries

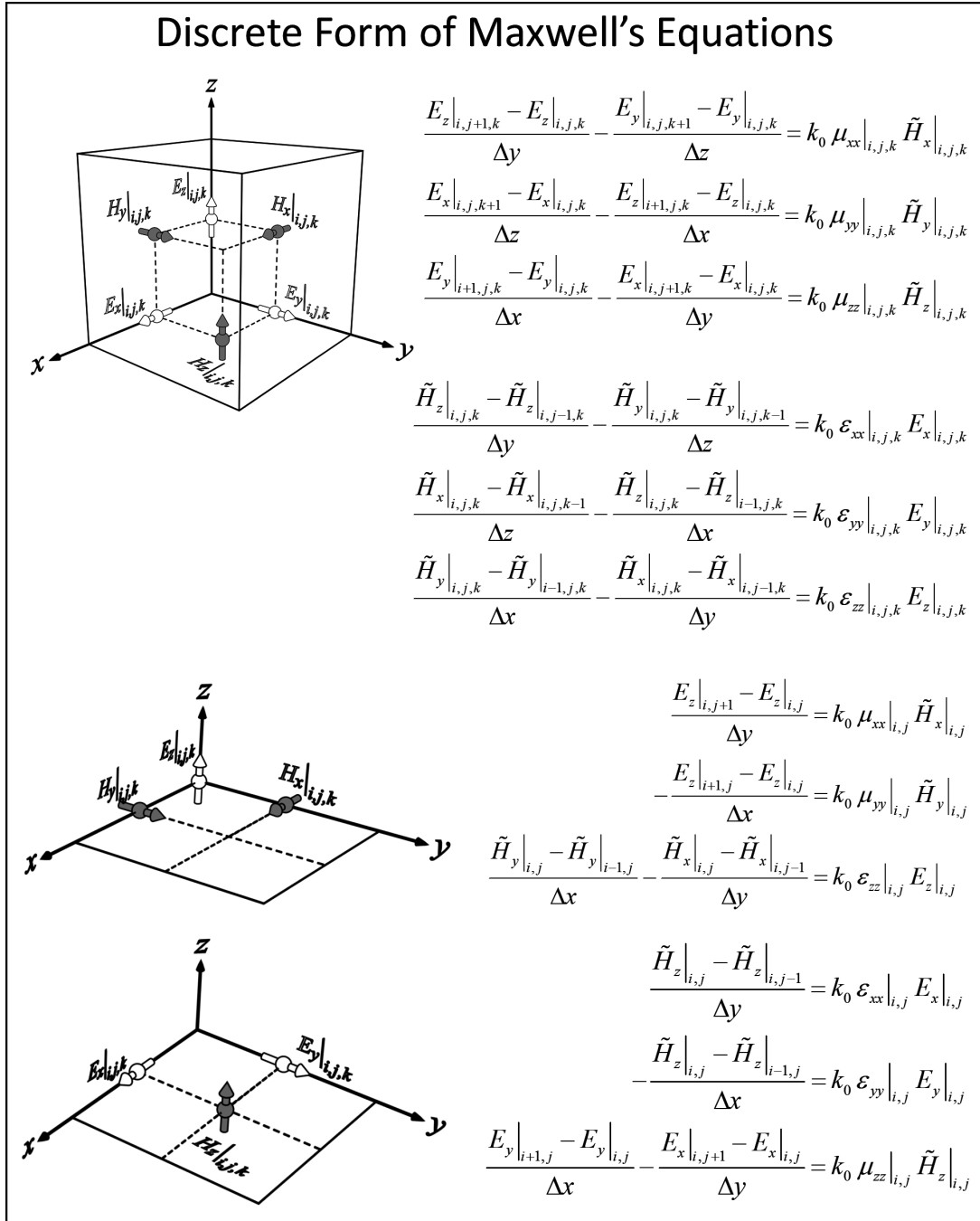


Figure A.3 Summary of the Yee grids and discrete form of Maxwell's equations with doubly diagonally anisotropic media.

Uniaxial Perfectly Matched Layer

$$[S] = \begin{bmatrix} s_x^{-1} & 0 & 0 \\ 0 & s_x & 0 \\ 0 & 0 & s_x \end{bmatrix} \begin{bmatrix} s_y & 0 & 0 \\ 0 & s_y^{-1} & 0 \\ 0 & 0 & s_y \end{bmatrix} \begin{bmatrix} s_z & 0 & 0 \\ 0 & s_z & 0 \\ 0 & 0 & s_z^{-1} \end{bmatrix} = \begin{bmatrix} s_y s_z / s_x & 0 & 0 \\ 0 & s_x s_z / s_y & 0 \\ 0 & 0 & s_x s_y / s_z \end{bmatrix}$$

$$\nabla' \times \vec{E} = k_0 [\mu_r'] \vec{H}$$

$$\nabla' \times \vec{H} = k_0 [\varepsilon_r'] \vec{E}$$

$$[\mu_r'] = [S][\mu_r] = \begin{bmatrix} s_x^{-1} s_y s_z \mu_{xx} & 0 & 0 \\ 0 & s_x s_y^{-1} s_z \mu_{yy} & 0 \\ 0 & 0 & s_x s_y s_z^{-1} \mu_{zz} \end{bmatrix}$$

$$[\varepsilon_r'] = [S][\varepsilon_r] = \begin{bmatrix} s_x^{-1} s_y s_z \varepsilon_{xx} & 0 & 0 \\ 0 & s_x s_y^{-1} s_z \varepsilon_{yy} & 0 \\ 0 & 0 & s_x s_y s_z^{-1} \varepsilon_{zz} \end{bmatrix}$$

Stretched-Coordinate Perfectly Matched Layer

$$\begin{bmatrix} 0 & -\frac{1}{s_z} \frac{\partial}{\partial z} & \frac{1}{s_y} \frac{\partial}{\partial y} \\ \frac{1}{s_z} \frac{\partial}{\partial z} & 0 & -\frac{1}{s_x} \frac{\partial}{\partial x} \\ -\frac{1}{s_y} \frac{\partial}{\partial y} & \frac{1}{s_x} \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix} = -j\omega \begin{bmatrix} \mu_{xx} & \mu_{xy} & \mu_{xz} \\ \mu_{yx} & \mu_{yy} & \mu_{yz} \\ \mu_{zx} & \mu_{zy} & \mu_{zz} \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix}$$

$$\begin{bmatrix} 0 & -\frac{1}{s_z} \frac{\partial}{\partial z} & \frac{1}{s_y} \frac{\partial}{\partial y} \\ \frac{1}{s_z} \frac{\partial}{\partial z} & 0 & -\frac{1}{s_x} \frac{\partial}{\partial x} \\ -\frac{1}{s_y} \frac{\partial}{\partial y} & \frac{1}{s_x} \frac{\partial}{\partial x} & 0 \end{bmatrix} \begin{bmatrix} H_x \\ H_y \\ H_z \end{bmatrix} = -j\omega \begin{bmatrix} \varepsilon_{xx} & \varepsilon_{xy} & \varepsilon_{xz} \\ \varepsilon_{yx} & \varepsilon_{yy} & \varepsilon_{yz} \\ \varepsilon_{zx} & \varepsilon_{zy} & \varepsilon_{zz} \end{bmatrix} \begin{bmatrix} E_x \\ E_y \\ E_z \end{bmatrix}$$

Calculating the PML Parameters

$$s_x(x) = a_x(x) [1 - j60\sigma_x(x)] \quad \sigma_x(x) = \sigma_{\max} \sin^2\left(\frac{\pi x}{2L_x}\right) \quad a_x(x) = 1 + (a_{\max} - 1) \left(\frac{x}{L_x}\right)^p$$

$$s_y(y) = a_y(y) [1 - j60\sigma_y(y)] \quad \sigma_y(y) = \sigma_{\max} \sin^2\left(\frac{\pi y}{2L_y}\right) \quad a_y(y) = 1 + (a_{\max} - 1) \left(\frac{y}{L_y}\right)^p$$

