AN FDTD CODE FOR MOBILE TELECOMMUNICATIONS ANTENNA DESIGN

MEIDE QIU

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

An FDTD Code for Mobile Telecommunications Antenna Design

By

©Meide Qiu

A thesis submitted to the School of Graduate Studies

in partial fulfillment of the requirement for

the degree of Master of Engineering

Faculty of Engineering and Applied Science

Memorial University of Newfoundland

August 1998

St. John's Newfoundland Canada

Abstract

In wireless telecommunications, antennas play a very important role where they can either enhance or constrain system performance. The design characterization of such antennas is dependent upon, to large extent, the development of simulation tools that can accurately model general antenna system topologies. In this thesis, where the basic theory of finite-difference time domain method (FDTD) is reviewed, an FDTD code is developed for antenna design, with particular emphasis on the modeling of the source region. The results from the developed code are compared with those produced by a moment method based code (*Numerical Electromagnetic Code -2:* NEC2) and very good agreement is obtained. The effects of the FDTD cell size and the distance between the outer absorbing boundary and the antenna on accuracy are explored. Some criteria for the choice of above parameters in FDTD calculation are given for antenna design. Using the developed code, a new planar monopole antenna which operates at dual wide band (800MHz band and 1800MHz band) is developed by cutting slots and determining the geometrical parameters. This dual frequency performance is required for the existing and potential mobile communication system providing analogue and digital services.

Acknowledgements

The author would like to thank Dr. B. P. Sinha and Dr. S. A. Saoudy for their supervision of this research.

The financial support from the Faculty of Engineering and Applied Science of the Memorial University of Newfoundland and the research grant from National Science and Engineering Research Council (NSERC) are gratefully acknowledged.

Contents

Abstract ii

Acknowledgments iii

List of Figures vi

List of Tables vi

1 Introduction 1

1.1 Statement of the Problem 1

1.2 Literature Review 4

1.2.1 Antennas for Mobile Telecommunications 4

1.2.2 Numerical Tools for Antenna Design 7

1.3 The Scope of the Thesis 9

1.4 Organization of the Thesis 10

2 FDTD Approach 11

2.1 Introduction 11

2.2 Maxwell's Equations 12
2.3 Discretization of Maxwell Equations 14
2.4 Radiation Boundary Condition 19
2.4.1 The Sommerfeld Radiation Condition 20
2.4.2 Absorbing Boundary Condition 21
2.5 The Cell Size and Time Step for Stability 21
2.6 Antenna Parameters in FDTD 23
2.6.1 Input Impedance, Power and Efficiency 23
2.6.2 Far Fields 25
2.7 Special Consideration 28
2.7.1 Thin Wire Structure 28
2.7.2 Antenna Feed 31
2.7.3 Source Forms 34

3 Development of the FDTD Code **44**

3.1 FDTD Code 44
3.1.1 Header File 49
3.1.2 Initialization 49
3.1.3 Field Computation within the Boundary 50
3.1.4 Fields on the Artificial Boundary 51
3.1.5 Thin Wire Structure 51
3.1.6 Steady-State Response 51

3.1.7 Data Input 56
3.1.8 Data Output 59
3.1.9 Procedure for Using the Code 59
3.2 Validation of Present FDTD Code 59
3.3 Dipole 60
3.3.1 Radiation Pattern 60
3.3.2 Input Impedance 64
3.4 Loop Antenna 68
3.5 A Monopole on a Finite Conducting Plate 75

4 A Proposed Mobile Communication Antenna Design 79

4.1 A Planar Monopole on a Conducting Box 81
4.2 The Implementation of Dual-Band Operation 83
4.3 Development of Controlling Two Specific Resonant Frequencies 86

5 Conclusion and Future Work 95

A A Input Data File Example 102

A.1 The Data File "ant.dat" 102
A.2 The Data File "wire.dat" 103
A.3 A Program to Create "material.dat' 103

B The FDTD Code 105

List of Figures

2.1 Positions of the field components about a unit cell of the FDTD grid 16

2.2 Find the far field from equivalent currents 27

2.3 The geometry of thin wire in a subcell approach 30

2.4 The thin gap method of voltage source 32

2.5 The equivalent magnetic current method of voltage source 33

2.6 The waveform of Gaussian pulse with amplitude of 1 36

2.7 The waveform of Gaussian pulse with amplitude of 1 in FDTD 38

2.8 The effects of α on high frequencies with $\beta = 32$. 39

2.9 The effects of β on frequency band with $er_0 = 16$ 40

2.10 The waveform of Rayleigh pulse 41

2.11 The spectrum of Rayleigh pulse with $\beta = 32$ and $er_0 = 16$ 43

3.1 Flow chart of a C++ program for FDTD algorithm 45

3.2 Subroutine and data file linkage chart 48

3.3 The form of a field component in time domain 53

3.4 Steady-state wave form 54

3.5 Method of finding the steady-state phase information 56

3.6 The radiation pattern of a dipole at 3GHz obtained using different FDTD cell size 61

3.7 The radiation patterns of the dipole at 3GHz with different Mur distance 63

3.8 The radiation patterns of the dipole at 2GHz with different cell size . 64

3.9 The effects of Mur distance on the accuracy of the radiation patterns 65

3.10 The effects of Mur distance on the accuracy of the input impedance . 66

3.11 The effects of the cell size on the accuracy of the input impedance . . 67

3.12 The radiation pattern of a loop antenna using delta-gap voltage source 70

3.13 The resultant current in the source region of the loop antenna 71

3.14 The input impedance of an loop antenna using delta-gap voltage source 72

3.15 The resultant current in source region of loop antenna with Rayleigh pulse 73

3.16 The input impedance of an loop antenna with Rayleigh pulse 74

3.17 The radiation pattern of a monopole on $\lambda \times \lambda$ ground 76

3.18 The input impedance of a $7.5cm$ monopole on a $30cm \times 30cm$ ground 77

3.19 The radiation pattern of a monopole on $2\lambda \times 2\lambda$ ground 78

4.1 The geometry of planar monopole on a conducting box 82

4.2 The return loss of a planar monopole mounted on ground and a box . 83

4.3 The implementation of dual-band operation by cutting a slot 84

4.4 Input impedance of the antenna 85

4.5 Return loss at the feed of the antenna 86

4.6 Radiation pattern of the antenna 87

4.7 Geometry of the new planar monopole 88

4.8 Return loss of the antenna with and without the second slot 89

4.9 Return loss of the antenna for different major axis length 90

4.10 Return loss of the antenna for different size of the second slot 91

4.11 Return loss of the antenna with dielectric cover 93

List of Tables

3.1 The functions of the subroutines 47

3.2 The data format of file ant.dat 57

3.3 The data format of file wire.dat 58

3.4 Feed forms and pulse forms for loop antennas 69

3.5 Feed forms and pulse forms for open circuit type antennas 69

Chapter 1

Introduction

1.1 Statement of the Problem

The area of wireless communications is developing rapidly, which is well known among not only the technical community but also the general population. The public's continuing demands for personal and mobile communications causes the explosive growth in terrestrial and satellite communications systems that promise to become the preferred medium of telecommunications in the future.

Antennas used in wireless communications play a very important role as they can either enhance or constrain the system performance. Present mobile communication antennas include whip antennas sticking out from automobile bodies, sleeve antennas and inverted-F antennas installed on portable telephones, and microstrip antennas and loop antennas for UHF pagers. A number of industrial groups are currently

planning to construct global satellite systems to provide personal communications services PCS (typically voice, data and fax) to users who are supposed to employ small, hand-held, cellular-type handsets. The demand for developing corresponding mobile antennas is increasing by the day.

The basic requirements for mobile antennas are small size, lightweight, and low cost. Since the inception of antennas, there has been continuing interest in reducing their physical size. Much more attention has been put on this especially after the appearance of hand-held, cellular-type handsets. Lightweight and low cost are also the key factors affecting the acceptance of a handset among the public. The specific environment for the handset antenna — the limited space provided on the surface of the housing unit of the handset as well as other requirements for the antenna present some difficulties for antenna designers. First, the radiation characteristic of an antenna on a small housing unit differs from that radiating in free space on a large conducting plane and also depends on the composition of the unit. The input impedance bandwidth can be quite narrow when an antenna is located on such a specific environment. Moreover, the antenna characteristics can be greatly affected by its operator because he/she is within the near field region of the antenna.

In such a complex situation, the antenna design is far beyond the scope of the early methods, which were based upon analytical techniques that attempted to generate closed-form solutions expressible in terms of known functions. Characterization of such antennas for hand-held communications devices such as portable phones has

to rely on, to large extent, the development of simulation tools that can accurately model general topologies, including conductors, dielectrics, thin wires and lumped elements.

Among various numerical methods, the method of moment (MOM) and finite-difference time-domain method (FDTD) are preferred most frequently. MOM is a very efficient method for wire antennas, especially at low frequencies. It can also be used for antennas consisting of solid bodies, usually, by using wire-grids, which requires a large memory. For modern sophisticated antennas which involve complicated geometry and materials structures, the large number of the unknowns needed for accurate results is a big burden for the matrix manipulation which is unavoidable in MOM.

The finite-difference time-domain (FDTD) method is a simple and elegant way to solve Maxwell's equations. Unlike MOM, FDTD does not involve matrix operation and moreover, the complexity of geometry and material does not make the algorithm more complicated. So a general purpose program can be developed, which is particularly useful in modern antenna design.

Although past contributions in this area have demonstrated the effectiveness of the FDTD approach in characterizing antenna configurations, only a limited amount of research has appeared relating to the simulation of practical antenna geometry operating in their true radiating environment. A few commercial FDTD programs such as *XFDTD* and *FIDELITY* are available, but are very expensive — thousands

of U.S. dollars. Therefore, it is very useful to develop our own FDTD code for sophisticated antenna design. This program will add to the faculty of engineering infrastructure in electromagnetic codes.

Once the code is validated through comparison of its results with those obtained from the moment method code (NEC) for special cases, it will be used to analyze a dual wide band antenna. Through variation of the antenna parameters, a design that can provide required characteristics can be achieved.

1.2 Literature Review

1.2.1 Antennas for Mobile Telecommunications

With the rapid development of mobile communications, the progress in mobile antenna design is keeping this trend. Antennas used in various antenna systems are different, and it is difficult to set a general rule for interfacing them with the rest of transmitting/receiving hardware assembly. Here, the antennas for portable phones will be introduced. Currently, frequencies from 800MHz to 1.8GHz have been assigned for mobile telephones, and future allocation of higher frequency bands is being considered. At present, the most widely used antennas are monopole antennas, sleeve antennas as well as recently developed low-profile antennas such as microstrip antennas and planar inverted F antennas (PIFA).

Monopole antenna: The monopole antenna is the most commonly used mobile antenna because of its broad band characteristics and simple structure. This type of antenna normally employs a flexible antenna element, so it is also called the whip antenna. If the radiating element is mounted on an infinite ground plane, the characteristics of the antenna are similar to those of a dipole. In practice, the monopole is not simply half of a dipole and even very large ground planes give radiation patterns significantly different from that on an infinite plane [1, 2]. In actual usage of our consideration, the "ground" is a portable housing unit, the input impedance and radiation patterns of the antenna depend on the actual size and composition of the housing unit [3, 4].

Sleeve antenna The radiating structure of a sleeve dipole is an asymmetric dipole made of conductors of different diameters and slightly different length. The thinner conductor is normally the inner conductor of the coaxial line feeding the antenna, and the larger diameter conductor, which is shorted to the braid of the coaxial line [5], must provide effective choking of the RF currents at its own open end and also one-half of the radiating dipole so that most of the antenna current does not leak into the outer surface of the coaxial cable. The antenna has almost the same characteristics as a monopole antenna, but it does not require a ground plane, so the gain degradation due to the mounting location is less than that experienced with whip antennas. The bandwidth limitation of this antenna is dictated more by radiation pattern performance than by impedance variation, since the deviation of operation

frequency will lead to the RF current leaking to the outer surface of the coaxial feeding line and thus changing the radiation pattern greatly[6].

The disadvantage of the whip antenna and the sleeve antenna is that they are not rigid, so they are easy to break. The implementation of dual band without increasing the size is also difficult.

Low-profile antennas: Microstrip antennas [7] and planar inverted F antennas [6] (PIFA) are well known as typical low-profile antennas. Microstrip antennas, also called patch antennas, are constructed by printing conductors on dielectric substrates. This antenna is derived from microstrip resonators, by using the radiation loss of the resonators in a positive manner. Therefore, the bandwidth of this antenna is basically narrow. PIFA was developed from the inverted L antenna. Because of its low-profile, it has been used in some cellular phones as a built-in antenna. PIFA has the same disadvantage as microstrip antenna, though some researchers are trying to improve its bandwidth by various methods [8].

To satisfy the great demand for the rapid growth of mobile telephones, it is often necessary to add a new frequency band to the existing system. For example, in the United States, the existing Advanced Mobile Phone Service (AMPS) analog standard and Interim Standard-54 (IS-54) digital standard cover the frequency range from 824MHz to 894MHz while the Global System for Mobile (GSM) communication, which is the digital telephone standard developed primarily in Europe and in Asia, covers the frequency range from 890MHz to 960MHz. The new generation of personal

communication systems (PCS) such as DCS-1800 has frequency band 1.710-1.880 GHz. The co-existence of GSM and DCS with a dual standard providing analogue and digital services in the same network means that the corresponding antennas should have the capability of operating at dual frequency bands (824-960MHz and 1.71-1.88GHz). The development of corresponding dual frequency antennas is in great demand.

It is understood that in any particular design only some of the objectives will be achievable, and each case must be treated as a separate entity. In this thesis, a dual wide-band antenna is our expectation. Some efforts have already been put to the development of dual-frequency antennas [9, 10]. Unfortunately, the bandwidth is not very wide. The planar monopole antenna has recently been proposed [11, 12]. This type of antenna has a very large impedance bandwidth when mounted on a large conducting plane. However, the application of this type antenna to mobile communication, in which the size of housing unit is generally small, has not been found in the literature.

1.2.2 Numerical Tools for Antenna Design

Antenna design is increasingly dependent on computer-aided design (CAD) based on well-known mathematical methods. When mobile antennas are essentially bent wire sections, they can be modeled with wire-grid modeling [13] provided that the housing units are conducting. Actually, wire-grid model has been used in several

reports dealing with portable radio units [14, 15]. But for radiating structures having arbitrary shape and composed of layers of heterogeneous material, one has to use other methods. One of the latest techniques is the finite difference time domain (FDTD) method, which shows much promise for complicated structures and material, and this can include components within a dielectric outer case.

The present popular FDTD was first proposed by Yee [16] in 1966. Yee used an electric-field grid, which was offset both spatially and temporally from a magnetic-field grid, to obtain update equations which yield the present fields throughout the computational domain, in terms of the past fields. The update equations are used in a leapfrog scheme, to incrementally march the E and H fields forward in time. In the 1970's, the FDTD method was not paid much attention because the outer absorbing boundary condition was not good enough to simulate the propagation of the out-going waves and also computer facilities were very slow. With the appearance of a better absorbing boundary condition [17, 18] and the decrease of computer cost, the interest in the FDTD method began to soar. The method has been used in hundreds of applications, ranging from electromagnetic scattering to radiation.

The application of FDTD to antenna problems is relative recent than other problems such as scattering, but the number of published papers is accumulating [2, 3, 4, 19, 8, 20, 21].

Despite many good papers on FDTD analysis of antenna problems, there are still some issues that need further discussion. When calculating the input impedance of

an antenna, the antenna is pulse excited so that wide band results can be obtained from a single FDTD computation. For the pulse, some researchers use Gaussian pulse[22], other researchers uses Rayleigh pulse[8]. The issue is which pulse is better for a specific antenna problem. The effects of cell size and the distance between the absorbing boundary condition and the antenna on the accuracy also need further discussion.

1.3 The Scope of the Thesis

In this thesis, a general-purpose FDTD computer program code is developed for modern mobile/personal communications antenna design. Using this code, a new planar monopole antenna, which can operate at dual wide-band, is developed to meet the requirements of existing and potential communication system providing both analogue and digital services. The main work in the thesis can be summarized as follows.

An FDTD program is developed using C++, which is available in MS-DOS, windows and Unix environment, with particular emphasis on the modeling of the source region including feed forms and source forms. The effects of the cell size and the distance between the absorbing boundary and the antenna on the accuracy are explored. The code is validated by comparison of its results with those obtained from NEC-2.

Using the code, a new dual band antenna is developed [23] for modern commu-

nications. The antenna performance is improved [24] by modifying the structures and geometrical dimensions to achieve required frequency band operation. This is a demonstration of using the code to design new types of antennas. It will be showed that a small modification of structure or changes in other physical parameters can produce great difference in antenna performance.

1.4 Organization of the Thesis

This thesis is divided into five chapters. The first chapter is an introductory chapter. including the statement of the problem, literature review and main contents to be discussed in the thesis.

In Chapter 2, the basic theory for FDTD is introduced. The emphasis is put on the source form and the feeding form of the antenna. Main antenna parameters such as input impedance, power, efficiency and far fields in FDTD algorithm are presented.

Chapter 3 introduces the FDTD code developed for antenna design as well as the effects of cell size and the Mur distance on the accuracy of the results. By calculating the input impedance and radiation patterns of various typical antennas and comparing the result with those from NEC-2, the FDTD code is validated.

In Chapter 4, the FDTD code is applied to practical antenna design. A new planar monopole antenna operating at dual wide-band is developed to meet the requirement of existing DCS/GSM communication system.

Chapter 5 gives the conclusion and some recommendations for future work.

Chapter 2

FDTD Approach

2.1 Introduction

The foundation of the FDTD (finite difference time domain) electromagnetic field analysis is Yee's algorithm [16], which was published in 1966. Yee used electric and magnetic field grids, which were offset spatially and temporally from each other, to obtain update equations that yield the present fields in terms of the past fields throughout the computational domain. Calculations of the electric field E and magnetic field H as governed by Maxwell's equations are marched forward in time in a leapfrog fashion. Although many alternative approaches have been proposed since his publication, Yee's algorithm is still the most elegant and simplest way of discretizing Maxwell's equations. There are many applications of FDTD method such as electromagnetic scattering, radiation problems, and antenna analysis. In this chapter, Yee's

algorithm is outlined with much emphasis on its related issues in antenna design.

2.2 Maxwell's Equations

All macroscopic electromagnetic phenomena are governed by Maxwell's equations. Both the differential and the integral forms of Maxwell's equations can be used to derive FDTD update equations. Here, the differential form is employed to demonstrate the basic principle of FDTD. The integral form will be used later to include thin wire structures in antenna problems.

Using the International System of Units (SI units), Maxwell's equations are given by

$$\nabla \times \vec{E} = -\frac{\partial \vec{B}}{\partial t} \qquad \text{Faraday's Law} \tag{2.1}$$

$$\nabla \times \vec{H} = \vec{J} + \frac{\partial \vec{D}}{\partial t} \qquad \text{Ampere's Law} \tag{2.2}$$

$$\nabla \cdot \vec{D} = \rho \qquad \text{Gauss's Law for the electric field} \tag{2.3}$$

$$\nabla \cdot \vec{B} = 0 \qquad \text{Gauss's Law for the magnetic field} \tag{2.4}$$

In linear isotropic medium, the constitutive relation is

$$\vec{D} = \epsilon \vec{E} \tag{2.5}$$

$$\vec{B} = \mu \vec{H} \tag{2.6}$$

$$\vec{J} = \sigma \vec{E} \tag{2.7}$$

The above equations are all the information needed to completely specify the field

behavior. It should be noted that for simplicity the equations are given for linear isotropic materials, which is usually met in antenna design.

The starting point for the FDTD formulation is the curl equations (2.1) and (2.2), which can be recast into the following form:

$$\frac{\partial \vec{E}}{\partial t} = \frac{1}{\epsilon}\nabla \times \vec{H} - \frac{\sigma}{\epsilon}\vec{E} \tag{2.8}$$

$$\frac{\partial \vec{H}}{\partial t} = -\frac{1}{\mu}\nabla \times \vec{E} \tag{2.9}$$

The other two divergence equations (2.3) and (2.4) are in fact redundant as they are contained within the curl equations and the initial boundary conditions. However, they can be used as a test on the predicted field response so that after forming $\vec{D} = \epsilon\vec{E}$ and $\vec{B} = \mu\vec{H}$ from the predicted fields, the resulting $\vec{D}$ and $\vec{B}$ should satisfy the divergence equations.

In order to simplify the formulas and make the programming job easier, one must further decompose the vector Maxwell's curl equations (2.8) and (2.9) into their component scalar parts, obtaining the following scalar equations in the Cartesian coordinate system:

$$\frac{\partial E_x}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} - \sigma E_x\right) \tag{2.10}$$

$$\frac{\partial E_y}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} - \sigma E_y\right) \tag{2.11}$$

$$\frac{\partial E_z}{\partial t} = \frac{1}{\epsilon}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma E_z\right) \tag{2.12}$$

$$\frac{\partial H_x}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_y}{\partial z} - \frac{\partial H_z}{\partial y}\right) \tag{2.13}$$

$$\frac{\partial H_y}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_z}{\partial x} - \frac{\partial H_x}{\partial z}\right) \tag{2.14}$$

$$\frac{\partial H_z}{\partial t} = \frac{1}{\mu}\left(\frac{\partial E_x}{\partial y} - \frac{\partial H_y}{\partial x}\right) \tag{2.15}$$

The above six equations form the basis of the FDTD approach. The Cartesian coordinate is used here since it is convenient and therefore most frequently used in antenna design. For the cylindrical and spherical coordinate systems, the formulation is similar. One can refer to [22] for details.

2.3 Discretization of Maxwell Equations

In discretization of the Maxwell's equations, finite differencing replaces derivatives with differences as follows:

$$\begin{aligned}\frac{\partial f}{\partial t} &= \lim_{\triangle t \to 0} \frac{\triangle f}{\triangle t} \\ &= \lim_{\triangle t \to 0} \frac{f(x, t + \frac{\triangle t}{2}) - f(x, t - \frac{\triangle t}{2})}{\triangle t} \\ &\approx \frac{f(x, t + \frac{\triangle t}{2}) - f(x, t - \frac{\triangle t}{2})}{\triangle t}\end{aligned} \tag{2.16}$$

$$\begin{aligned}\frac{\partial f}{\partial x} &= \lim_{\triangle x \to 0} \frac{\triangle f}{\triangle x} \\ &= \lim_{\triangle x \to 0} \frac{f(x + \frac{\triangle x}{2}, t) - f(x - \frac{\triangle x}{2}, t)}{\triangle x} \\ &\approx \frac{f(x + \frac{\triangle x}{2}, t) - f(x - \frac{\triangle x}{2}, t)}{\triangle x}\end{aligned} \tag{2.17}$$

In the above approximation $\triangle t$ and $\triangle x$ are finite rather than infinitesimal. Thereafter, calculus becomes algebra. An explicit central difference scheme is used in the above equations (2.16) and (2.17) to achieve the second-order accuracy.

Following Yee's notation [16], we quantize space by letting $x = i\triangle x$, $y = j\triangle y$, $z = k\triangle z$, and time by letting $t = n\triangle t$. Then the uniform cells in the problem can be defined by space index (i, j, k) and time index n. For example, $E_z^{n_0}(i_0, j_0, k_0)$ represents the z component of the electric field at time $t = n_0\triangle t$ and at spatial location $x = i_0\triangle x$, $y = j_0\triangle y$, $z = k_0\triangle z$. It should be noted that the field components have different offset as shown in Figure 2.1.

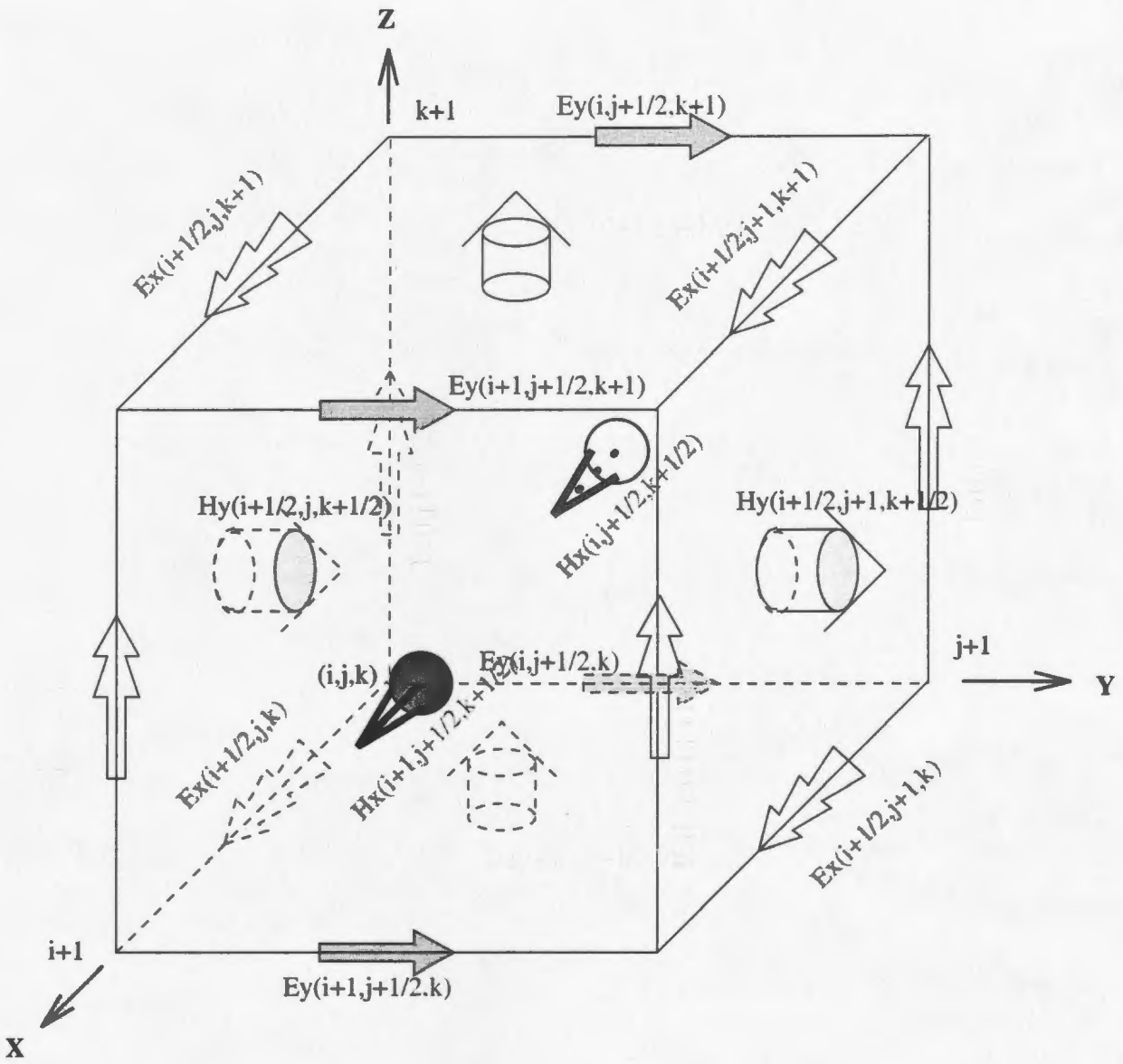


Figure 2.1: Positions of the field components about a unit cell of the FDTD grid

One can replace the time derivatives in equation (2.10) — (2.15) with differences. The time center for the first three equations (2.10) — (2.12) is set at $(n-\frac{1}{2})\triangle t$ while the time center for the second three equations (2.13) — (2.15) is set at $n\triangle t$. Finally, one has

$$\frac{E_x^n - E_x^{n-1}}{\triangle t} = \frac{1}{\epsilon}\left(\frac{\triangle H_z^{n-1/2}}{\triangle y} - \frac{\triangle H_y^{n-1/2}}{\triangle z} - \sigma E_x^{n-1/2}\right) \tag{2.18}$$

$$\frac{E_y^n - E_y^{n-1}}{\triangle t} = \frac{1}{\epsilon}\left(\frac{\triangle H_x^{n-1/2}}{\triangle z} - \frac{\triangle H_z^{n-1/2}}{\triangle x} - \sigma E_y^{n-1/2}\right) \tag{2.19}$$

$$\frac{E_z^n - E_z^{n-1}}{\triangle t} = \frac{1}{\epsilon}\left(\frac{\triangle H_y^{n-1/2}}{\triangle x} - \frac{\triangle H_x^{n-1/2}}{\triangle y} - \sigma E_z^{n-1/2}\right) \tag{2.20}$$

$$\frac{H_x^{n+1/2} - H_x^{n-1/2}}{\triangle t} = \frac{1}{\mu}\left(\frac{\triangle E_y^n}{\triangle z} - \frac{\triangle E_z^n}{\triangle y}\right) \tag{2.21}$$

$$\frac{H_y^{n+1/2} - H_y^{n-1/2}}{\triangle t} = \frac{1}{\mu}\left(\frac{\triangle E_z^n}{\triangle x} - \frac{\triangle E_x^n}{\triangle z}\right) \tag{2.22}$$

$$\frac{H_z^{n+1/2} - H_z^{n-1/2}}{\triangle t} = \frac{1}{\mu}\left(\frac{\triangle E_x^n}{\triangle y} - \frac{\triangle E_y^n}{\triangle x}\right) \tag{2.23}$$

It is observed that in the above three equations (2.18) — (2.20) each E filed component consists of three parts: E^n, E^{n-1}, and $E^{n-\frac{1}{2}}$. The procedure involves predicting E^n from the values of E^{n-1} and $H^{n-\frac{1}{2}}$ only. Accordingly, $E^{n-\frac{1}{2}}$ should be replaced. In some publications [3], the mean value of E^n and E^{n-1} is used, while in other publications [25], $E^{n-\frac{1}{2}}$ is just replaced by E^{n-1}. Here $E^{n-\frac{1}{2}}$ is replaced by E^n. By such a replacement, we have the tangential electrical component $E_T^n = 0$ in the limit as σ goes to infinity. By replacing the space-derivatives of the equations (2.18) - (2.23) with difference, and after some simplification, the previous set of equations

become as follows:

$$E_x^n(i+\frac{1}{2},j,k) = \left[\frac{1}{\epsilon(i+\frac{1}{2},j,k)+\sigma(i+\frac{1}{2},j,k)\triangle t}\right] E_x^{n-1}(i+\frac{1}{2},j,k) + \frac{\triangle t}{\epsilon(i+\frac{1}{2},j,k)+\sigma(i+\frac{1}{2},j,k)\triangle t} \cdot \left[\frac{H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k) - H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j-\frac{1}{2},k)}{\triangle y} - \frac{H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) - H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k-\frac{1}{2})}{\triangle z}\right] \tag{2.24}$$

$$E_y^n(i,j+\frac{1}{2},k) = \left[\frac{1}{\epsilon(i,j+\frac{1}{2},k)+\sigma(i,j+\frac{1}{2},k)\triangle t}\right] E_y^{n-1}(i,j+\frac{1}{2},k) + \frac{\triangle t}{\epsilon(i,j+\frac{1}{2},k)+\sigma(i,j+\frac{1}{2},k)\triangle t} \cdot \left[\frac{H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2}) - H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k-\frac{1}{2})}{\triangle z} - \frac{H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k) - H_z^{n-\frac{1}{2}}(i-\frac{1}{2},j+\frac{1}{2},k)}{\triangle x}\right] \tag{2.25}$$

$$E_z^n(i,j,k+\frac{1}{2}) = \left[\frac{1}{\epsilon(i,j,k+\frac{1}{2})+\sigma(i,j,k+\frac{1}{2})\triangle t}\right] E_z^{n-1}(i,j,k+\frac{1}{2}) + \frac{\triangle t}{\epsilon(i,j,k+\frac{1}{2})+\sigma(i,j,k+\frac{1}{2})\triangle t} \cdot \left[\frac{H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) - H_y^{n-\frac{1}{2}}(i-\frac{1}{2},j,k+\frac{1}{2})}{\triangle x} - \frac{H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2}) - H_x^{n-\frac{1}{2}}(i,j-\frac{1}{2},k+\frac{1}{2})}{\triangle y}\right] \tag{2.26}$$

$$H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2}) = H_x^{n-\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2}) + \frac{\triangle t}{\mu(i,j+\frac{1}{2},k+\frac{1}{2})} \cdot \left[\frac{E_y^n(i,j+\frac{1}{2},k+1) - E_y^n(i,j+\frac{1}{2},k)}{\triangle z} - \frac{E_z^n(i,j+1,k+\frac{1}{2}) - E_z^n(i,j,k+\frac{1}{2})}{\triangle y} \right] \tag{2.27}$$

$$H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) = H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) + \frac{\triangle t}{\mu(i+\frac{1}{2},j,k+\frac{1}{2})} \cdot \left[\frac{E_z^n(i+1,j,k+\frac{1}{2}) - E_z^n(i,j,k+\frac{1}{2})}{\triangle x} - \frac{E_x^n(i+\frac{1}{2},j,k+1) - E_x^n(i+\frac{1}{2},j,k)}{\triangle z} \right] \tag{2.28}$$

$$H_z^{n+\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k) = H_z^{n-\frac{1}{2}}(i+\frac{1}{2},j+\frac{1}{2},k) + \frac{\triangle t}{\mu(i+\frac{1}{2},j+\frac{1}{2},k)} \cdot \left[\frac{E_x^n(i+\frac{1}{2},j+1,k) - E_x^n(i+\frac{1}{2},j,k)}{\triangle y} - \frac{E_y^n(i+1,j+\frac{1}{2},k) - E_y^n(i,j+\frac{1}{2},k)}{\triangle x} \right] \tag{2.29}$$

In the above formulation, there is a half cell offset in both space and time indices. One should note that both space and time indices must be integers in a computer code.