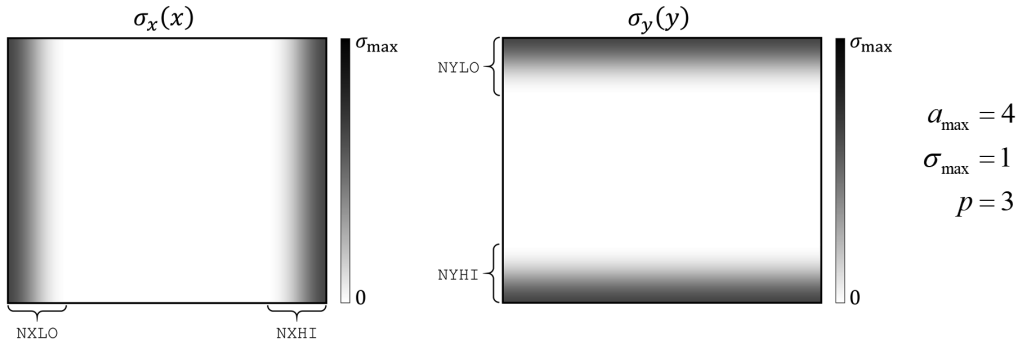
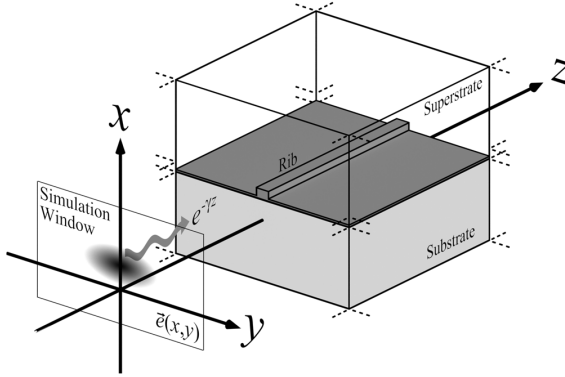


Figure A.4 Summary of both the uniaxial and the stretched-coordinate perfectly matched layer absorbing boundaries.

Hybrid Mode Calculation in Channel Waveguides



$$\Omega^2 \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix} = \tilde{\gamma}^2 \begin{bmatrix} \mathbf{e}_x \\ \mathbf{e}_y \end{bmatrix}$$

$$\Omega^2 = \mathbf{P}\mathbf{Q}$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h & -(\mathbf{D}_{x'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h + \boldsymbol{\mu}_{yy}) \\ \mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{y'}^h + \boldsymbol{\mu}_{xx} & -\mathbf{D}_{y'}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{x'}^h \end{bmatrix}$$

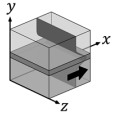
$$\mathbf{Q} = \begin{bmatrix} \mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e & -(\mathbf{D}_{x'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e + \boldsymbol{\epsilon}_{yy}) \\ \mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{y'}^e + \boldsymbol{\epsilon}_{xx} & -\mathbf{D}_{y'}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{x'}^e \end{bmatrix}$$

Slab Waveguide Analysis

Coordinates

E Mode

H Mode



$$-(\mathbf{D}_{yy}^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{yy}^e + \boldsymbol{\epsilon}_{zz}) \mathbf{e}_z = \tilde{\gamma}^2 \boldsymbol{\mu}_{yy}^{-1} \mathbf{e}_z$$

$$\tilde{\mathbf{h}}_x = \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{xz}^e \mathbf{e}_z$$

$$\tilde{\mathbf{h}}_y = \tilde{\gamma} \boldsymbol{\mu}_{yy}^{-1} \mathbf{e}_z$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{zz}) \tilde{\mathbf{h}}_z = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{zz}^{-1} \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_x = \boldsymbol{\epsilon}_{xx}^{-1} \mathbf{D}_{xz}^e \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_y = \tilde{\gamma} \boldsymbol{\epsilon}_{yy}^{-1} \tilde{\mathbf{h}}_z$$



$$-(\mathbf{D}_{zz}^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{zz}^e + \boldsymbol{\epsilon}_{yy}) \mathbf{e}_y = \tilde{\gamma}^2 \boldsymbol{\mu}_{zz}^{-1} \mathbf{e}_y$$

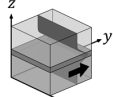
$$\tilde{\mathbf{h}}_x = -\boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{xz}^e \mathbf{e}_y$$

$$\tilde{\mathbf{h}}_z = -\tilde{\gamma} \boldsymbol{\mu}_{zz}^{-1} \mathbf{e}_y$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{yy}) \tilde{\mathbf{h}}_y = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{zz}^{-1} \tilde{\mathbf{h}}_y$$

$$\mathbf{e}_x = -\boldsymbol{\epsilon}_{xx}^{-1} \mathbf{D}_{xz}^e \tilde{\mathbf{h}}_y$$

$$\mathbf{e}_z = -\tilde{\gamma} \boldsymbol{\epsilon}_{zz}^{-1} \tilde{\mathbf{h}}_y$$



$$-(\mathbf{D}_{yy}^h \boldsymbol{\mu}_{xx}^{-1} \mathbf{D}_{yy}^e + \boldsymbol{\epsilon}_{zz}) \mathbf{e}_x = \tilde{\gamma}^2 \boldsymbol{\mu}_{yy}^{-1} \mathbf{e}_x$$

$$\tilde{\mathbf{h}}_y = \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{yz}^e \mathbf{e}_x$$

$$\tilde{\mathbf{h}}_z = \tilde{\gamma} \boldsymbol{\mu}_{zz}^{-1} \mathbf{e}_x$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{zz}) \tilde{\mathbf{h}}_x = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{zz}^{-1} \tilde{\mathbf{h}}_x$$

$$\mathbf{e}_y = \boldsymbol{\epsilon}_{yy}^{-1} \mathbf{D}_{yz}^e \tilde{\mathbf{h}}_x$$

$$\mathbf{e}_z = \tilde{\gamma} \boldsymbol{\epsilon}_{zz}^{-1} \tilde{\mathbf{h}}_x$$



$$-(\mathbf{D}_{xx}^h \boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{xx}^e + \boldsymbol{\epsilon}_{zz}) \mathbf{e}_z = \tilde{\gamma}^2 \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_z$$

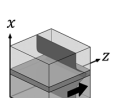
$$\tilde{\mathbf{h}}_x = -\tilde{\gamma} \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_z$$

$$\tilde{\mathbf{h}}_y = -\boldsymbol{\mu}_{yy}^{-1} \mathbf{D}_{xz}^e \mathbf{e}_z$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{zz}) \tilde{\mathbf{h}}_z = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{xx}^{-1} \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_x = -\tilde{\gamma} \boldsymbol{\epsilon}_{xx}^{-1} \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_y = -\boldsymbol{\epsilon}_{yy}^{-1} \mathbf{D}_{xz}^e \tilde{\mathbf{h}}_z$$



$$-(\mathbf{D}_{yy}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{yy}^e + \boldsymbol{\epsilon}_{xx}) \mathbf{e}_y = \tilde{\gamma}^2 \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_y$$

$$\tilde{\mathbf{h}}_x = \tilde{\gamma} \boldsymbol{\mu}_{xx}^{-1} \mathbf{e}_y$$

$$\tilde{\mathbf{h}}_z = \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{xz}^e \mathbf{e}_y$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{yy}) \tilde{\mathbf{h}}_y = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{xx}^{-1} \tilde{\mathbf{h}}_y$$

$$\mathbf{e}_x = \tilde{\gamma} \boldsymbol{\epsilon}_{xx}^{-1} \tilde{\mathbf{h}}_y$$

$$\mathbf{e}_z = \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xz}^e \tilde{\mathbf{h}}_y$$



$$-(\mathbf{D}_{yy}^h \boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{yy}^e + \boldsymbol{\epsilon}_{xx}) \mathbf{e}_x = \tilde{\gamma}^2 \boldsymbol{\mu}_{yy}^{-1} \mathbf{e}_x$$

$$\tilde{\mathbf{h}}_y = -\tilde{\gamma} \boldsymbol{\mu}_{yy}^{-1} \mathbf{e}_x$$

$$\tilde{\mathbf{h}}_z = -\boldsymbol{\mu}_{zz}^{-1} \mathbf{D}_{yz}^e \mathbf{e}_x$$

$$-(\mathbf{D}_{xx}^e \boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{xx}^h + \boldsymbol{\mu}_{zz}) \tilde{\mathbf{h}}_x = \tilde{\gamma}^2 \boldsymbol{\epsilon}_{yy}^{-1} \tilde{\mathbf{h}}_x$$

$$\mathbf{e}_y = -\tilde{\gamma} \boldsymbol{\epsilon}_{yy}^{-1} \tilde{\mathbf{h}}_x$$

$$\mathbf{e}_z = -\boldsymbol{\epsilon}_{zz}^{-1} \mathbf{D}_{yz}^e \tilde{\mathbf{h}}_x$$

Guessing the Eigenvalue

$$\tilde{\gamma}_0^2 \approx -n_{\text{guess}}^2$$

Call to `eigs()`

```
ev = -rib_n2^2;
[Exy, D2] = eigs(P*Q, NMODES, ev);
D = sqrt(D2);
NEFF = -1i*diag(D);
```

Guided-Mode Parameters

$$\gamma_m = k_0 \sqrt{\tilde{\gamma}_m^2}$$

$$\alpha_m = \text{Re}[\gamma_m]$$

$$\beta_m = \text{Im}[\gamma_m]$$

$$n_{m,\text{eff}} = \frac{\gamma_m}{jk_0}$$

Figure A.5 Summary of the formulation and implementation of FDFD for waveguide analysis. Slab waveguide analysis is shown in multiple orientations.