2.4 Radiation Boundary Condition

When the outer boundary of the domain concerned recedes to infinity, the domain is called "unbounded" or "open". A condition must also be specified at the outer

boundary in order to obtain a unique solution for the problem and such a condition is referred to as the radiation condition.

2.4.1 The Sommerfeld Radiation Condition

Assuming that all sources and objects are immersed in free space and located within a finite distance $r'(x', y', z')$ from the origin of a coordinate system, the electric and magnetic fields at location $r(x, y, z)$ are required to satisfy the following equations:

$$\lim_{r\to\infty}\left[r\nabla\times\begin{pmatrix}\vec{E}\\ \vec{H}\end{pmatrix}+jk_0\vec{r}\times\begin{pmatrix}\vec{E}\\ \vec{H}\end{pmatrix}\right]=0 \tag{2.30}$$

where r is the distance between the observing point and the origin, and

$r=\sqrt{(x-x')^2+(y-y')^2+(z-z')^2}$.

Equation (2.30) is usually referred to as the *Sommerfield radiation condition* for the general three-dimensional fields.

However, *Sommerfield radiation condition* cannot be used in our computer program, since no computer can store an unlimited amount of data. The field computation domain must be limited in size, but the computation domain must be large enough to enclose the structure of interest so that a suitable absorbing boundary condition (ABC) on the outer perimeter of the domain can be used to simulate its extension to infinity.

2.4.2 Absorbing Boundary Condition

The quest for a good ABC that produces negligible reflections has been a very active area of FDTD research. Most of the popular ABCs can be grouped into those derived from differential equations [18, 17], or those that employ a material absorber [26]. Among a large number of ABCs, the Mur absorbing boundary condition is both very simple and accurate for engineering applications. In this thesis, the Mur absorbing boundary condition, or more particularly the first and second order Mur conditions are used to estimate the field on the boundary. The first order Mur condition looks back one step in time and into the space one cell location and the second order Mur condition looks back two steps in time and inward two cell locations. The effects of the distance between the Mur condition and the antenna under consideration on the accuracy of the numerical results will be discussed in the next chapter.

2.5 The Cell Size and Time Step for Stability

The choice of cell size is critical in applying FDTD. It must be small enough to permit accurate results at the highest frequency of interest, and yet be large enough to keep resource requirements manageable. Due to the approximation inherent in FDTD, fields of different frequencies will propagate at slightly different speeds through the cells. This difference in propagation speed also depends on the direction of propagation relative to the cells. This phenomena is called numerical dispersion. In order to

get accurate and stable results, the numerical dispersion error must be reduced to an acceptable level, which can be readily accomplished by reducing the cell size.

Another cell size consideration is that the important characteristics of the problem must be accurately modeled. Normally this will be met automatically by making the cells smaller than $\lambda/20$ or so, unless some special geometry features smaller than this factors in determining the response of interest. An example is thin wire antennas, in which a small change in the wire thickness will affect the antenna impedance. Good results in these situations may require extremely small cells or alternative measures such as sub-cell modeling, which will be discussed later.

The time step is generally determined by the Courant condition[25]. For three-dimensional rectangular grid, the Courant condition is

$$v\triangle t \leq \frac{1}{\sqrt{\frac{1}{(\triangle x)^2}+\frac{1}{(\triangle y)^2}+\frac{1}{(\triangle z)^2}}} \tag{2.31}$$

where v is the wave velocity in the media, $\triangle t$ is the time for one time step and $\triangle x, \triangle y, \triangle z$ are the cell size.

This is a basic requirement for stability. People usually use this equation for time step calculation if a pulse is used as source form. However, when the sine wave is used as the source form of the antenna, one may have to use smaller $\triangle t$ to get more accurate data, because the amplitude error and the phase error for steady-state response are decided directly by $\triangle t$, which will be discussed later.

2.6 Antenna Parameters in FDTD

There are lots of parameters for antennas, and it is not necessary to give the expression for each parameter. Here we discuss several parameters that are most important and also the basis of other parameters.

2.6.1 Input Impedance, Power and Efficiency

The electric and magnetic fields in the source region (generally one cell) are used to determine the input impedance of the antenna. The detailed expression for input impedance can be different for various feed forms, which will be discussed later. Here the feed of the antenna is assumed to be a thin gap voltage source. In time domain the impressed excitation voltage is defined by

$$v_s(n\triangle t) = -\vec{E}\cdot d\vec{l} = -E_z^n(i,j,k+\frac{1}{2})\triangle z \tag{2.32}$$

and the resultant current flowing in the source region is found by calculating

$$\begin{aligned} i_s\left[(n+\frac{1}{2})\triangle t\right] &= \int \vec{H}\cdot d\vec{l} \\ &= [H_x^{n+\frac{1}{2}}(i,j-\frac{1}{2},k+\frac{1}{2}) - H_x^{n+\frac{1}{2}}(i,j+\frac{1}{2},k+\frac{1}{2})]\triangle x \\ &- [H_y^{n+\frac{1}{2}}(i-\frac{1}{2},j,k+\frac{1}{2}) - H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2})]\triangle y \end{aligned} \tag{2.33}$$

where n is the time index.

The input impedance of the antenna is determined from the ratio of the Fourier transform of the voltage wave and that of the input current wave, i.e.,

$$Z(\omega) = Z(k\triangle f) = \frac{DFT[v_s(n)]}{DFT[i_s(n)]} = \frac{V_s(k\triangle f)}{I_s(k\triangle f)} = \frac{V_s(\omega)}{I_s(\omega)} \tag{2.34}$$

where $\triangle f = \frac{1}{N\triangle t}$, and N is the total number of time steps needed for the current to attenuate to a very small value. It should be noted that the time difference $\triangle t/2$ between voltage wave and current wave is ignored since its effect is very small. Either the discrete Fourier transform (DFT) or the fast Fourier transform (FFT) can be used for the transformation process, and the latter will be used in this thesis.

The calculation of the input power is given by

$$P_{in}(\omega) = Re\left[V_s(\omega)I_s^*(\omega)\right] \tag{2.35}$$

The dissipated power can be computed as follows. Consider that an FDTD electric field component $E_z(i, j, k+\frac{1}{2})$ is in a region with conductivity σ. If we assume that the electric field is uniform within a single FDTD cell, then the equivalent steady-state power dissipated in this region is given by

$$\begin{aligned} P_{diss} &= \iiint \sigma |E_z(\omega)|^2 dv \\ &= \sigma |E_z(\omega)|^2 \triangle x \triangle y \triangle z = \frac{\sigma \triangle x \triangle y}{\triangle z} |E_z(\omega) \triangle z|^2 \\ &= \frac{|V_z(\omega)|^2}{R} \end{aligned} \tag{2.36}$$

where R is the resistance, and $R = \frac{\sigma \triangle x \triangle y}{\triangle z}$. The antenna efficiency is determined from the input P_{in} and dissipated power P_{diss} as follows

$$\eta_{eff} = \frac{P_{in} - P_{diss}}{P_{in}} \tag{2.37}$$

2.6.2 Far Fields

There are two different ways to obtain the far fields from the time-domain fields in the defined space. One is sine steady-state response method [25]. Another is the so-called FFT method [27, 28]. Here the first method is used because it is more efficient when the radiation patterns of antenna are not very sensitive to frequency change where only two or three frequencies are required.

Since the source is the form of sine wave, one can find the amplitude and phase of the fields when steady-state condition is reached. After obtaining the tangential electric and magnetic currents in terms of surface fields on a closed surface surrounding the antenna, one can use the equivalent electric and magnetic currents to compute the corresponding radiated fields in the far zone.

It is assumed that $\vec{J}_s = \hat{n} \times \vec{H}$ and $\vec{M}_s = -\hat{n} \times \vec{E}$, where $\hat{n}$ is the unit normal vector of the surface surrounding the antenna while $\vec{E}$ and $\vec{H}$ are the fields on the surface. The retarded potentials $\vec{F}$ and $\vec{A}$ can be defined in terms of the magnetic source and electric source respectively [7]. For a homogeneous isotropic medium, the relations are

$$\vec{A} = \mu \int_{s'} \frac{\vec{J}_s e^{-ikr}}{4\pi r} ds' \tag{2.38}$$

$$\vec{F} = \epsilon \int_{s'} \frac{\vec{M}_s e^{-ikr}}{4\pi r} ds' \tag{2.39}$$

The fields in terms of the potentials are [7]

$$\vec{E} = -j\omega\vec{A} - \frac{j\omega}{k^2}\nabla(\nabla\cdot\vec{A}) - \frac{1}{\epsilon}\nabla\times\vec{F} \tag{2.40}$$

$$\vec{H} = -j\omega\vec{F} - \frac{j\omega}{k^2}\nabla(\nabla\cdot\vec{F}) - \frac{1}{\epsilon}\nabla\times\vec{A} \tag{2.41}$$

For simplicity, $\vec{A}$ and $\vec{F}$ can be expressed as

$$\vec{A} = \mu\frac{e^{-jkr}}{4\pi r}\vec{N} \tag{2.42}$$

$$\vec{F} = \epsilon\frac{e^{-jkr}}{4\pi r}\vec{L} \tag{2.43}$$

where

$$\vec{N} = \int_{s'}\vec{J}_s e^{jkr'\cos\psi} \tag{2.44}$$

$$\vec{L} = \int_{s'}\vec{M}_s e^{jkr'\cos\psi} \tag{2.45}$$

where

$$r'\cos\psi = z'\cos\theta + (x'\cos\phi + y'\sin\phi)\sin\theta$$

and k is the wave number and (x', y', z') is the source point.

If electric and magnetic field components are now written in the usual way in terms of two vector potentials, the only components in the far fields not decreasing faster than $1/r$ are

$$E_\theta = \eta H_\phi = -j\frac{e^{-jkr}}{2\lambda r}(\eta N_\theta + L_\phi) \tag{2.46}$$

$$E_\phi = -\eta H_\theta = j\frac{e^{-jkr}}{2\lambda r}(-\eta N_\phi + L_\phi) \tag{2.47}$$

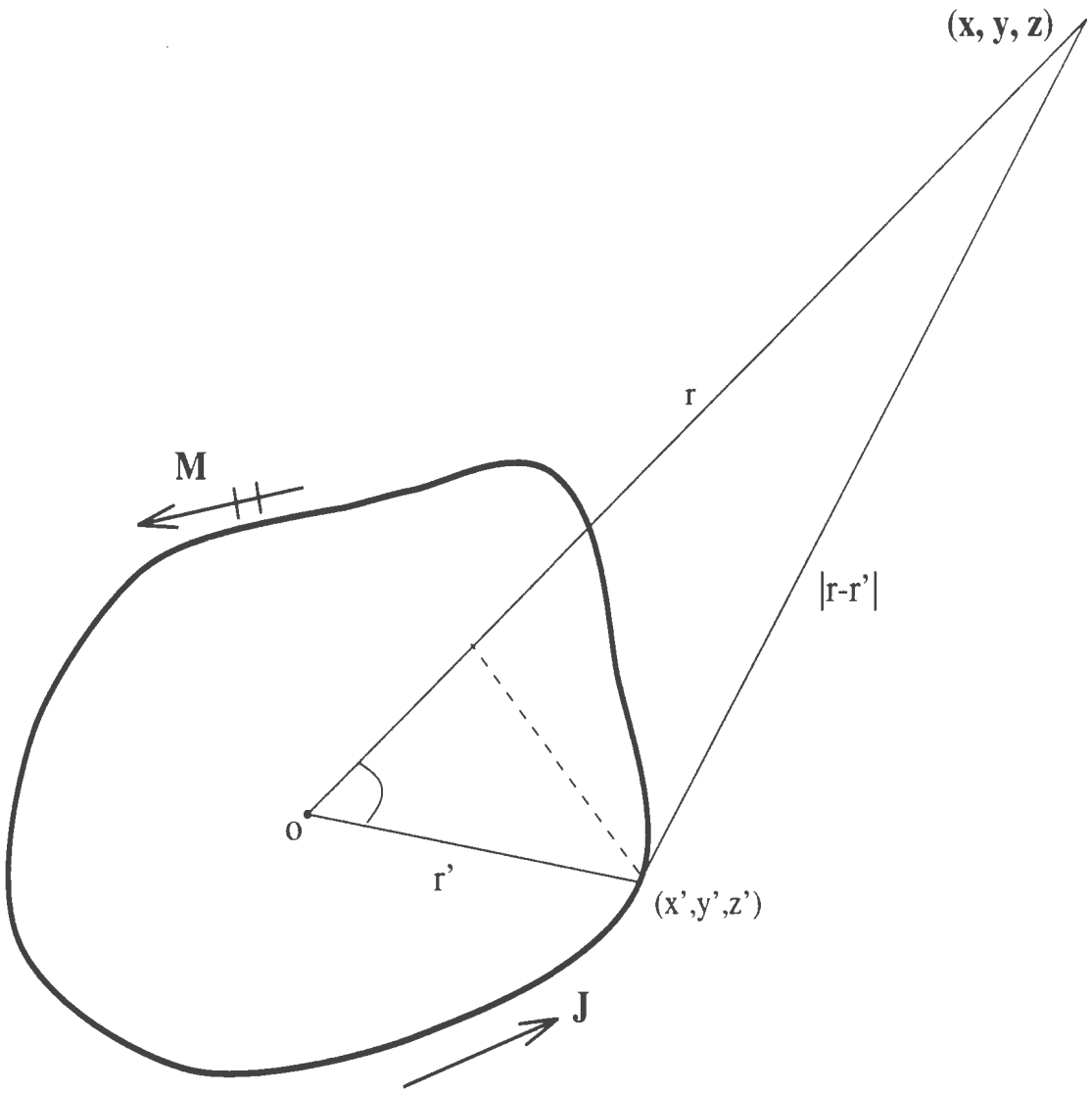


Figure 2.2: Find the far field from equivalent currents

In this thesis, $\vec{N}$ and $\vec{L}$ are expressed in terms of the Cartesian coordinate component N_x, N_y, N_z since $\vec{J}$ and $\vec{M}$ are conveniently expressed in terms of fields in

Cartesian coordinates

$$N_\theta = (N_x \cos\phi + N_y \sin\phi)\cos\phi - N_z \sin\theta \tag{2.48}$$

$$N_\phi = -N_x \sin\phi + N_y \cos\phi \tag{2.49}$$

2.7 Special Consideration

The basic principle of FDTD for various electromagnetic problems is the same, but there are some differences for the application of FDTD in various aspects. In the problem of antenna radiation, special considerations for thin wire structures and the feed forms should be taken into account.