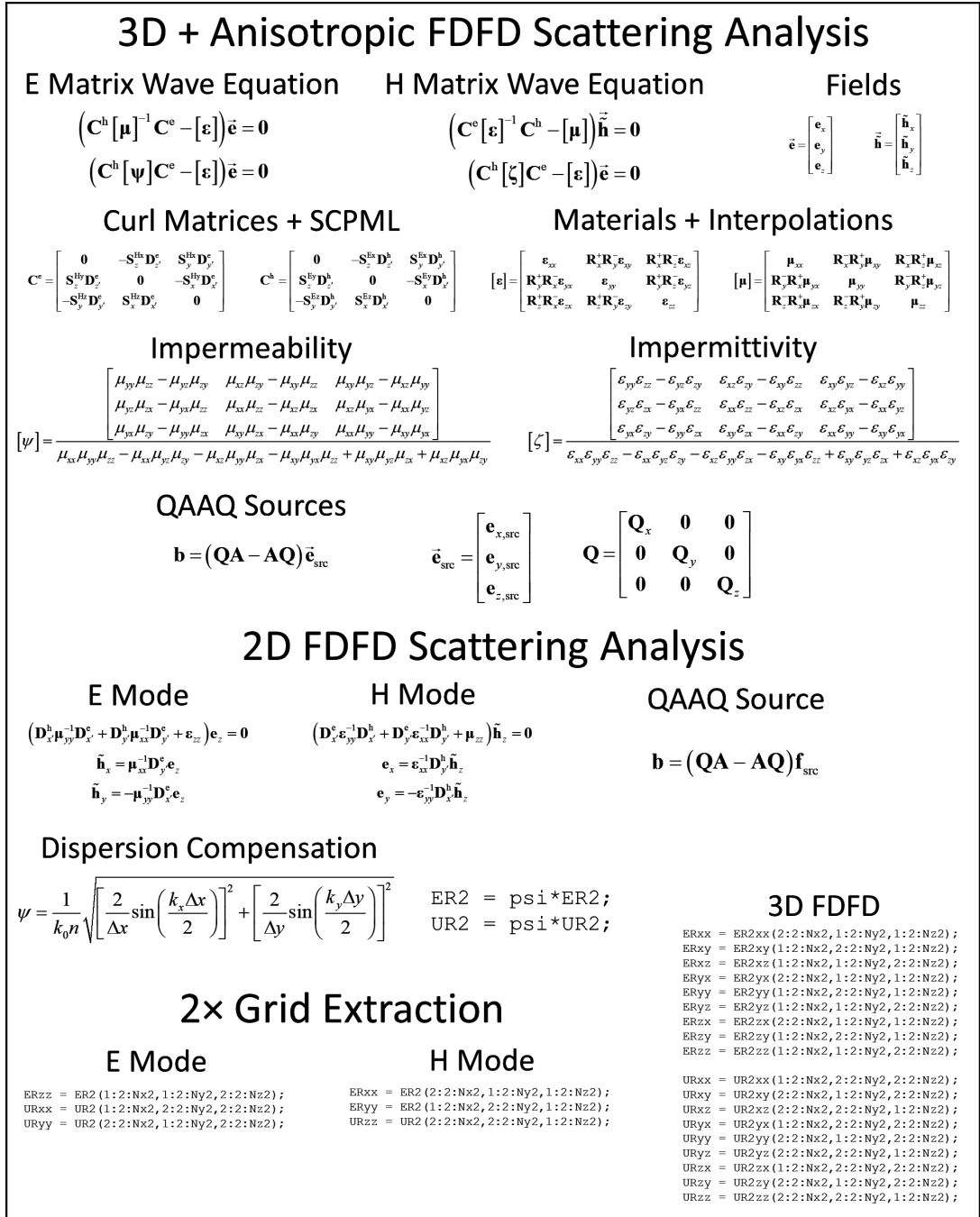
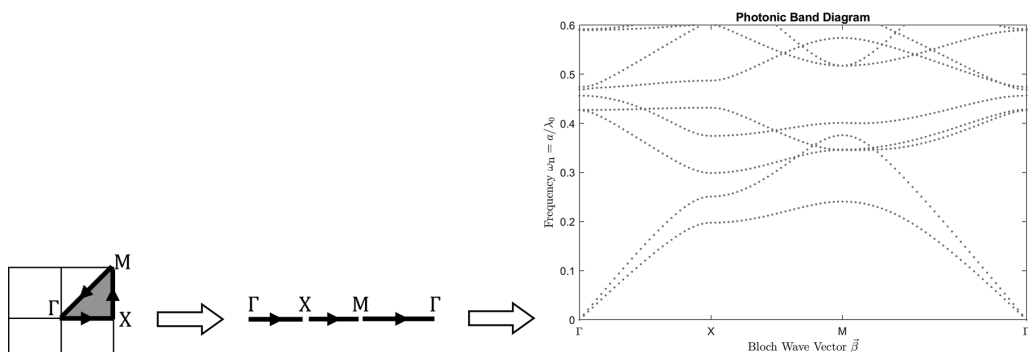
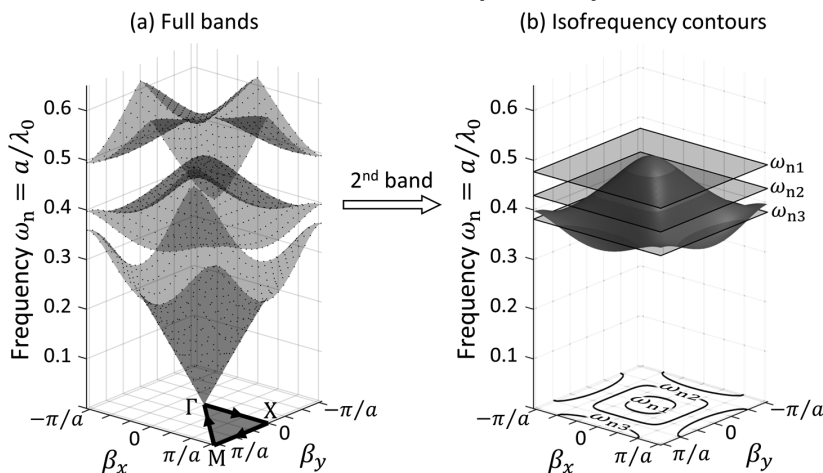


Figure A.6 Summary of the formulation and implementation of FDFD for scattering analysis.