2.7.1 Thin Wire Structure

In antenna design, a common geometry to be modeled is a thin wire of finite radius. Often a wire is much smaller in radius than its length, and approximation of the wires as being without a radius may yield poor results, as both antenna impedance and coupling are sensitive to the wire radius. Also it is often desirable to avoid sizing the FDTD cells small enough to accurately model the thin wire because the computational cost of the FDTD technique is decided directly by the number of cells. An alternative solution is a computational more efficient concept of a sub cell approach. A new set of equations that take the near-field characteristics of the problem into account is derived with the aid of the contour integral form of Maxwell's

equations. The new equations allow for a much larger spatial grid size to be used. The standard FDTD field equations are used for majority of the cells in the grid while the sub-cells are used near the wire structure. The sub cell approach was first proposed by Umashankar and Taflove [29], and then widely used among other researchers.

The geometry is shown in Figure 2.3. A conducting circular wire of radius r_0 is positioned to align with the center on the $E_z(i, j, k+\frac{1}{2})$ field component. The wire is assumed to have a radius smaller than $0.5\Delta x$, and Δx must be much smaller than the wavelength (less than $\lambda/20$) for FDTD to be applied. therefore, the wire radius also must be much smaller than a wavelength. The circumferential magnetic field component $H_\phi(r)$ and the radial electric field component $E_r(r)$ surrounding the wire are assumed to vary as $\frac{1}{r}$ near the wire, where r is the radial distance form the center of the wire. These fields are represented as follows:

$$H_\phi(r) = \frac{H_\phi(0)}{r} \cdot f1(t) \tag{2.50}$$

$$E_r(r) = \frac{E_r(0)}{r} \cdot f2(t) \tag{2.51}$$

Strictly speaking, such an assumption holds only for the fields around an infinite cylindrical line source, but numerical results have verified that it can approximate the fields near the wire of finite length very well.

With the above assumptions, the spatial dependence of the fields in the vicinity of the wire can be approximated as follows:

$$H_y(r) \approx H_y(i, j, k+\frac{1}{2}) \cdot \frac{\Delta x}{2r} \tag{2.52}$$

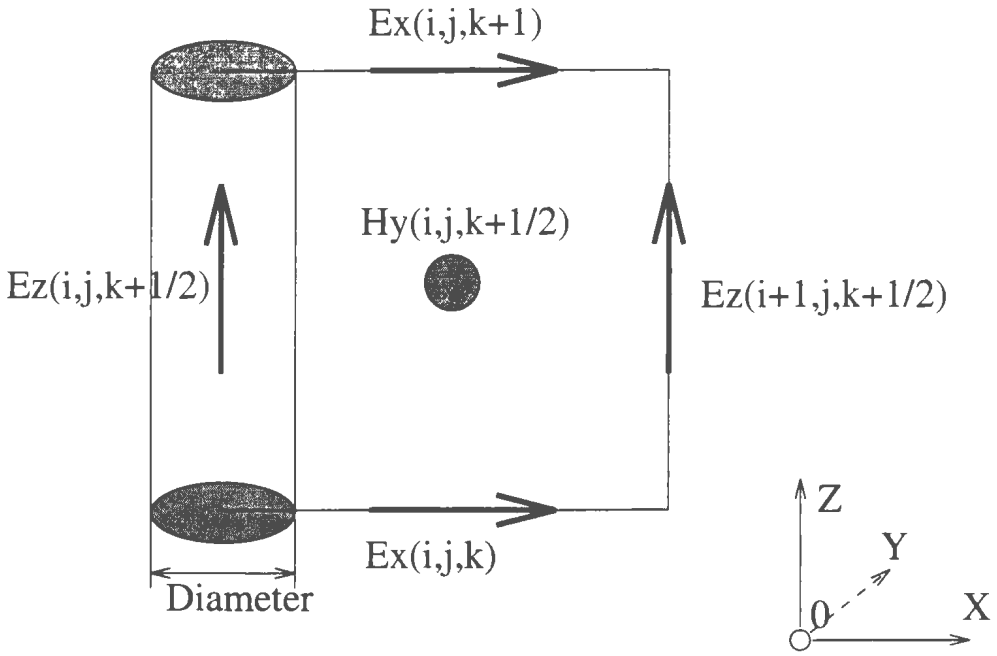


Figure 2.3: The geometry of thin wire in a subcell approach

within the contour, and

$$E_x(r) \approx E_x(i,j,k) \cdot \frac{\triangle x}{2r} \tag{2.53}$$

along the upper and the lower integration contours. We now apply the Maxwell Faraday's law equation to the cell containing the wire.

$$\int \vec{E} \cdot d\vec{l} = -\mu \frac{\partial}{\partial t} \iint \vec{H} \cdot d\vec{s} \tag{2.54}$$

After simplifying the integration, one gets

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) &= H_y^{n-\frac{1}{2}}(i+\frac{1}{2},j,k+\frac{1}{2}) \\ &+ \frac{\triangle t}{\mu \triangle z}\left[E_x^n(i+\frac{1}{2},j,k) - E_x^n(i+\frac{1}{2},j,k+1)\right] \\ &+ \frac{2\triangle t}{\mu \triangle x \ln \frac{\triangle x}{r_0}} E_z^n(i+1,j,k+\frac{1}{2}) \end{aligned} \tag{2.55}$$

For each $E_z(i, j, k+\frac{1}{2})$ component at the center of the thin wire, there are four associated H field components in the surrounding adjunct cells that must be computed at each time step. The electric field values are updated using the usual FDTD equations. The above equation is derived from the cell containing the conducting wire cell. If a thin gap source is to be modeled, then the voltage difference at the terminals of the dipole will give rise to a z-directed electric field in the source region. The modified time stepping expression for the circumferential magnetic field in the source region is given by

$$\begin{aligned} H_y^{n+\frac{1}{2}}(i+\frac{1}{2}, j, k+\frac{1}{2}) &= H_y^{n-\frac{1}{2}}(i+\frac{1}{2}, j, k+\frac{1}{2}) \\ &+ \frac{\triangle t}{\mu \triangle z}\left[E_x^n(i+\frac{1}{2}, j, k) - E_x^n(i+\frac{1}{2}, j, k+1)\right] \\ &+ \frac{2\triangle t}{\mu \triangle x \ln\frac{\triangle x}{r_0}}\left[E_z^n(i+1, j, k+\frac{1}{2}) - E_z^n(i, j, k+\frac{1}{2})\right] \end{aligned} \tag{2.56}$$

Actually, (2.56) is the formula that will be used in our program since it takes into account the situation of both thin wires and source region.

2.7.2 Antenna Feed

Modeling of the antenna feed can be accomplished using different methods: voltage source and current source method. The voltage source method has two different implementation approaches. For the convenience of demonstration, a monopole antenna is studied as an example. One can start with the thin gap approach of voltage source method, which is shown in Figure 2.4.

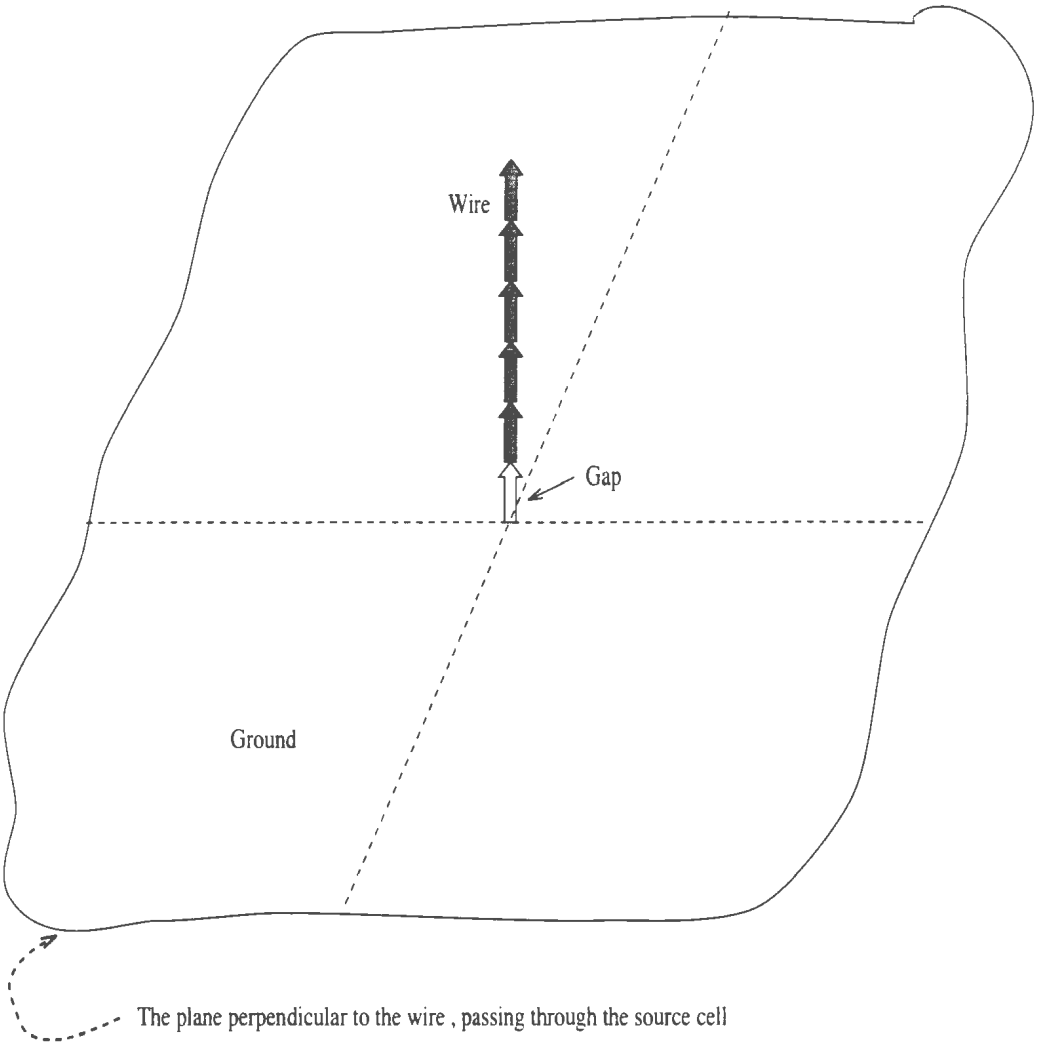


Figure 2.4: The thin gap method of voltage source

The E_z component along the wire axis is zero except at the feeding cell where E_z is excited. The electric field at the feeding cell is expressed as

$$E_z^n(i,j,k+\frac{1}{2}) = -V_s(n\triangle t)/\triangle z \tag{2.57}$$

The resultant current can be obtained by (2.33).

Another approach is shown in Figure 2.5, where the E_z field component along

the wire axis is set to zero, including the one of source region. The four electric field components going radially from the wire axis are given by

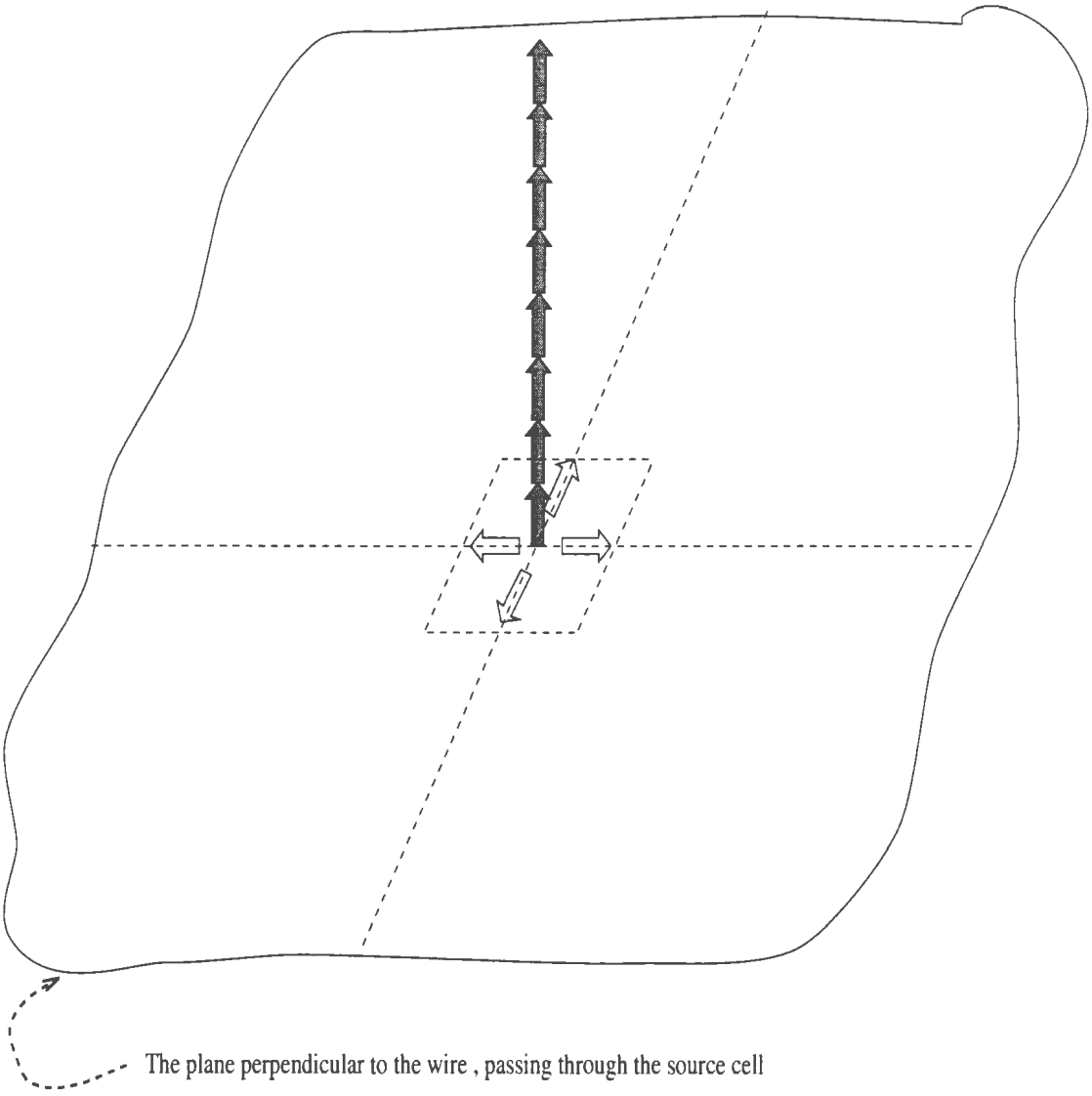


Figure 2.5: The equivalent magnetic current method of voltage source

$$E_x^n(i+\frac{1}{2},j,k) = -E_x^n(i-\frac{1}{2},j,k) = -\frac{V_s(n\triangle t)}{\frac{\triangle x}{2}\ln(\frac{\triangle x}{r_0})} \tag{2.58}$$

$$E_y^n(i,j+\frac{1}{2},k) = -E_y^n(i,j-\frac{1}{2},k) = -\frac{V_s(n\triangle t)}{\frac{\triangle y}{2}\ln(\frac{\triangle y}{r_0})} \tag{2.59}$$

where r_0 the radius of the wire.

For current source, an impressed current source $I_s(n\triangle t)$ can be incorporated directly into the Maxwell's equation (2.2) to give

$$\begin{aligned} E_z^n(i,j,k+\frac{1}{2}) &= E_z^{n-1}(i,j,k+\frac{1}{2}) \\ &+ \left[H_y(i+\frac{1}{2},j,k) - H_y(i-\frac{1}{2},j,k)\right]\cdot\frac{\triangle t}{\epsilon\triangle x} \\ &- \left[H_x(i,j+\frac{1}{2},k) - H_y(i,j-\frac{1}{2},k)\right]\cdot\frac{\triangle t}{\epsilon\triangle y} \\ &- I_s(n\triangle t)\cdot\frac{\triangle t}{\epsilon\triangle x\triangle y} \end{aligned} \tag{2.60}$$

The corresponding voltage can be found by (2.32).

2.7.3 Source Forms

In this thesis, "source" is used to supply energy to the electromagnetic calculations. A great variety of waveforms for antenna source (current source or voltage source) are possible, but experience has led to only two or three waveforms suitable as source forms, which are Gaussian pulse[22], Rayleigh pulse[8] and sine wave[25]. Although modulated Gaussian and Rayleigh pulse, which are used for high operation frequency, have also been found in the literature [30], they will not be discussed here since the

operation frequency to be considered is not very high —generally $1-2GHz$ for mobile communication.

Gaussian Pulse

The most frequently used waveform in published literature is the Gaussian pulse, which is expressed as

$$f(t) = Ae^{-(t-t_0)^2/T^2} \tag{2.61}$$

where T denotes the pulse width, t_0 is the time delay, and A is the amplitude, usually normalized as "1". Its waveform is shown in Figure 2.6. Since the spectrum of the Gaussian pulse can be very wide, if the parameters are chosen properly, the Gaussian pulse is usually used to obtain the input impedance of an antenna, which is usually sensitive to frequency change.

Although an ideal Gaussian pulse extends to infinity in time, the Gaussian pulse for FDTD must be truncated in calculation to improve computing efficiency. t_0 is chosen to enable a smooth "turn on" of the pulse. The effects of the truncation must be considered since the discontinuity in time domain may produce ringing in frequency domain[22]. The pulse width T should be chosen so that its spectrum (FFT of the pulse) has a suitable bandwidth.

For the convenience of programming, equation (2.61) is rewritten in the following form

$$f(n\Delta t) = Ae^{-\alpha(n\Delta t-\beta\Delta t)^2} \tag{2.62}$$

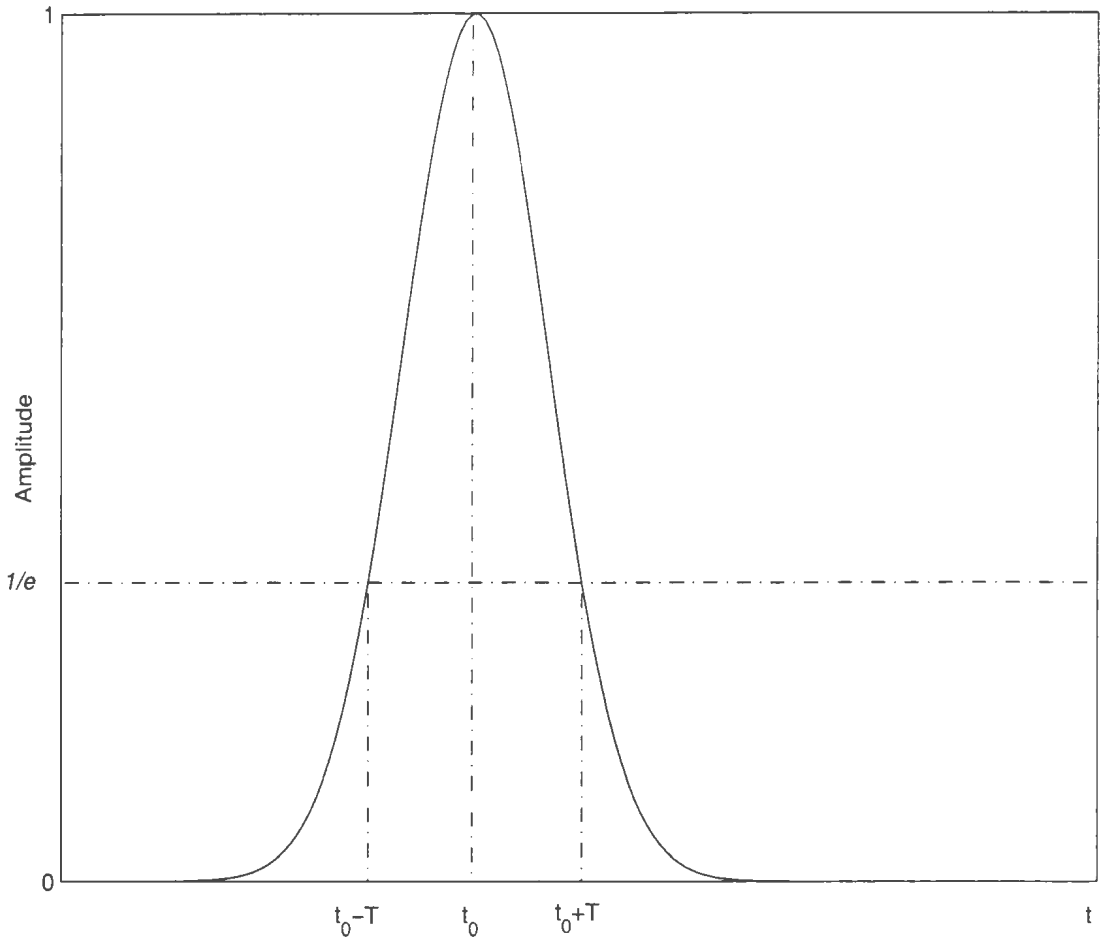


Figure 2.6: The waveform of Gaussian pulse with amplitude of 1

where the time step $\triangle t$, determined by Courant stability condition (2.31), is dependent on the cell size in FDTD. n is the time index. $\beta = t_0/\triangle t$, and $\alpha = 1/T^2$.

In computer programming, the pulse is chosen to exist from $n = 0$ until $n = 2\beta$, approximated as zero outside this range, where n is the time index and β is the number of time steps in the Gaussian pulse from the peak value to the truncation

value. This can be formulated explicitly as

$$f(n\Delta t) = \begin{cases} Ae^{-\alpha(n\Delta t - \beta\Delta t)^2} & \text{if } 0 \leq n \leq 2\beta \\ 0 & \text{otherwise} \end{cases} \tag{2.63}$$

and the corresponding waveform is shown in Figure 2.7.

The pulse amplitude at the truncation is $Ae^{-\alpha(\beta\Delta t)^2}$. Now let

$$\alpha = \frac{er_0}{\beta\Delta t} \tag{2.64}$$

where er_0 is a constant, so that the amplitude of the pulse at truncation is always the same value for different β and Δt. When Δt is fixed, β controls the pulse width and therefore the corresponding spectrum band. It should be chosen according to actual design requirements for antenna operating frequency.

Then we need to determine er_0 so that this truncation does not introduce unwanted high frequency components into spectrum, and yet does not waste computation time on determining values of the source that are essentially zero. Because the accuracy for a single precision calculation is about -120 dB (six significant decimal digits), we choose $er_0 = 16$ so that α equals to $exp(-16)$ or about -140 dB.

To illustrate the effect of this choice of er_0 we consider a situation which will be used later for antenna design. For a three dimensional cubic cell with 0.5cm sides, applying the Courant stability condition (2.31) which is $\Delta t \leq 0.005/(\sqrt{3}c)$ here, one obtains a Δt of $0.00963ns$. Taking the FFT of equation (2.63), one can get the amplitude variation with frequency as plotted in Figure 2.8. From the figure one

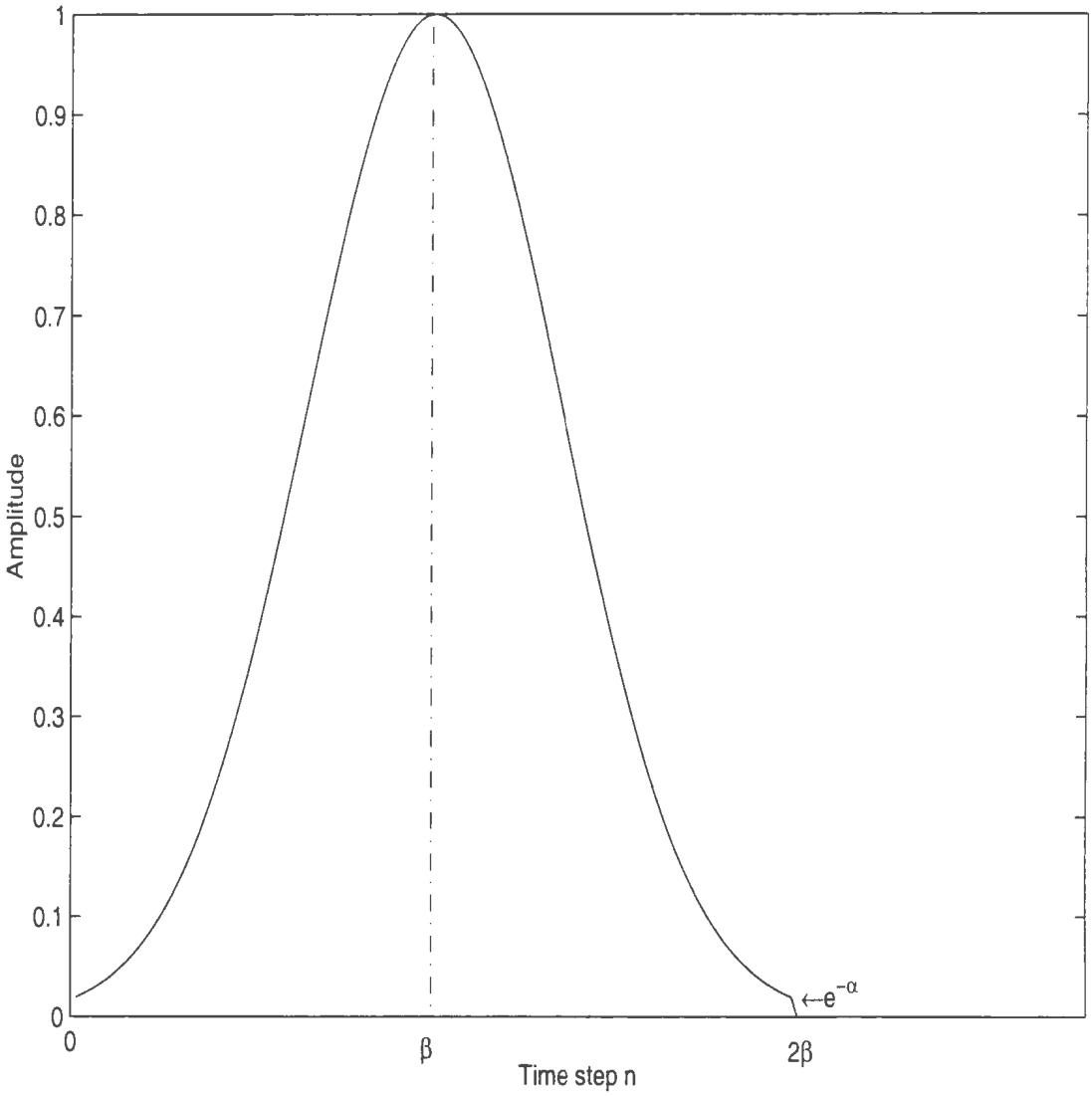


Figure 2.7: The waveform of Gaussian pulse with amplitude of 1 in FDTD

can see increase in α increases the dynamic range. For single precision calculation, $er_0 = 16$ is accurate enough.

Figure 2.9 shows the effect of β on its frequency band. It is clear that the bigger β, the narrower the corresponding frequency band.

When choosing β, the frequency band should not be too narrow or too wide.

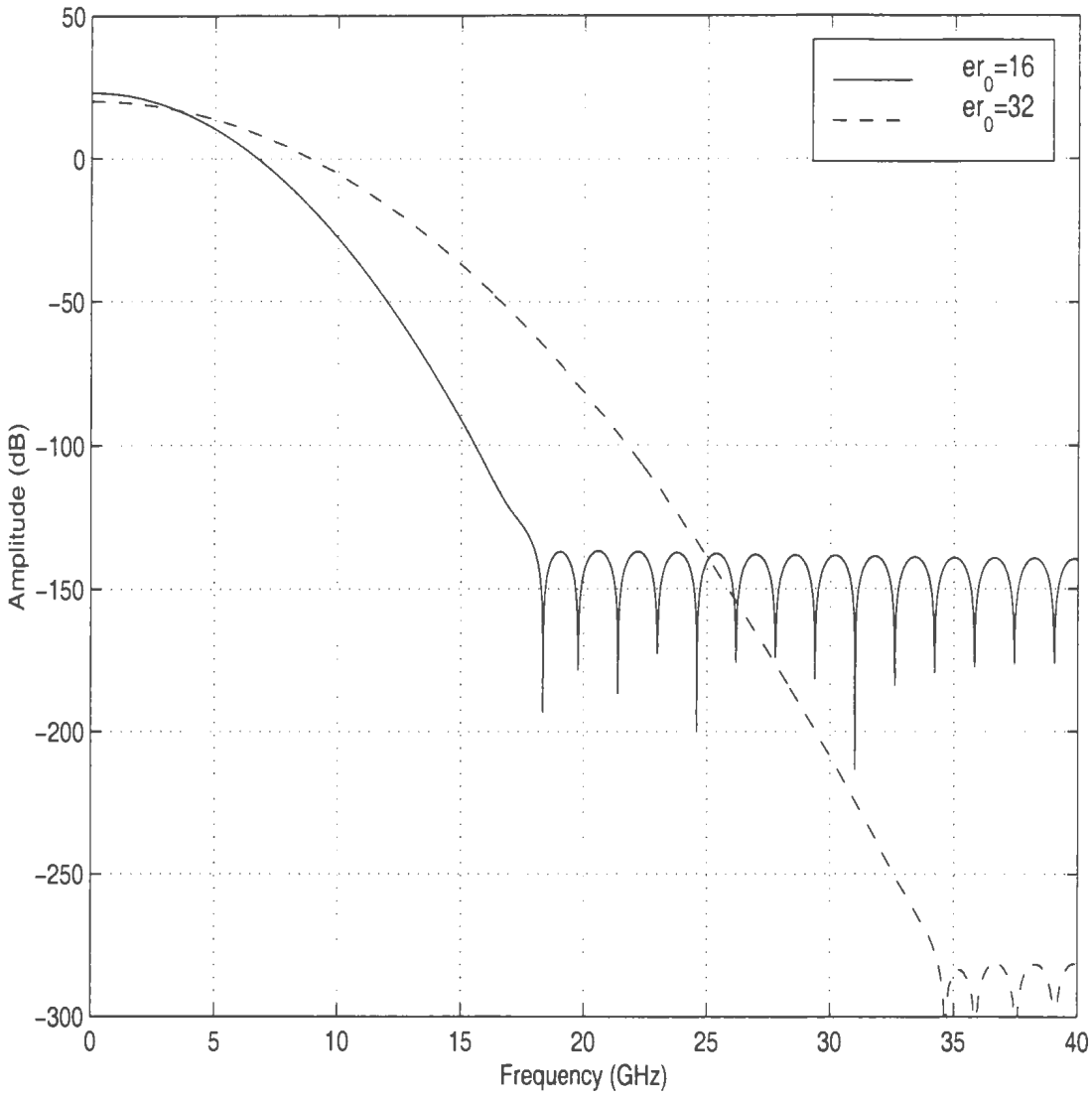


Figure 2.8: The effects of α on high frequencies with $\beta = 32$.

On one hand, the frequency band should be wide enough to contain the operation frequency range of the antenna. On the other hand, the spectrum component outside the operation frequency should be as small as possible since those components can possibly cause numerical noise in the interested operating frequency band due to the numerical dispersion in the calculation.