Construction of Photonic Band Diagrams



Construction of Isofrequency Contours



FDFD Formulation

E Mode Eigenvalue Problem

$$-(\mathbf{D}_{xx}^h \mu_{yy}^{-1} \mathbf{D}_x^e + \mathbf{D}_y^h \mu_{xx}^{-1} \mathbf{D}_y^e) \mathbf{e}_z = k_0^2 \epsilon_{zz} \mathbf{e}_z$$

$$\tilde{\mathbf{h}}_x = \frac{1}{k_0} \mu_{xx}^{-1} \mathbf{D}_y^e \mathbf{e}_z$$

$$\tilde{\mathbf{h}}_y = -\frac{1}{k_0} \mu_{yy}^{-1} \mathbf{D}_x^e \mathbf{e}_z$$

H Mode Eigenvalue Problem

$$-(\mathbf{D}_x^e \epsilon_{yy}^{-1} \mathbf{D}_x^h + \mathbf{D}_y^e \epsilon_{xx}^{-1} \mathbf{D}_y^h) \tilde{\mathbf{h}}_z = k_0^2 \mu_{zz} \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_x = \frac{1}{k_0} \epsilon_{xx}^{-1} \mathbf{D}_y^h \tilde{\mathbf{h}}_z$$

$$\mathbf{e}_y = -\frac{1}{k_0} \epsilon_{yy}^{-1} \mathbf{D}_x^h \tilde{\mathbf{h}}_z$$

Figure A.7 Summary of the formulation of FDFD for photonic band analysis.

List of Acronyms and Abbreviations

1-D	One-Dimensional
2-D	Two-Dimensional
3-D	Three-Dimensional
ABS	Acrylonitrile Butadiene Styrene
BZ	Brillouin Zone
CEM	Computational Electromagnetics
CP	Circular Polarization
CPU	Central Processing Unit
EIM	Effective Index Method
EMF	Electromotive Force
FDFD	Finite-Difference Frequency-Domain
FDM	Finite-Difference Method
FDTD	Finite-Difference Time-Domain
FEM	Finite Element Method
FSS	Frequency Selective Surface
GMRF	Guided-Mode Resonance Filter
GPU	Graphical Processing Unit
IBZ	Irreducible Brillouin Zone
IFC	Isofrequency Contour
KCL	Kirchoff's Current Law
KVL	Kirchoff's Voltage Law
LCP	Left Circular Polarization
LHI	Linear, Homogeneous, and Isotropic
LP	Linear Polarization
NRW	Nicolson–Ross–Weir
OIC	Optical-Integrated Circuit
PBC	Periodic Boundary Condition
PEC	Perfect Electric Conductor
PMC	Perfect Magnetic Conductor
PML	Perfectly Matched Layer

POI	Plane of Incidence
RCP	Right Circular Polarization
SCPML	Stretched-Coordinate Perfectly Matched Layer
SF	Scattered-Field
SoI	Silicon-on-Insulator
SPP	Surface Plasmon Polariton
TE	Transverse Electric
TEM	Transverse Electromagnetic
TF	Total-Field
TFSF	Total-Field/Scattered-Field
TM	Transverse Magnetic
TO	Transformation Optics
UPML	Uniaxial Perfectly Matched Layer

About the Author

Raymond C. Rumpf has been a professor in the Department of Electrical and Computer Engineering at the University of Texas at El Paso in El Paso, TX, since 2010. He has focused most of his career on pursuing high-risk/high-payoff research in the areas of electromagnetics, photonics, three-dimensional printing, advanced packaging, and more. His research produced many significant breakthroughs, world records, and first-ever achievements that established him as a Fellow of SPIE and earned him into the Florida Tech Career Hall of Fame. Computation and simulation played a key role in all of his achievements. For a summary of his research and publications, visit <https://raymondcrumpf.com>.

Professor Rumpf is also a highly accomplished teacher. At his University, he has been recognized with numerous teaching awards, including the prestigious University of Texas Regents' Outstanding Teaching Award, which is the largest teaching award at the largest university system in the United States. For almost 10 years, Professor Rumpf has been developing online content for students to learn about electromagnetics and computation. To access this content, visit <https://empossible.net/>.

Professor Rumpf earned his BS and MS in electrical engineering from the Florida Institute of Technology in 1995 and 1997, respectively. He earned his PhD in Optics from the University of Central Florida in 2006. Prior to joining the University of Texas at El Paso, he was a principal investigator at Harris Corporation in Palm Bay, FL, and later the Chief Technology Officer for Prime Photonics LC in Blacksburg, VA. He has been awarded over a dozen U.S. Patents and authored dozens of peer-reviewed journal articles and conference proceedings. Other notable achievements include five world records for skydiving and teaching Maxwell's equations to his dog Rocky.

Index

A

Absorbing boundary
 about, 141
 development, 141–42
 doubly diagonally anisotropic, 143
 SCPML, 153–59, 309
 UPML, 143–47, 309
Acronyms and abbreviations, this book, 313–14
Acrylonitrile butadiene styrene (ABS) plastic, 291
addupm12d() function
 about, 149–50
 initialization, 150
 MATLAB program, 150–53
 using, 150–53
Algebra, MATLAB and, 3–8
Amplitude function, 114
Anisotropic materials, 37–38
Anisotropy, 269
Arrays
 binary, converting, 25
 complex, visualization of data in, 31–32
 defined, 8
 indexing, 8
 lattice to build into, 13
 MATLAB and, 10
 matrices versus, 8, 9
 meshgrid, 20
 visualization of data in, 29
Asymmetric diffraction gratings, 232, 284
Attenuation coefficient, 45, 65, 68

B

Backward division, 8
Biaxial materials, 39
Bloch modes, 200
Bloch wave vectors, 199, 206, 207
Block matrix, 7, 91, 165, 275
Boolean operations, 23–25
Boundary conditions
 diffraction orders and, 227
 Dirichlet, 81–82, 83, 91, 108–11
 numerical, 81–82
 periodic, 82, 85, 112–19
Brillouin zones (BZs), 199, 200

C

calcpml3d() function, 155–59
Centering algorithm, 17–19
Channel waveguides
 about, 63
 geometry of translational object, 162
 guided modes in, 63–64
 slab waveguide comparison, 64
 See also Waveguides
Characteristic impedance, 68, 193
Circles, adding to grids, 20–22
Circular polarization (CP), 47
Coaxial transmission lines, 66
Coefficients, terms, 4
Complex permittivity, 36
Complex propagation constant, 65, 67, 193, 195
Composite rotation matrix, 41
Computational electromagnetics (CEM)
 about, xv, xix
 convergence studies in, 256
 importance of, xiii
 modeling, xiv
 process block diagram, xx
 sweeping parameters and variables, xiv
Condition number, 278
Constitutive relations, 34
Continuous functions, 10
Contour() function, 245, 252
Convergence
 about, xx, 11
 FDFD codes and, 14
 grid resolution and, 187–88
 rate, 176, 181, 230, 249
 slow, 13, 18
 testing for, 174, 181
 tracking, xiii
Convergence studies
 about, 11–12, 256
 in computational electromagnetics, 256
 of grid resolution, 187
 importance of, 187–88
 parameter sweeps and, 188
 photonic bands, 209–10, 212
 point use on grid, 206

Convergence studies (*cont.*)
 scattering analysis, 239, 245–46
 SPP calculation, 191, 192
 use of, 256
 waveguide analysis and, 178, 184, 187–88
 Coupling length, 246
 Crossed-grating GMRF, 285, 287–90
 Crossed gratings
 defined, 60
 diffraction efficiency, 62
 diffraction from, 60–61
 period and symmetry of, 61
 Curl equations, Maxwell's, 42–45, 55, 99–103
 Cutoff frequency, 63

D

Derivative matrices
 about, 82
 convenient form, 84
 Dirichlet boundary conditions, 82–83, 108–11
 in finite-difference equations, 102
 function written in MATLAB for, 124
 numerical differentiation and, 85
 periodic boundary conditions (PBCs), 84, 115–19, 230
 relationship between, 119–20
 staggered grids and, 91
 for three-dimensional FDFD, 120–23
 for two-dimensional FDFD, 107–20
 Diag() function, 224
 Diagonal matrices
 about, 6
 inverse of PML terms, 273
 materials, 183
 PBCs and, 124
 Diagonals, 6
 Differential equations
 coupled, 89, 92
 differential, solving, 5–6
 discrete form, 85
 finite-difference approximations of, 85–86
 generic, converted to matrix form, 86
 matrix, solving, 87–89
 solving, 72
 solving a multivariable problem, 92–94
 Diffraction efficiency
 about, 59–60
 crossed gratings, 62
 defined, 56, 59
 diffraction gratings, 56–57, 59–60
 reflectance and, 56–57
 of transmitted diffraction orders, 283
 Diffraction gratings
 about, 56–57
 amplitude and, 57

asymmetric, 232, 284
 defined, 56
 diffraction efficiency and, 56–57, 59–60
 equation, 57–59
 generalization to crossed gratings, 60–62
 geometry, 57
 groove depth, 13, 230
 sawtooth, 233–39
 Diffraction orders, 56, 226–28, 280–81
 Directional coupler. *See* OIC directional coupler
 Direction plane, 258
 Dirichlet boundary conditions
 about, 81–82
 applied to magnetic fields, 179
 derivative matrices, three-dimensional FDFD, 120–21
 derivative matrices, two-dimensional FDFD, 108–11
 derivative matrix construction with, 83, 91
 interpolation matrices and, 274
 relationship between derivative matrices, 119–20
 rib waveguide analysis, 174, 179
 slab waveguide analysis, 181
 Discrete Fourier transform, 227, 281
 Dispersion relation, 49
 Double-curl operation, 43