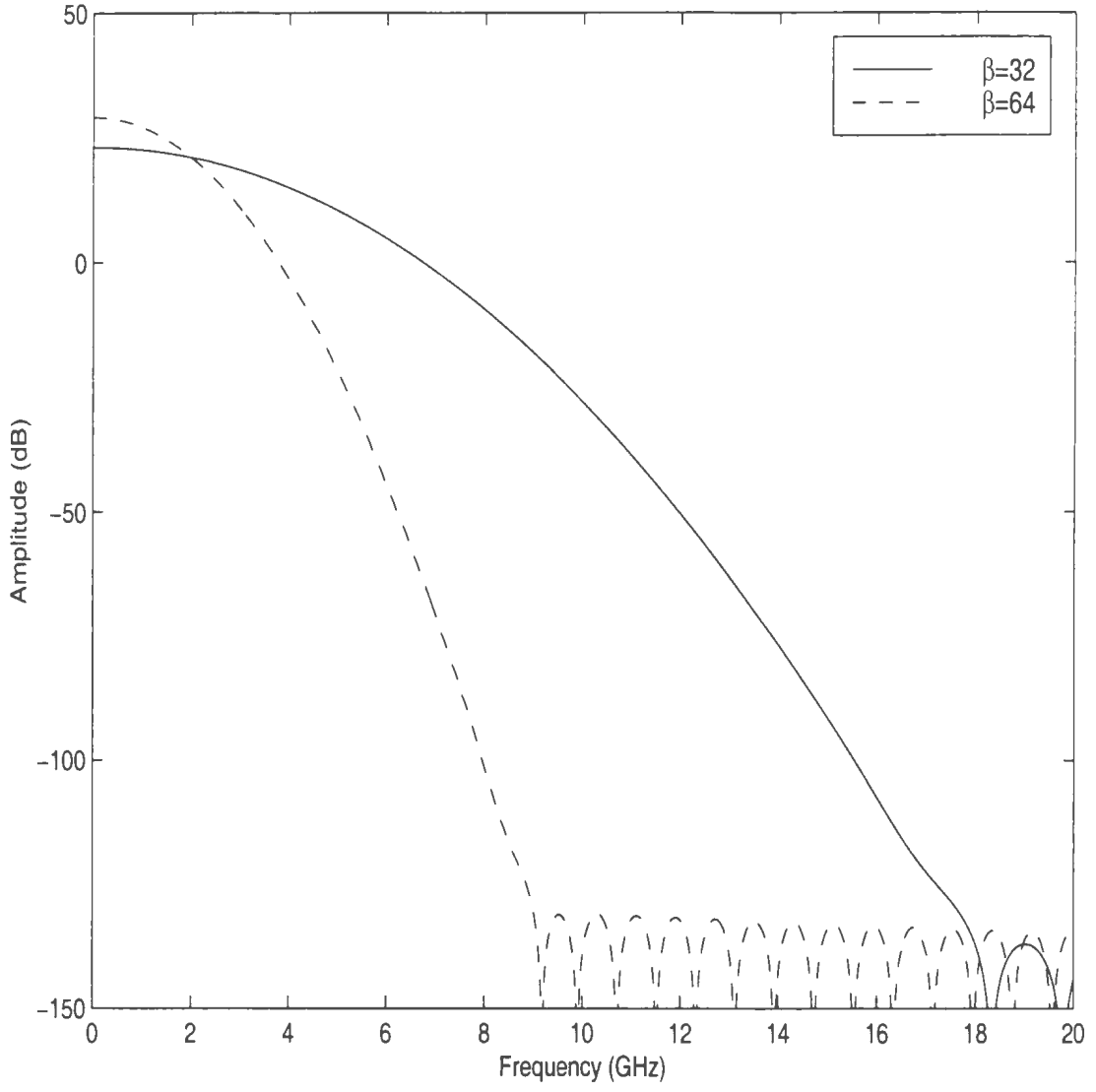


Figure 2.9: The effects of β on frequency band with $er_0 = 16$

Rayleigh Pulse

Rayleigh pulse is the time derivative of the Gaussian pulse. Because of its smooth shaped spectrum (FFT of the Gaussian pulse), it provides information from a little above dc to the desired frequency simply by adjusting the width of the pulse. Its

mathematical form is given as

$$f(t) = -2\frac{(t - t_0)}{T^2} A e^{-(t-t_0)^2/T^2} \tag{2.65}$$

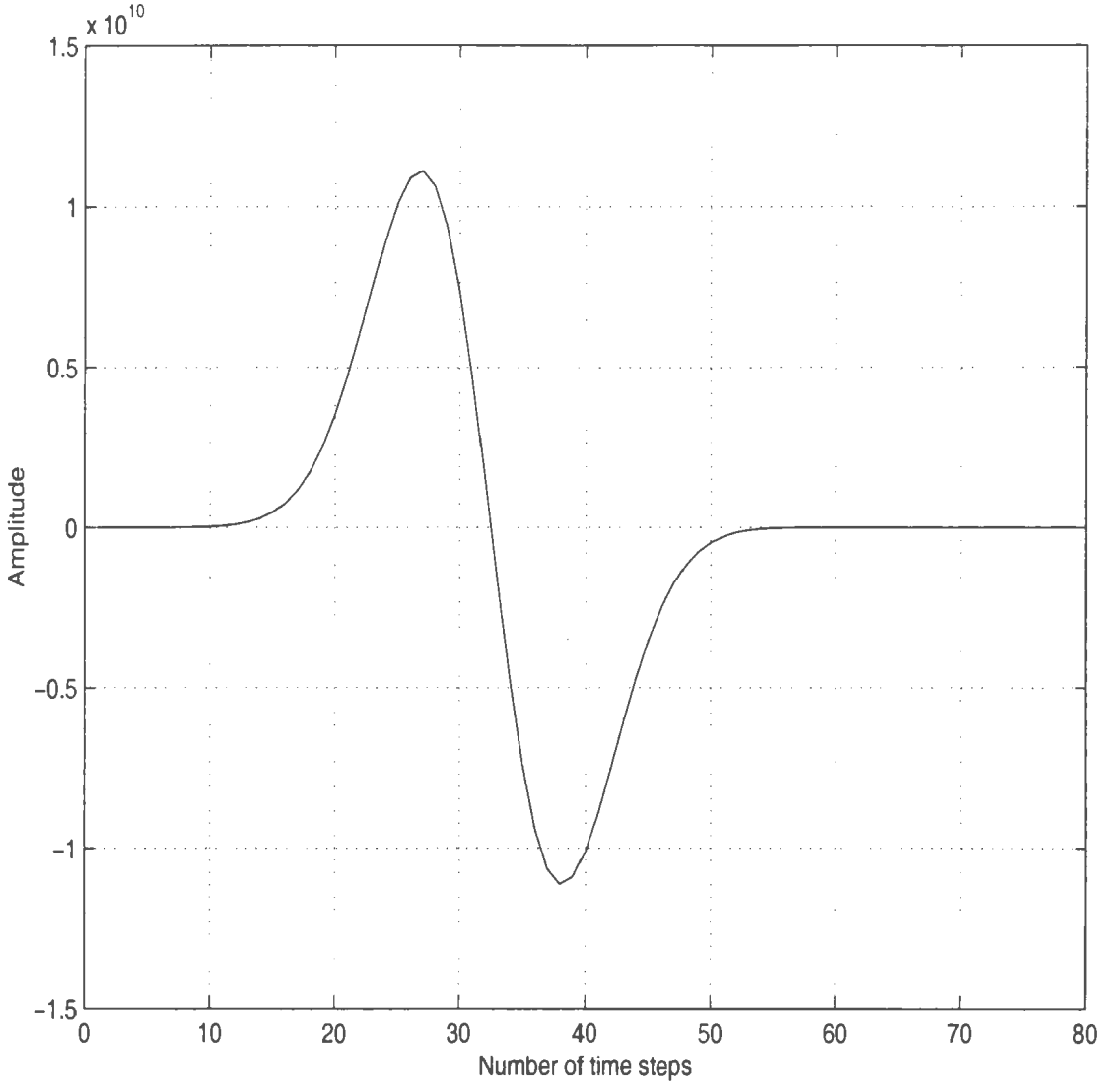


Figure 2.10: The waveform of Rayleigh pulse

Similarly, the formulation in FDTD calculation should be changed as follows

$$f(n\Delta t) = \begin{cases} -2\alpha(n\Delta t - \beta\Delta t)Ae^{-\alpha(n\Delta t-\beta\Delta t)^2} & \text{if } 0 \leq n \leq 2\beta \\ 0 & \text{otherwise} \end{cases} \tag{2.66}$$

where the parameters in the above formula have the same definitions as those for the Gaussian pulse in the last section. Figure 2.10 shows the waveform of the Rayleigh pulse and Figure 2.11 is the corresponding spectrum (FFT) of the Rayleigh pulse with the same parameters as in the Gaussian pulse. It should be noted that Rayleigh pulse has no DC component and the maximum of the spectrum amplitude is not located at the frequency approaching zero. The Gaussian pulse and the Rayleigh pulse have similar spectrum characteristics, and it seems that the choice of using either form will produce the same results. But the similarity may not hold true for all cases as will be shown in a later chapter.

Sine Wave

Sine wave has a fixed frequency. It is efficient to use sine wave for calculating the radiation patterns, which are not very sensitive to frequency change. At $t = 0$, a source of frequency f is assumed to be turned on. The radiation of this source is simulated by solving the finite-difference update equations on the grids of cells, within the computational domain. Time-stepping is continued until the sinusoidal steady state is achieved at each cell. The field envelope, or maximum absolute value, during the final cycle of time-stepping is taken as the magnitude of the phasor of the steady-state field at each cell. In the meantime, the phase of the field at each cell can also

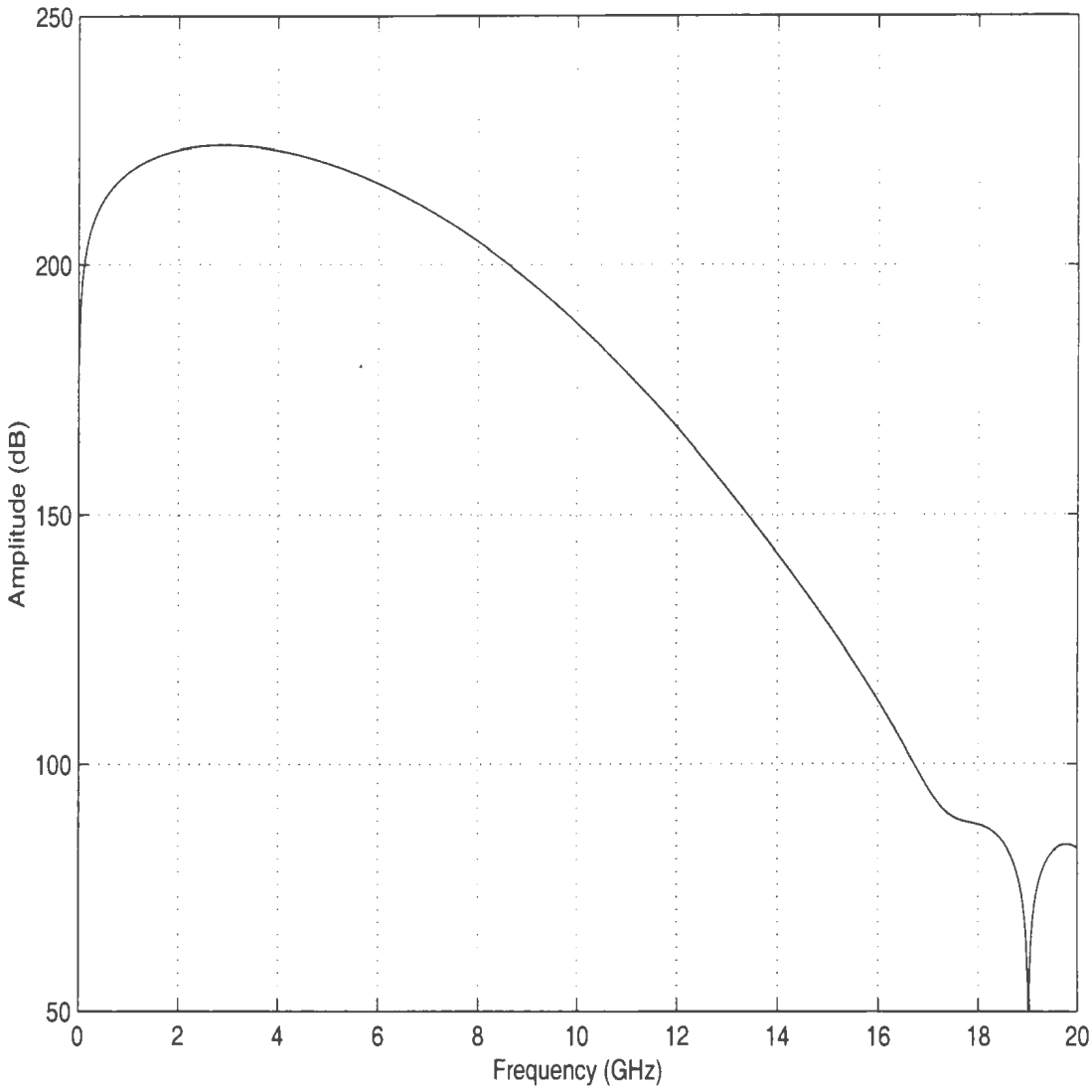


Figure 2.11: The spectrum of Rayleigh pulse with $\beta = 32$ and $er_0 = 16$

be obtained. After obtaining the field information on the outer surface containing the entire antenna system, it is easy to get the far-field information such as radiation patterns by field integration[7].

In this thesis, the Gaussian pulse and the Rayleigh pulse are used to find the input impedance of antennas, while sine wave is used to calculate the radiation pattern.

Chapter 3

Development of the FDTD Code

In this chapter the FDTD code developed for antenna design is introduced first. Then the factors that effect the accuracy of the results such as input impedance and radiation patterns are demonstrated numerically. These factors include the grid size, the distance between the Mur boundary and the antenna, the feed forms of the antenna and the source forms.

3.1 FDTD Code

Based on the formulation shown in the previous chapter, an FDTD code was developed for antenna design purpose. The code was written using C++, which can run in MS-DOS, Win95 and Unix environment.

The code is composed of one header file, one main program and several subroutine programs. The flow chart for the code is shown in Figure 3.1. The main program

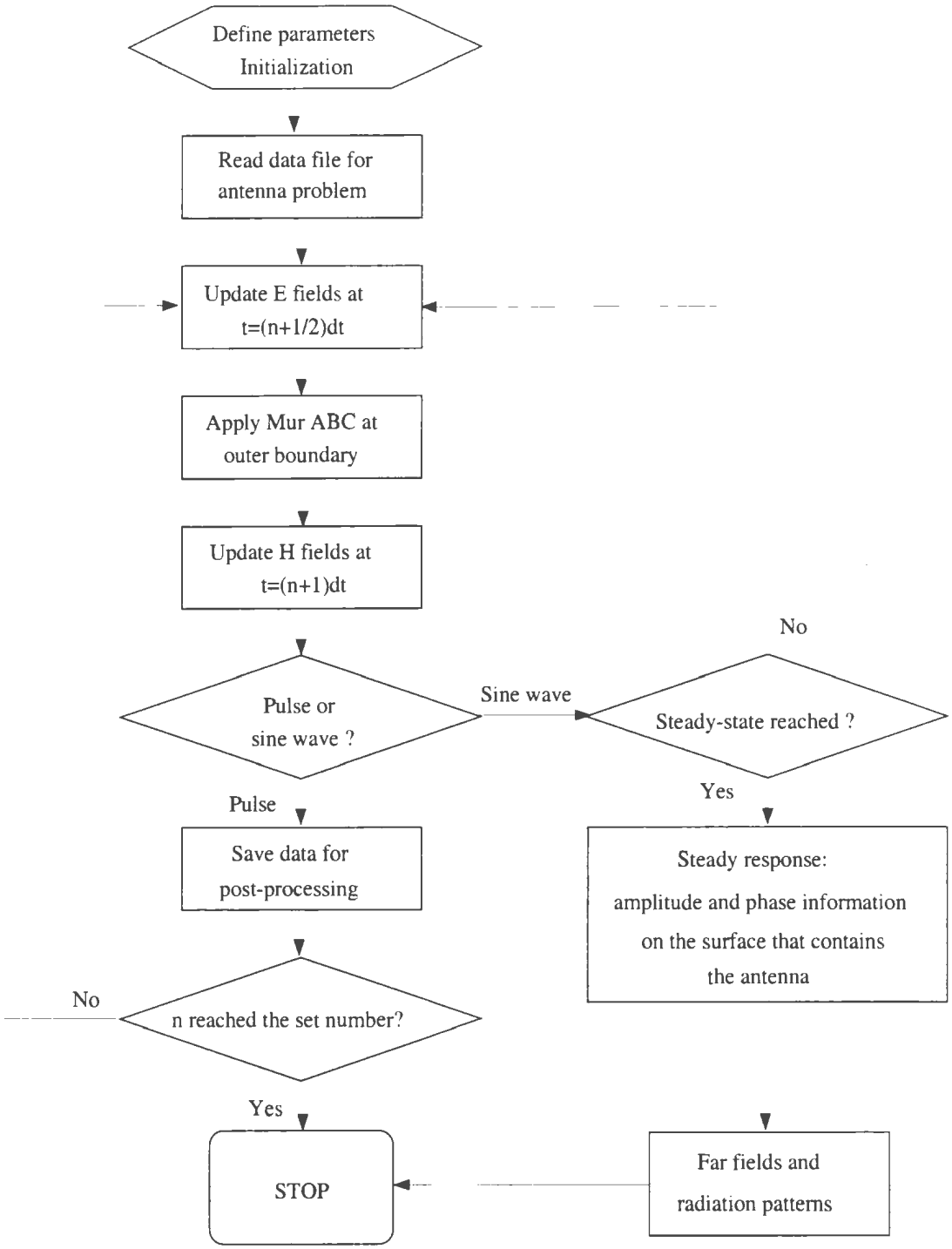


Figure 3.1: Flow chart of a C++ program for FDTD algorithm

"main" in the code reads the primary parameters, calls subroutines for initialization and controls the execution of the subroutines for field calculation as well as the data output. The functions of the subroutines are summarized in Table 3.1, while organization of the subroutines and data files in the FDTD code is shown in Figure 3.2.