E

Effective index method (EIM)
 about, 161, 169–71
 in reducing three-dimensional problems, 168, 171
 in reducing two-dimensional problem, 170
 Effective permittivity, 195, 296–98
 Effective refractive index, 65, 178, 179, 184–85, 187, 295, 299
 Eigenvalue matrix, 165–66
 Eigenvector matrix, 166, 184
 Eigs() function, 171–72, 177, 183, 204, 208
 Electric charge density, 34
 Electric conductivity, 36
 Electric current density, 34, 36
 Electric field intensity, 33–34
 Electric flux density, 34, 35
 Electric permittivity, 34
 Electromagnetic wave equation, 43–45
 Electromagnetic waves
 energy propagation in, 33, 34
 frequencies of, 290
 fundamental states for, 113
 in LHI media, 45–48
 polarization, 46–48
 splitting and dispersing, 56
 waveguides for, 62

Electromotive force (EMF), 34

Ellipses

adding to grids, 20–22

built into array, 21

built onto three-dimensional grid, 27

rotated, building into array, 22–23

Elliptical polarization (EP), 47–48

E mode

discrete equations (known frequency), 106

discrete equations (unknown frequency), 105

numerical plane wave for, 136

slab waveguide analysis, 167–68

3 x 3 grid for, 107

2x grid, 133

Yee cell for, 104

See also H mode

Envelope functions, 281

Excitation, terms, 4

Extinction coefficient, 44–45

F

Fast Fourier transform (FFT), 227, 306

`fd3d()` function, 285–87, 289

FDFD analysis

OIC directional coupler, 246–52

rib waveguide, 172–79

sawtooth diffraction grating, 233–39

scattering, 215–52

self-collimating photonic crystal, 239–46

slab waveguide, 179–85

surface waves, 191

of transmission lines, 192–97

of waveguides, 171–92

FDFD program structure (MATLAB), 1–3

FDFD simulation

Gaussian beam simulation, 31

grid setup, 14, 15

known frequency and, 103

of point source with/without absorbing boundary, 142

scattering from dielectric cylinder, 16

standard sequence, scattering analysis, 231–33

two-dimensional, UPML and, 147

`Fftshift()` function, 287

Finite-difference approximations

about, 72

of derivatives of magnetic field terms, 101

deriving expressions for, 73–76

of differential equations, 85–86

first-order derivative, 72

illustrated, 73, 100

interpolations and derivatives from four points, 79–80

interpolations and derivatives from three points, 76–78

interpolations and derivatives from two points, 78–79

Maxwell's curl equations, 99–103, 270–73

Maxwell's equations, 95–139

with periodic boundary conditions, 115

Finite-difference coefficients, 76

Finite-difference equations

definition of terms rule, 101

derivative matrices in, 102

for two-dimensional FDFD, 103–7

writing as a single matrix, 101–2

Finite-difference frequency-domain (FDFD)

about, xv, xix–xx, 71

advantages of, xiii

animating fields calculated by, 32

for calculating guided modes, 161–97

codes, accuracy and convergence, 14

common problems in, identifying, 266–67

FEM and, xix

formulation, 8

material loss and, 36

parameter sweeps with, 255–67

in photonic bands calculation, 199–213

PML for, 142

preparing Maxwell's equations for, 97–99

source functions in, 221

three-dimensional, 269–302

Finite-difference method

derivative matrices, 82

introduction to, 71

multiple variables and staggered grids, 89–94

numerical boundary conditions, 81–82

numerical differentiation, 80–81

solving matrix differential equations, 87–89

Finite element method (FEM), xix

`Floor()` function, 17, 265

Forward division, 8

Free space impedance, 46, 98

Free space permeability, 37

Free space permittivity, 37, 98

Free space wavenumber, 43–44

Frequency selective surfaces (FSSs)

ABS plastic, 291

defined, 290

frequency sweep, 293

geometry of, 291

grid calculation, 292

illustrated, 291

MATLAB code for, 292

simulation of, 290–94

transmittance of TE-polarized wave from, 293

Frequency sweep, 293, 297–98

Fresnel equations, 51–52, 143

G

Gauss' law for electric fields, 35
 Graphical processing unit (GPU), 279
 Grating equation, 57–59
 Grating vector, 57
 Grid parameters
 about, 10–11
 calculating, 11–15
 defined, 11
 resolution, 10–12
 three-dimensional grids, 25
 Grid resolution
 calculation of, 11–12, 228–29
 convergence and, 187–88
 OIC directional coupler, 249
 parameters, 10–12
 sawtooth diffraction grating, 236
 self-collimating photonic crystal, 243
 Grids
 adding circles to, 20–22
 adding ellipses to, 20–22
 adding rectangles to, 16–17, 18
 building geometries into, 15–25
 data, visualizing, 27–29
 defined, 10
 lattices, 13
 rotation, 22–23
 setup for FDFD, 14, 15
 setup in MATLAB, 8–15
 snapping to a critical dimension, 14
 staggered, 89–94
 three-dimensional, 25–27
 Yee scheme, 95–97, 99
 Guided-mode resonance filters (GMRFs)
 crossed-grating, 285, 287–90
 `fddfd2d()` function and, 258–60
 illustrated, 257
 MATLAB program to simulation, 260–62
 parameter sweeps and, 257
 reflectance, transmittance, power
 conservation, 290
 sensitivity of, 261, 285
 spectral response to, 262
 in testing new codes, 232–33
 Guided modes
 calculation, implementation of, 171–92
 calculation, purpose of, 66
 in channel waveguides, 63–64
 complex propagation constant and, 65
 decay of, 65
 effective refractive index, 65, 178, 179, 184–85
 electric field of, 64
 FDFD for calculating, 161–97
 hybrid mode calculation, 161–66
 information about, 65

Maxwell's equations and, 161
 propagating in +x-direction, 169
 propagating in +y-direction, 169
 propagating in +z-direction, 170
 transmission line analysis, 192–97
 waveguide mode calculation, formulation for, 166–71
 waveguide mode calculation, implementation of, 171–92

H

Hermitian transpose, 91
 H mode
 discrete equations (known frequency), 106–7
 discrete equations (unknown frequency), 105–6
 numerical dispersion and, 138
 slab waveguide analysis, 168
 3 x 3 grid for, 107
 2x grid, 133
 Yee cell for, 104
 See also E mode

I

Identity matrix, 4, 7, 278
 Ill-conditioned matrices, 278
`Imagesc()` function, 27
 Impedance. *See* Characteristic impedance; Free space impedance
 Impermeability, 277
 Interpolation matrices, 274
 Inverse FFT, 306
 Invisibility cloak
 grid strategy to simulation, 300
 MATLAB code for, 300–302
 simulation of, 300–302
 simulation results, 302
 TO technique and, 300
 Irreducible BZs (IBZs), 199, 200–201, 206–7, 210, 211
 Isofrequency contours (IFCs)
 about, 199
 concept, 201
 in dispersion property analysis, 213
 MATLAB code for calculating, 210–13
 professional, 213
 for TE and TM bands, 212
 Isotropic materials, 37, 39

J

Jacobi preconditioner, 278

K

Kirchhoff's current law (KCL), 67
 Kirchhoff's voltage law (KVL), 67

L

Lattice vectors, 199
 Left circular polarization (LCP), 47–48
 Linear, homogeneous, and isotropic (LHI) media
 dispersion relation in, 49
 electromagnetic waves in, 45–48
 Maxwell's equations for, 135
 Linear polarization (LP), 47–48
 Lorentz force law, 35
 Loss tangent, 37

M

Magnetic field intensity, 34
 Magnetic flux density, 34, 35
 Magnetic permeability, 34
 Material dispersion, 34
 MATLAB
 about, 3
 array indexing, 8–10
 backward division, 8
 Boolean operations in, 24
 commas and, 5
 forward division, 8
 grid parameters in, 10–15
 grid setup in, 8–15
 implementation for calculating SPPs, 188–92
 implementation of rib waveguide analysis, 172–79
 implementation of slab waveguide analysis, 179
 linear algebra and, 3–8
 matrix, 3–4
 matrix division in, 5
 meshgrid technique, 19–23
 postdivision, 8
 predivision, 8
 sparse matrices and, 111
 UPML implementation in, 149–53
 MATLAB codes
 addupml2d() function, 150–53
 animated GIF, 32
 building an ellipse into an array, 21–22
 building rotated ellipse, 22
 building small matrices, 7
 calcpml3d() function, 155–59
 crossed-grating GMRF, 288
 cylindrical source function calculation, 220
 diagonalization, 223–24
 Dirichlet boundary conditions, 83–84
 downloading, xiv
 fdfd2d() function, 258–60
 fdfd3d() function, 286
 FDFD program in, 1–3
 frequency selective surfaces (FSSs), 292
 fyeeeder2d() function, 124
 GMRF simulation, 260–62

GPU, 279
 ideal structure of a program, 2–3
 IFC calculation, 210–13
 invisibility cloak, 300–302
 metal polarizer analysis, 266–67
 metamaterials, 296–98
 microstrip analysis, 194–97
 numerical differentiation, 80–81
 OIC directional coupler, 247–48
 periodic boundary conditions, 85
 photonic band calculation, 205–10
 rib waveguide analysis, 173
 sawtooth diffraction grating, 235
 self-collimating photonic crystal, 241
 slab waveguide analysis, 180–85
 source function calculation, 220
 SPP calculation, 189–92
 start and stop array indices, 17
 three-dimensional grids, 25–26
 visualization of data in arrays, 28–29
 visualization of three-dimensional ellipsoid, 30
 yeeder3d() function, 128

Matrices

about, 3–4
 arrays versus, 8, 9
 block, 7, 91, 165, 275
 defined, 8
 diagonals of, 6
 identity, 4, 7, 278
 manipulating rows and columns in, 5
 rotation, 39–40, 41
 sparse, 6, 111, 125, 224
 special, 6–7
 tables of numbers, 5
 using, 4
 variables as, 8
 zero, 6–7, 124–25

Matrix algebra, 8**Matrix differential equations**

converting to, 86
 solving, 87–89

Matrix division, 5, 92, 277**Matrix equations**

block, 275
 coupled, 164
 differential, 86–89
 preconditioning, 278
 three-dimensional, 275–77
 for two-dimensional scattering analysis, 215–17
 wave, 275–79

MATrix LABoratory. See MATLAB**Matrix wave equation**

deriving, 276
 direct solution of, 278

- Matrix wave equation (*cont.*)
 - impermeability and, 277
 - three-dimensional, 275–78
 - vector components, 277
 - Maximum refractive index, 12
 - Maxwell's curl equations
 - electromagnetic wave equations and, 43–45
 - expansion in Cartesian coordinates, 42–43
 - finite-difference approximation of, 99–103, 270–73
 - Maxwell's equations
 - about, 33
 - constitutive parameters, 37–41
 - discrete form, summary of, 308
 - electromagnetic fields and, 62–63
 - electromagnetic wave equation and, 43–45
 - finite-difference approximation of, 95–139
 - in frequency-domain differential form, 35
 - general form of, 33
 - guided modes and, 161
 - incorporating UPML into, 146–47
 - for LHI media, 135
 - material parameters and, 34
 - in matrix form, 273–74
 - numerical solution to, 69
 - preparing for FDFD analysis, 97–99
 - relationships embedded in, xiii
 - scalability of, 68–69
 - with SCPML, 270
 - See also* Maxwell's curl equations
 - Meshgrid() function, 20
 - Meshgrids, 19–23, 26, 30, 206, 211
 - Metamaterials
 - composition of, 294
 - defined, 294
 - effective refractive index, 295, 299
 - frequency sweep, 297–98
 - geometry of translational object, 294
 - left-handed, parameter retrieval for, 294–99
 - MATLAB code for, 297
 - Nicolson-Ross-Weir (NRW) method and, 295, 298
 - parameter retrieval results, 298–99
 - Microstrip transmission lines, 66, 194–97
 - Multivariable finite-difference analysis, 92–94
- N**
- Nicolson-Ross-Weir (NRW) method, 295, 298
 - Normalized complex propagation constant, 163, 184, 196
 - Numerical boundary conditions, 81–82
 - Numerical differentiation, 80–81, 85
 - Numerical dispersion
 - about, 135
 - comparison, 137
 - concept, 136
 - H mode and, 138
 - relation, 138
 - scattering analysis and, 224–26
 - Yee grid and, 136
 - Numerical wave vectors, 136, 138
- O**
- OIC directional coupler
 - block diagram of FDFD analysis of, 248
 - coupling length, 246
 - FDFD analysis, 246–52
 - grid calculation, 250
 - grid resolution, 249
 - grid strategy for, 247
 - input waveguide analysis, 250
 - MATLAB code, 247–48
 - simulation results, 252
 - three-dimensional, reducing to two dimensions, 246
 - Optical integrated circuit (OIC), 169–71
 - Organization, this book, xxi–xxii
- P**
- Parameter sweeps
 - block diagrams of, 256
 - convergence studies and, 188
 - FDFD problems, identifying, 266–67
 - GMRFs and, 257
 - introduction to, 255
 - modifying FDFD for, 257–66
 - polarizer, 264
 - Pcolor() function, 27, 197, 252
 - Perfect electric conductor (PEC), 179
 - Perfectly matched layer (PML)
 - about, xix
 - absorbing boundary, 141–59
 - parameters, 141
 - size of, 228
 - terms, calculation of, 155–59
 - See also* Stretched-coordinate PML (SCPML); Uniaxial PML (UPML)
 - Perfect magnetic conductor (PMC), 179
 - Periodic boundary conditions (PBCs)
 - concept, 112
 - derivative matrices, 85, 230
 - derivative matrices, three-dimensional FDFD, 122–23
 - derivative matrices, two-dimensional FDFD, 115–19
 - diagonal matrix and, 124
 - efficiency, 112
 - for electric fields, 114
 - finite-difference approximations with, 115
 - limitation of, 113

- for magnetic fields, 115
 - N -point grid and, 82
 - in photonic bands calculation, 202
 - relationship between derivative matrices, 119–20
 - Periodic structures
 - as crossed grating, 280
 - double, reflection and transmission for, 280–82
 - simulation of, 258–60
 - three-dimensional FDFD function for simulation of, 285–87
 - Permittivity
 - averaging, 307
 - complex, 36
 - effective, 195, 296–98
 - electric, 34
 - relative, 37, 193, 36
 - Phase constant, 65, 68, 190
 - Photonic bands
 - about, 199
 - analysis, summary of formulation, 312
 - block diagram of FDFD calculation of, 204
 - Brillouin zones (BZs) and, 199, 200
 - construction of, 200
 - convergence study, 209–10
 - FDFD in calculating, 199–213
 - irreducible BZs (IBZs) and, 199, 200–201, 206–7, 210
 - isofrequency contours (IFCs) and, 199, 201, 210–13
 - periodic boundary conditions and, 202
 - professional, 210
 - for rectangular lattices, 199–201
 - square lattice for, 205
 - structure, MATLAB code, 205–10
 - unit cells and, 199, 206, 209
 - Plane of incidence (POI), 49
 - Polarization
 - equation for, 51
 - surface plasmon (SPP), 161
 - TE, 49–52
 - TM, 49–52
 - types of, 47–48
 - wave, 46–48
 - Polarizer
 - extinction ratio, 263, 265
 - frequency response, 266
 - MATLAB program to analyze, 262–66
 - parameter sweep, 264
 - TE and TM definitions, 263
 - Polynomial coefficients, 73–74
 - Postdivision, 8
 - Poynting vector, 52
 - “PQ” form, of eigenvalue problem, 165
 - Preconditioning, 278
 - Predivision, 8
 - Principal axes, 38, 39
 - Principal values, 38
 - Propagation constant
 - complex, 65, 67, 193, 195
 - normalized complex, 163, 184, 196
- Q**
- QAAQ equation, 218–20, 237, 245, 251, 302
 - QAAQ technique, 217
- R**
- Rectangles
 - adding to grids, 16–17
 - building onto a grid, 18
 - centered, 18–19
 - Rectangular lattices, photonic bands for, 199–201
 - Reflectance
 - about, 53
 - calculation of, 59, 226, 230
 - for doubly periodic structure, 280–82
 - GMRF, 290
 - overall, 56, 230
 - See also* Transmittance
 - Refractive index, 44
 - Relative permittivity, 37, 193
 - Rib waveguides
 - about, 172
 - building onto Yee grid, 175
 - convergence study of grid resolution, 187
 - geometry and coordinate setup, 173
 - grid strategy for modeling, 176
 - MATLAB implementation of analysis, 172–79
 - See also* Waveguides
 - Right circular polarization (RCP), 47–48
 - Right-hand rule, 46
 - Robustness, solver, 279
 - Rotating tensors, 41
 - Rotating vectors, 41
 - Rotation matrices, 39–40, 41
- S**
- Sawtooth diffraction grating
 - characteristics of, 233–34
 - convergence study, 239
 - diffraction efficiency calculations, 240
 - FDFD analysis of, 233–39
 - grid optimization, 236
 - grid resolution parameters, 236
 - grid strategy for, 234
 - illustrated, 234
 - MATLAB code to simulate, 235
 - Scalability, of Maxwell’s equations, 68–69