First, "para" and "init" are called for parameter definition and variable initialization. Then "geo" is called to read the data file for the user-defined structure(an antenna and its surroundings). After that, "exfld", "eyfld", and "ezfld" are called for updating the E field at each grid except those on the boundaries. For the fields at those grids on the boundaries, "murxyp", "murxyn", "muryzp", "muryzn", "murxzp" and "murxzn", which implement the Mur boundary conditions introduced in the previous chapter are called instead. Then "hxfld", "hyfld" and "hzfld" are called for updating the H fields. If the antenna contains wire structures, "thin_wire" is called for the data input of the wire structures. This subroutine also saves the current (in time domain) on the wire in a separate output file, which can be used for post-processing such as impedance calculation.

If the source is sine wave and the far field is expected, "wave_stable" is called to decide if the steady-state is reached. If the steady-state is reached, "find_amp" and "find_phase" are called to find the field amplitude and the relative phase, respectively. Then "out_surface_field" is called to get the surface fields, and "far_field" is called to calculate the far field which will be output for plotting or other analysis.

In the following sections, we will introduce the subroutines in terms of their func-

Table 3.1: The functions of the subroutines

Subroutines	Functions
para, init	parameter definition and initialization
geo	antenna data input
exfld, eyfld, ezfld	E field updating
murxyp, murxyn, muryzp, muryzn, murxzp, murxzn	Mur boundary condition implementation
exfld, eyfld, ezfld	H field updating
thin_wire	dealing with the thin wire structures
wave_stable	judging if the steady-state is reached
find_amp	getting the field amplitude on the surface
find_phase	getting the relative field phases
out_surface_field	surface field calculation
far_field	far field calculation

tions. The description of the subroutines with similar functions will be put in the same section. Since there are detailed description in the previous chapter for the field updating, the boundary conditions as well as the treatment for thin wire structures, the emphasis here will be put on the subroutines for steady-state response, which were not described before. After that we will introduce the data input and output format as well as the procedures for using the program.

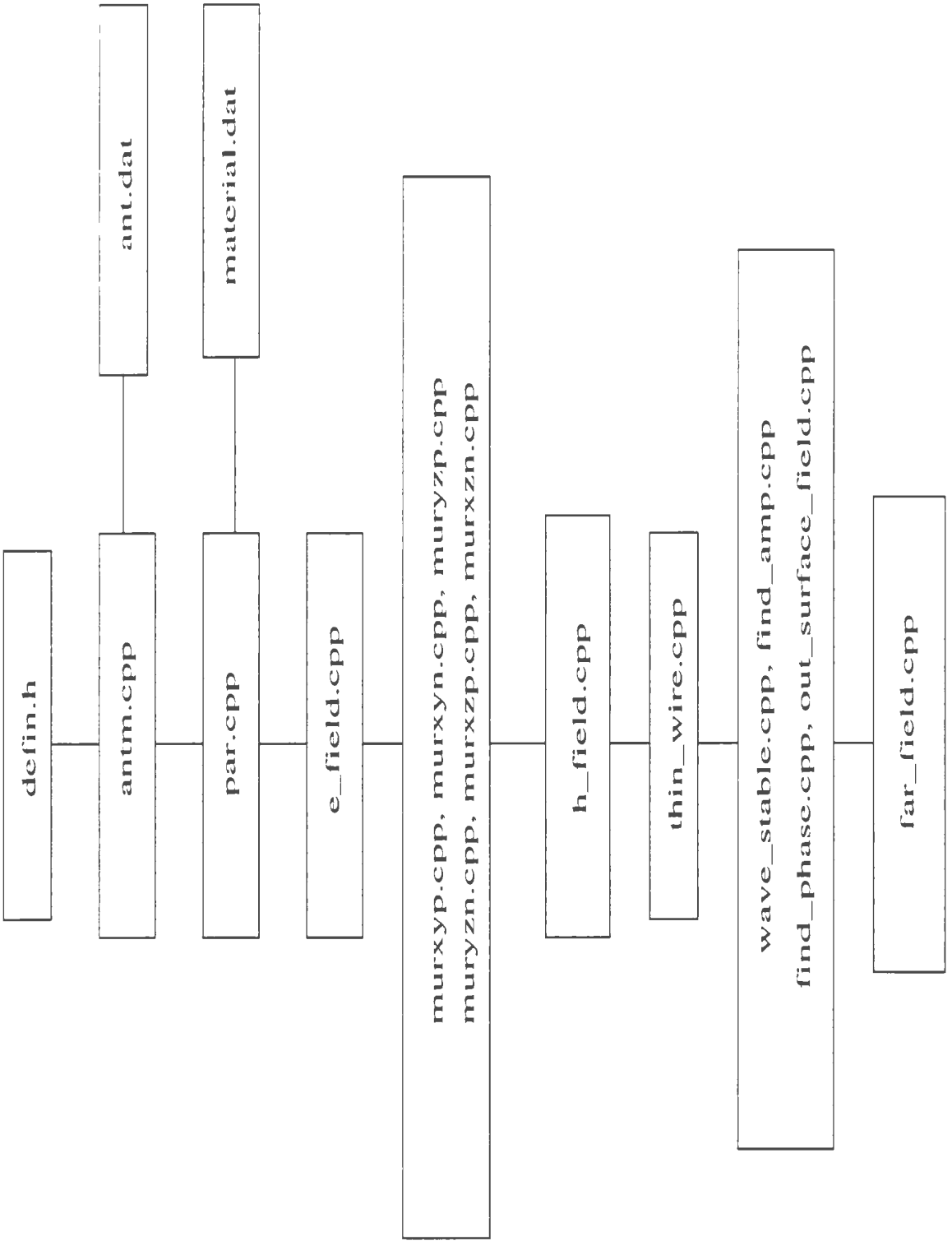


Figure 3.2: Subroutine and data file linkage chart

3.1.1 Header File

In the header file, various standard header files in the C++ library such as "math.h" (math library) and "iostream .h" (library for input and output) are included. In addition to the standard header files, the maximal size of the problem to be analyzed is given. For example, consider a problem of $100 \times 110 \times 120$ cells, we define $n\_x \geq 101$, $n\_y \geq 111$, and $n\_z \geq 121$. Here we define one more cell for the reason of code simplicity, since the arrays in C++ start from 0 instead of 1 and we do not want to use, say x[0], as the data for the cell Number 1. Various data structures also are defined, including:

- the actual size of the problem "*n_xyz*", which defines the number of the cells in x, y, z directions;
- the cell size "*del*0" defining the cell length in the x, y, z directions;
- the constant coefficient "*coff*0" for the convenience of expressing E fields in a simple way.

3.1.2 Initialization

Two subroutine programs "para" and "init" are called for initialization. The functions of "para" are

- defines the parameters conductivity σ, relative permittivity ϵ_r for the materials in the problem;

- calculates the time step size Δt by Courant stability condition equation (2.31) according to the cell size Δx, Δy, Δz;
- calculates the constant α $(= \frac{\sigma \tau_0}{\beta \Delta t})$ in Gaussian pulse in terms of the parameter β;
- defines the various constant coefficients such as $\frac{\Delta t}{\epsilon_0 \Delta z}$ in the field update equations.

The functions of "init.cpp" are

- initializes the E and H fields to zero;
- initialize the identity variable arrays for the material to free space;
- initialize the arrays, which temporarily store the field values on the boundary, to zero.

3.1.3 Field Computation within the Boundary

While looping over n (the index of the time steps), the subroutines "exfld", "eyfld" and "ezfld" are called for the E field updating in x-, y- and z- direction, respectively. The subroutines "hxfld", "hyfld" and "hzfld" are called for the H field updating, in x-, y- and z- direction, respectively. These subroutines calculate the present value of a component from its own prior time value and that of the nearest-neighbor field quantities according to the type of material present at that component location. The expressions for each field component can be found in the previous chapter, and the code for the field updating is straightforward, therefore we will not introduce them

furthermore. Since the formulae for the fields in different directions are different, we use different subroutines for the field updating in different directions here.

3.1.4 Fields on the Artificial Boundary

On the artificial boundary, the fields cannot be updated directly, as introduced in the previous chapter. The outer radiation boundary condition should be used instead to absorb the radiated fields at the outermost portion of the antenna space. In our code, the subroutines "murxyp", "murxyn", "muryzp", "muryzn", "murxzp" and "murxzn" are called for field updating on the outer boundary.

3.1.5 Thin Wire Structure

If there are thin wire structures, "thin_wire" will be called to calculate the H fields around the wire. At each time step the fields in the source region are stored for post-processing such as input impedance calculation.

3.1.6 Steady-State Response

As mentioned before, for the far field calculation, sine steady-state response method is more efficient than FFT method. Specifically, far fields are not sensitive to the distance between the outer absorbing boundary and the antenna, which will be verified later numerically. This makes the problem size much smaller, and hence the corresponding running time of the program is reduced greatly. In this program, if the

source is sine wave, the subroutine "wave_stable" can decide if the steady state has been reached or not. If the steady-state is reached, "find_amp" and "find_phase" will be called respectively to obtain the amplitude and phase information on the surface containing the antenna for far field computation.

A. Steady-State Condition

Figure 3.3 shows the electric field varying with time at a certain point. The corresponding steady-state wave form is shown in Figure 3.4. It is clear from the figures that one cannot judge that the response has reached steady-state condition just by comparing the peak-values within two continuous cycles.

To solve this problem, one may consider peak-values in four continuous cycles. Assume the positive peak values are A_1, A_2, A_3, A_4 in four cycles, respectively, if

$$\left| \frac{A_i - A_j}{A_i} \leq e_c, \right| \quad i = 1, ...4, j = 1, ...4, i \neq j \tag{3.1}$$

a steady-state condition is considered achieved. Note e_c is a constant controlling the steady-state error and is input from the data file "ant.dat", varying from 1% to 0.1% in this program.

From Figure 3.4, one can also observe that the discreting error around the peaks(the curve is not smooth). Obviously, when e_c is very small, the corresponding $\triangle t$ should be decreased also. Otherwise, we have the risk of getting into dead looping(in which case the expected steady-state will never be reached).