- Scattered-field (SF)
 - masking matrix, calculating, 222–24
 - quantity, 217, 218
 - region, 223, 231
- Scattering, at interface, 49–54
- Scattering analysis
 - block diagram of FDFD for, 229
 - formulation and implementation, summary of, 311
 - formulation of FDFD for, 215–17
 - implementation of FDFD method for, 228–52
 - incorporating sources and, 217–26
 - numerical dispersion and, 224–26
 - QAAQ technique and, 217–20
 - SF masking matrix calculation, 222–24
 - source field calculation, 220–22
 - two-dimensional, matrix wave equation for, 215–17
- Self-collimating photonic crystal
 - about, 239–40
 - FDFD analysis of, 239–46
 - grid calculation, 242–43
 - grid resolution, 243
 - grid strategy for, 240–41
 - lattice illustration, 241
 - lattice size, 242
 - MATLAB code, 241
 - simulation concept, 240
 - simulation results, 245
 - source field calculation, 244
- Self-collimation, 239
- Slab waveguide mode
 - animating, 185–87
 - second-order, single-frame of animation, 187
 - surface waves and, 188–89
 - visualization, 185
- Slab waveguides
 - about, 63
 - analysis, summary of, 310
 - channel waveguide comparison, 64
 - eigenmodes, calculating, 182–83
 - E mode analysis of, 167–68, 185
 - geometry for, 166
 - grid strategy for, 182
 - guiding onto Yee grid, 182
 - H mode analysis of, 168, 186
 - MATLAB implementation of analysis, 179–85
 - mode calculation, 166–71
 - modes propagating in +x-direction, 169
 - modes propagating in +y-direction, 169
 - modes propagating in +z-direction, 170
 - in other orientations, formulation for, 168–69
 - See also* Waveguides
- Snell's law of reflection, 51, 143
- Source field, calculating, 220–22
- Source wave vector, 226
- Sparse() function, 224
- Sparse matrix
 - adding and deleting elements from, 125
 - defined, 6
 - MATLAB and, 111
- Spdiags() function, 84, 125
- SPPs
 - about, 189
 - calculating, 188–92
 - calculation results, 192
 - convergence study, 192
 - geometry, 189
 - phase constant, 190
 - typical behavior of, 191
 - See also* Surface waves
- Staggered grids, 89–94
- Stretched-coordinate PML (SCPML)
 - about, 141
 - absorbing boundary, 153–59
 - absorbing boundary, summary, 309
 - arrays, 155, 156
 - calcpml3d() function and, 155–59
 - derivation of, 153
 - equations, 154
 - implementation of, 155
 - matrix conditioning, 142, 155
 - Maxwell's equations with, 270
 - three-dimensional FDFD, 269
- Substrate medium, 63
- Superstrate medium, 63
- Surface plasmon polarization (SPP), 161
- Surface waves
 - defined, 188
 - FDFD analysis, 191
 - slab waveguide modes and, 188–89
 - SPP calculation, 189–92
- T**
 - Telegrapher equations, 67
 - Text() function, 184
 - TF/SF interface, 220, 221, 222, 230, 232, 245
 - TF/SF technique, 215–17
 - Three-dimensional 2x grid, 134–35
 - Three-dimensional FDFD
 - about, 269
 - derivative matrices for, 120–23
 - finite-difference approximation of Maxwell's curl equations, 270–73
 - formulation of FDFD for, 269–77
 - implementation of, 282–302
 - incorporating sources into, 277–78
 - interpolation matrices, 274
 - iterative solution for, 278–79
 - Maxwell's equations in matrix form, 273–74

- SCPML, 269
 - standard sequence of simulations for, 282–85
 - three-dimensional matrix wave equation, 275–77
 - UPML, 269
 - Three-dimensional grids
 - ellipsoid built onto, 27
 - grid parameters, 25
 - MATLAB codes, 25–26
 - visualization of data in, 29–30
 - Three-dimensional matrix wave equation, 275–77
 - Total-field (TF)
 - finite-difference equations, 218
 - quantity, 217, 218, 220
 - region, 223
 - See also* TF/SF interface
 - TO technique, 300
 - Transmission lines
 - analysis implementation, 192–97
 - coaxial, 66
 - defined, 63
 - equivalent circuit model for, 66–67
 - microstrip, 66, 194
 - normalized magnetic fields and, 193
 - parameters, 66–68
 - parameters, calculating, 195–96
 - Transmission plane, 258
 - Transmittance
 - about, 53
 - calculation of, 59, 226, 230
 - for doubly periodic structure, 280–82
 - GMRF, 290
 - See also* Reflectance
 - Transverse electric (TE) polarization, 49–52
 - Transverse electromagnetic (TEM) mode, 63
 - Transverse magnetic (TM) polarization, 49–52
 - 2x grid
 - about, 131–32
 - arrays, extracting tensor element arrays from, 134
 - E and H modes, 133
 - simulated device and, 132
 - technique, 131–35
 - three-dimensional, 134–35
 - Yee grid extraction from, 132, 133
 - Two-dimensional FDFD
 - derivation of E mode equations (known frequency), 106
 - derivation of E mode equations (unknown frequency), 105
 - derivation of H mode equations (known frequency), 106–7
 - derivation of H mode equations (unknown frequency), 105–6
 - derivative matrices, Dirichlet boundary conditions, 108–11
 - derivative matrices, periodic boundary conditions, 115–19
 - derivative matrices for, 107–20
 - finite-difference equations for, 103–7
 - relationship between derivative matrices, 119–20
 - Two-dimensional simulation, 54, 55–56
- U**
- Uniaxial PML (UPML)
 - about, 141, 143
 - absorbing boundary, derivation of, 143–46
 - absorbing boundary, summary, 309
 - conductivities, 147–48
 - conductivity profiles, 148
 - implementation in MATLAB, 149–53
 - incorporating into Maxwell's equations, 146–47
 - parameters, 145–46
 - parameters, calculating, 147–49
 - size of, 149
 - tensor, 146
 - three-dimensional FDFD, 269
 - for two-dimensional simulations, 147
 - Unit cells, 199, 206, 209, 212
 - Unknowns, terms, 4
- V**
- Vandermonde matrix, 74
 - Visualization
 - complex data, 31–32
 - data on grids, 27–29
 - importance of, xiii
 - slab waveguide mode, 185
 - source field f_{src} for a guided mode, 251
 - techniques, 27–32
 - three-dimensional data, 29–30
 - Volume charge density, 35
- W**
- Waveguides
 - analysis, summary of formulation and implementation, 310
 - channel, 63, 64, 162
 - defined, 62
 - FDFD analysis of, 171–92
 - modes and parameters, 63–66
 - rectangular metal, 161
 - rib, 172–79, 187
 - slab, 63, 64, 166–71
 - types of, 63
 - Wavelength sweep, 261
 - Wavenumber, 43–44

Wave propagation

- x -direction, 145

- xy plane, 103, 205, 216

- y -direction, 145

- z -direction, 55, 205

Well-conditioned matrices, 278**Y**

- `yeeder2d()` function, 124–28, 237

- `yeeder3d()` function, 128–31

Yee grids

- building devices onto, best practices, 305–12

- illustrated, 97

- importance of, xiv

- Maxwell's equations and, xxi

- metal square placement onto, 305

- numerical advantages, 131

- numerical dispersion caused by, 136

- size of, 150

- staggered layout on, 101, 132

- summary of, 308

- 2x grid overlaid onto, 133

- Yee grid scheme, 95–97, 99, 163

Z

- Zero column vector, 6

- Zero matrix, 6, 7, 124–25

Artech House Applied Photonics Library

Brian Culshaw and Marc Wuilpart, Series Editors

The ABCs of Fiber Optic Communication, Sudhir Warier

Acousto-Optic Signal Processing: Fundamentals and Applications, Pankaj K. Das and Casimer De Cusatis

Advanced Optical Communication Systems and Networks, Milorad Cvijetic and Ivan B. Djordjevic

Applications of Modern RF Photonics, Preetpaul Singh Devgan

Bistabilities and Nonlinearities in Laser Diodes, Hitoshi Kawaguchi

Chemical and Biochemical Sensing with Optical Fibers and Waveguides, Gilbert Boisdé and Alan Harmer

Coherent and Nonlinear Lightwave Communications, Milorad Cvijetic

Coherent Doppler Wind Lidars in a Turbulent Atmosphere, Viktor Banakh and Igor Smalikho

Coherent Lightwave Communication Systems, Shiro Ryu

Digital Optical Measurement Techniques and Applications, Pramond Rastogi, Editor

DWDM Fundamentals, Components, and Applications, Jean-Pierre Laude

Electromagnetic and Photonic Simulation for the Beginner: Finite-Difference Frequency-Domain in MATLAB®, Raymond C. Rumpf

Electro-Optical Systems Performance Modeling, Gary Waldman

Elliptical Fiber Waveguides, R. B. Dyott

Engineering Optical Networks, Sudhir Warier

Fiber Bragg Gratings: Fundamentals and Applications in Telecommunications and Sensing, Andrea Othonos and Kyriacos Kalli

Fiber Optics Communications, Henry F. Taylor

The Fiber-Optic Gyroscope, Third Edition, Hervé C. Lefèvre

Fiber-Optic Sensors for Biomedical Applications, Daniele Tosi and Guido Perrone

Field Theory of Acousto-Optic Signal Processing Devices, Craig R. Scott

Finite Element Modeling Methods for Photonics, B. M. Azizur Rahman and Arti Agrawal

Frequency Stabilization of Semiconductor Laser Diodes, Tetsuhiko Ikegami, Shoichi Sudo, and Yoshihisa Sakai

Fundamentals of Multiaccess Optical Fiber Networks, Dennis Mestdagh

Germanate Glasses: Structure Spectroscopy and Properties, Alfred Margaryan and Michael Piliavin

Handbook of Distributed Feedback Laser Diodes, Second Edition, Geert Morthier and Patrick Vankwikelberge

Highly Coherent Semiconductor Lasers, Motoichi Ohtsu

High-Power Optically Activated Solid-State Switches, Arye Rosen, Fred Zutavern
Integrated Optics: Design and Modeling, Reinhard Marz
Introduction to Electro-Optical Imaging and Tracking Systems, Shaheen A. Hovanessian, Khalil Seyrafi
Introduction to Glass Integrated Optics, S. Iraj Najafi
Introduction to Infrared and Electro-Optical Systems, Second Edition, Ronald G. Driggers, Melvin H. Friedman, and Jonathan Nichols
Introduction to Radiometry and Photometry, Second Edition, William Ross McCluney
Introduction to Semiconductor Integrated Optics, Hans P. Zappe
Laser Communications in Space, Stephen Lambert
Liquid Crystal Devices: Physics and Applications, Vladimir G. Chigrinov
Lithium Niobate Photonics, James E. Toney
Military Laser Technology and Systems, David H. Titterton
Natural Photonics and Bioinspiration, Sébastien R. Mouchet and Olivier Deparis
New Photonics Technologies for the Information Age: The Dream of Ubiquitous Services, Shoichi Sudo and Katsunari Okamoto, editors
Optical Document Security, Third Edition, Rudolf L. van Renesse
Optical FDM Network Technologies, Kiyoshi Nosu
Optical Fiber Amplifiers: Materials, Devices, and Applications, Shoichi Sudo, editor
Optical Fiber Communication Systems, Leonid Kazovsky, Sergio Benedetto and Alan Willner
Optical Fiber Sensors, Volume One: Principles and Components, Brian Culshaw and John P. Dakin
Optical Fiber Sensors, Volume Two: Systems and Applications, Brian Culshaw and John P. Dakin
Optical Fiber Sensors, Volume Three: Components and Subsystems, John Dakin and Brian Culshaw, editors
Optical Fiber Sensors, Volume Four: Applications, Analysis, and Future Trends, John Dakin and Brian Culshaw, editors
Optical Interconnection: Foundations and Applications, John Caulfield and Christopher S. Tocci
Optical Measurement Techniques and Applications, Pramod Rastogi
Optical Network Theory, Yitzhak Weissman
Optical Transmission for the Subscriber Loop, Norio Kashima
Optical Transmission Systems Engineering, Milorad Cvijetic
Optoelectronics for Low-Intensity Conflicts and Homeland Security, Anil K. Maini
Optoelectronic Techniques for Microwave and Millimeter-Wave Engineering, William M. Robertson
Passive Optical Components for Optical Fiber Transmission, Norio Kashima
Polarization in Optical Fibers, Alan Rogers
Photonic Applications for Radio Systems and Networks, Fabio Cavaliere and Antonio D'Errico

Principles of Modern Optical Systems, Volume 1, Deepak Uttamchandani and Ivan Andonovic, editors

Principles of Modern Optical Systems, Volume 2, Deepak Uttamchandani and Ivan Andonovic, editors

Reliability and Degradation of III-V Optical Devices, Osamu Ueda

Securing Information and Communications Systems, Stephen Furnell, Sokratis Katsikas, Javier Lopez, and Ahmed Patel

Semiconductor Raman Lasers, Ken Suto and Jun-Ichi Nishizawa

Semiconductors for Solar Cells, Hans Joachim Moller

Semiconductor Quantum Wells and Superlattices for Long-Wavelength Infrared Detectors, M. Omar Manasreh

Signal Processing and Performance Analysis for Imaging Systems, S. Susan Young, Ronald G. Driggers, and Eddie L. Jacobs

Single-Mode Optical Fiber Measurement: Characterization and Sensing, Giovanni Cancellieri and Franco Chiaraluce

Smart Structures and Materials, Brian Culshaw

Substrate Surface Preparation Handbook, Max Robertson

Surveillance and Reconnaissance Imaging Systems: Modeling and Performance Prediction, Jon C. Leachtenauer and Ronald G. Driggers

Tunable Laser Diodes, Markus-Christian Amann and Jens Buus

Ultrafast Diode Lasers: Fundamentals and Applications, Peter Vasilev

Understanding Optical Fiber Communications, Alan Rogers

Wavelength Division Multiple Access Optical Networks, Andrea Borella, Giovanni Cancellieri, and Franco Chiaraluce

Wired and Wireless Seamless Access Systems for Public Infrastructure, Tetsuya Kawanishi

For further information on these and other Artech House titles, including previously considered out-of-print books now available through our In-Print-Forever® (IPF®) program, contact:

Artech House
685 Canton Street
Norwood, MA 02062
Phone: 781-769-9750
Fax: 781-769-6334
e-mail: artech@artechhouse.com

Artech House
16 Sussex Street
London SW1V 4RW UK
Phone: +44 (0)20 7596-8750
Fax: +44 (0)20 7630-0166
e-mail: artech-uk@artechhouse.com

Find us on the World Wide Web at: www.artechhouse.com