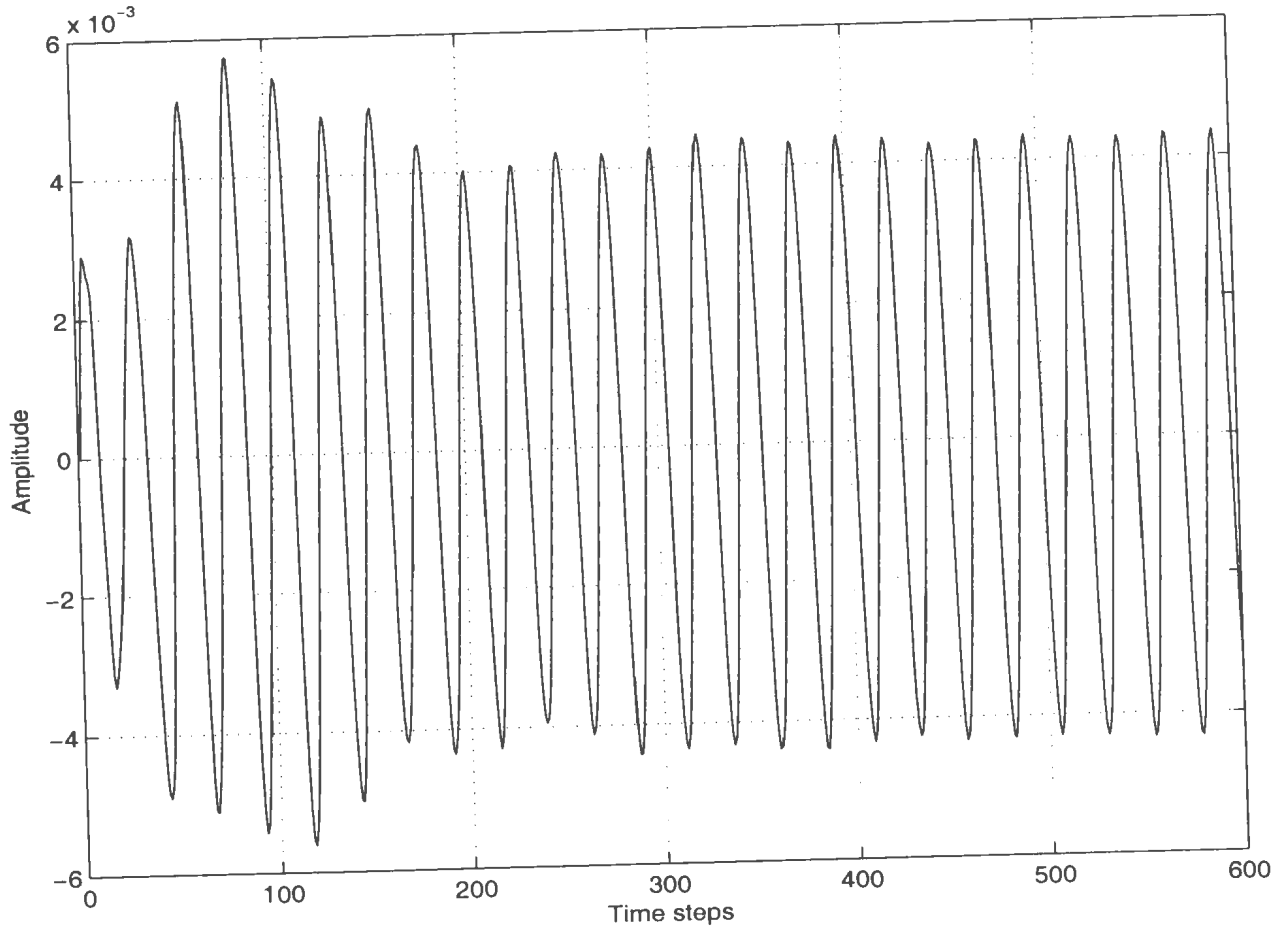


Figure 3.3: The form of a field component in time domain

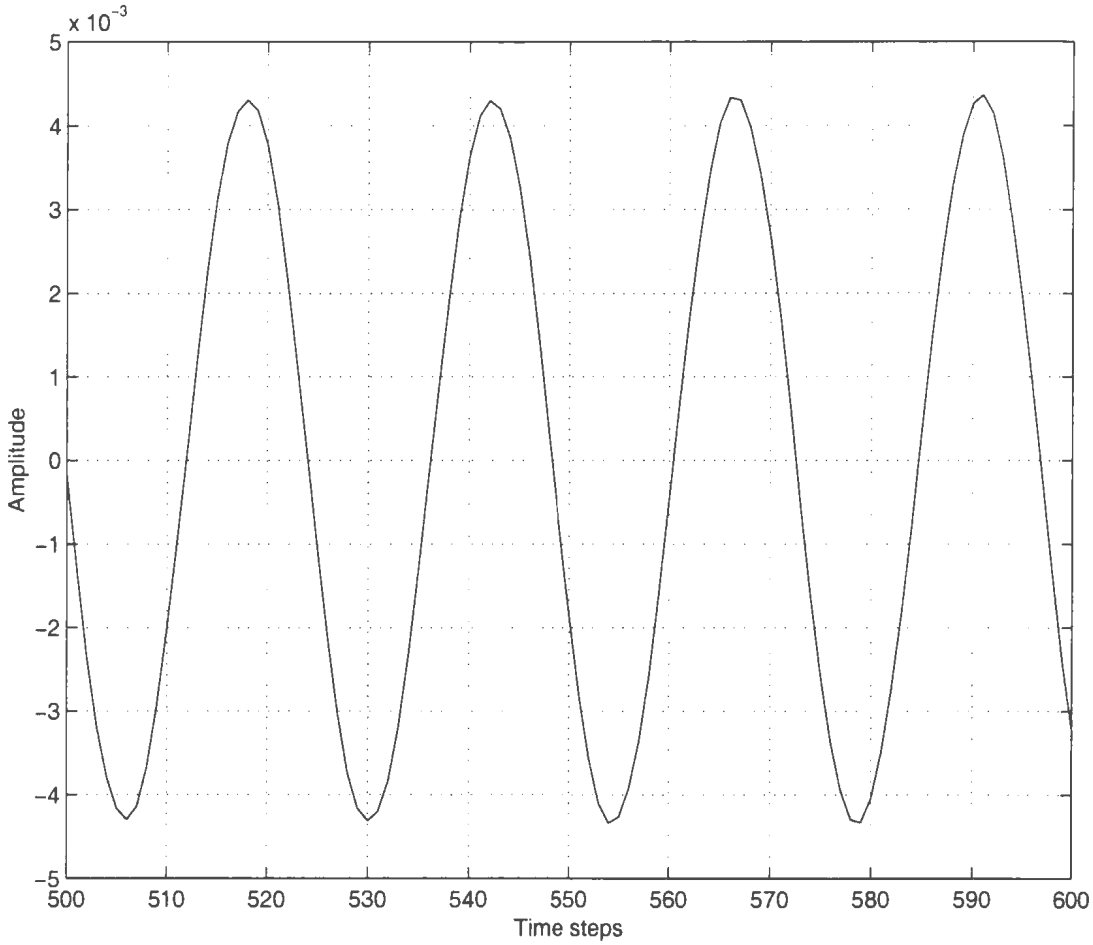


Figure 3.4: Steady-state wave form

B. Find the Amplitude and Phase of the Sine Wave

It is easy to get the amplitude information for fields at each cell. When steady state condition is achieved, one can find the maximum values for each field component at each point just by simple comparison. The maximum values are the amplitude information expected.

For the phase information, a point is chosen as a reference point, as shown in Figure 3.5. When the field (E or H) at this reference point jump from a negative value to a positive one or zero, the phase for the field at this point is chosen to be zero, the index of time step is recorded. To find the relative phase of other fields in the space with respect to the field at this point, one just needs to find the index of time step when the field value jumps from negative to positive or zero. The formula for the relative phase is

$$\phi = \frac{2\pi}{N_c} * (n - n_0) - sin^{-1}(\frac{E_x}{A}) \tag{3.2}$$

where N_c is the number of time steps in one period and n_0 is the index of time step at the reference time and n is the index of time step when a field value jump from negative value. A is the amplitude of the field and E_x is the value the field jump to from a negative value. It should be noted that for the phase of H field, π/N_c should be added because there is a time difference $\triangle t/2$ between E field and H field.

As soon as one finds the fields on the surface enclosing the antenna and its attached structure, the formulation in section 2.6.2 can be used to obtain the far fields. The radiation pattern is obtained by using plotting tools such as "*Matlab*".

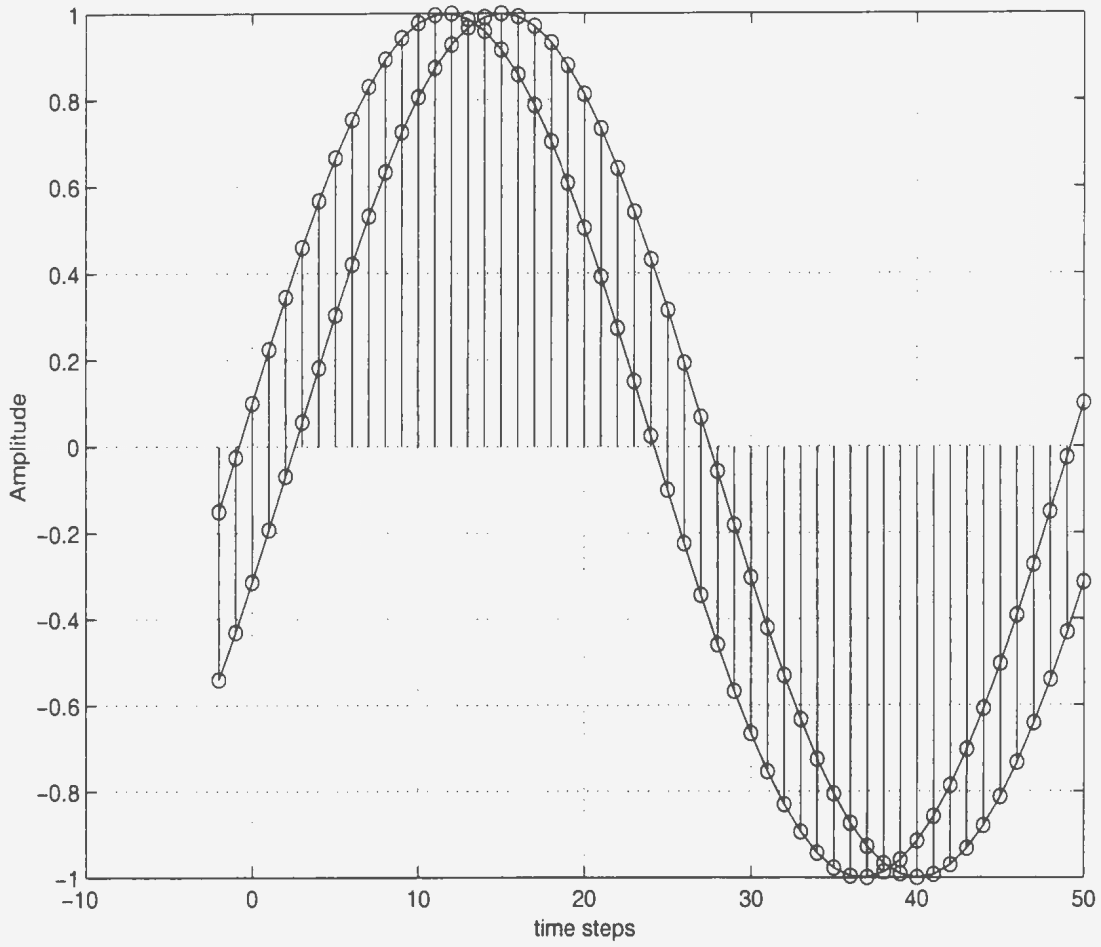


Figure 3.5: Method of finding the steady-state phase information

3.1.7 Data Input

There are three input data files: "ant.dat". "material.dat", "wire.dat". An example for the data files is given in the appendix. The data file "ant.dat" contains the primary parameters for the problem to solved, such as the number of cells, the cell size, and the data file name of the antenna geometry and material structures whose format is given in table 3.2. The user just needs to change the values of the parameters in the

file.

Table 3.2: The data format of file ant.dat

Input data		Comments
line 1	$nn.x$, $nn.y$, $nn.z$	number of cells along x, y, z direction
line 2	$del.x$, $del.y$, $del.z$	cell size (unit: meter)
line 3	r_0	thin wire radius (unit: meter)
line 4	*amp*, *beta*, *lambda*	"A" and "β" of the pulse which are used only for impedance calculation, wavelength "λ" for sine wave which is used only for radiation pattern calculation
line 5	*nstop*	maximal allowed number time step.
line 6	*material_File*	file name of the antenna structure
line 7	*flag_inc*	flag controlling the form of the source. 0: pulse; 2: sine wave
line 8	Ist, Ind, Jst, Jnd, Kst, Knd	the surfaces for far field calculation, which should be as small as possible provided they enclose the antenna
line 9	e_c	the constant controlling the steady-state error
line 10	$cen.x$, $cen.y$, $cen.z$	the index numbers for the origin of the coordinate for far field calculation
line 11	$epsi1$, $epsi2$, ...	the dielectric constant for the materials used

If the file name of the antenna structure(in the sixth line of "ant.dat") is "mate-

rial.dat", the user will create this file manually or by a small program. There are six integers in each line in "material.dat". They are i, j, k, $ID\_x[i][j][k]$, $ID\_y[i][j][k]$, $ID\_z[i][j][k]$. The former three numbers are the cell index numbers, and the latter three numbers are the material(permittivity) index numbers. We use three index numbers here because the field components are offset by half cell size, as discussed in the second chapter. In this code, the index number for perfect conductor is 1. For dielectric material, an index number can be any number which is less than 10, but it cannot be 1. It should be noted that the dielectric constant corresponding to the index number has to be input in the last line of "ant.dat".

"wire.dat" contains the thin wire structure in the antenna problem. The structure of "wire.dat" is shown in table 3.3. It should be noted that no inclined wires are allowed directly. The inclined wires can be represented by using stair step.

Table 3.3: The data format of file wire.dat

Input data	Comments
n	total number of straight thin wires involved
Z, x, y, z1, z2	"Z" oriented wire, starting and ending index number
Y, x, y1, y2, z	"Y" oriented wire, starting and ending index number
X, x1, x2, y, z	"X" oriented wire, starting and ending index number

3.1.8 Data Output

The currents on the antenna(in time domain) are output to the file "cur.dat". If the wide band impedance is expected(the source is a pulse), we need to take the Fourier transform of the data, which is straightforward. If the patterns are what we are interested in(the source is a sine wave), we can get them directly from the output data files "Ectx.dat", "Ecty.dat", and "Ectf.dat", which correspond to the pattern in xz- yz- and xy- planes respectively.

3.1.9 Procedure for Using the Code

(1.) Prepare the three input data files. (2.) Run the code "antm". (3.) Take the FFT of the time domain data to get the input impedance or use *Matlab* to plot the radiation patterns.

3.2 Validation of Present FDTD Code

To test the accuracy and robustness of this developed code, comparison is made with an available code *Numerical Electromagnetic Code* (NEC-2) for some cases. Three models are presented here. They are a dipole, a loop antenna, and a monopole on a finite plate. The results from NEC-2 will be used as reference. In NEC-2, which is based on the method of moment, wire-grid replaces the flat plate shape. First the effects of Mur distance and cell size on the accuracy will be discussed. Then one

compares the results from Gaussian pulse and Rayleigh pulse for different feed forms.

3.3 Dipole

Dipole has been the simplest antenna since the early time when antenna was invented. It is a very good model for the validation of our code since no approximation is introduced for the antenna when using NEC-2. The NEC solution is actually being used as exact solution for dipole antennas. The computation cost is low for this simple structure.

A center fed dipole of total length $15cm$ with diameter of $1.0mm$ is considered here (the diameter of the inner conductor for RG402/U 50 semi-rigid coaxial cable is 0.9195mm, hence $1.0mm$ is a very good approximation to the actual antenna). The radiation patterns and the input impedance will be calculated separately.

3.3.1 Radiation Pattern

Figure 3.6 shows the radiation pattern of the above antenna at 3GHz. The solid line is obtained from NEC-2, where 61 segments, which corresponds to $\lambda/40$ length per segments, was used. The dashed line and the dotted line are obtained by our present FDTD code. For the dashed line, the cell size is $\lambda/20$, the distance between the outer boundary and the antenna is λ (i.e. 20 cells). For the dotted line, the cell size is $\lambda/40$, the distance between outer boundary and the antenna is still λ (i.e. 40 cells). From the Figure one can see that for the cell size of $\lambda/40$ very good agreement between

FDTD and NEC-2 is obtained for the radiation patterns of a dipole antenna. When the steady-state judging constant e_c changes from 1% to 0.1%, no obvious change in radiation patterns is observed. It should be noted that when e_c is very small, the time step $\triangle t$ should be comparatively small too, otherwise it it possible that one will not get the steady-state expected. Since the running time would be much longer when e_c is smaller, e_c will be fixed at 1% in the following calculations.

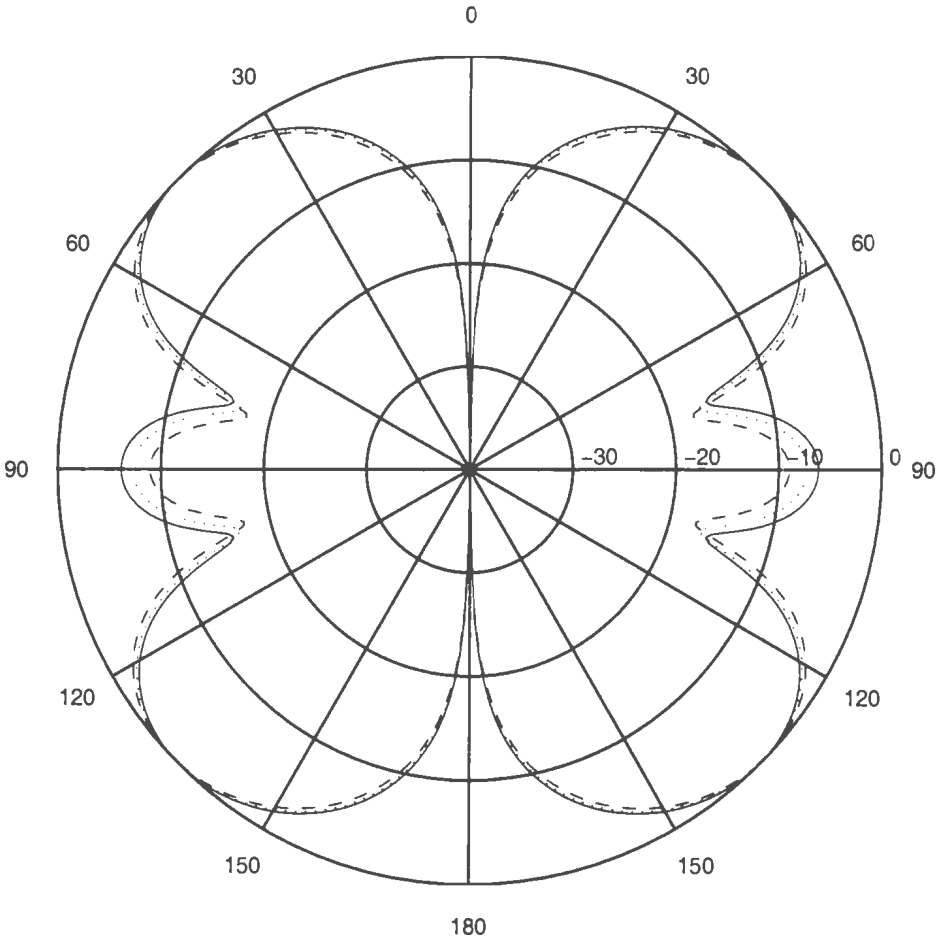


Figure 3.6: The radiation pattern of a dipole at 3GHz obtained using different FDTD cell size

—— NEC-2, - - - cell size:$\lambda/20$, ····· cell size:$\lambda/40$.

Next we test the effects of the distance between Mur boundary and the antenna (we call this distance the Mur distance herein). Theoretically, the larger the Mur distance, the better the radiation pattern results. In the meantime, the computational cost will be much higher when using larger distance. Figure 3.7 shows the radiation pattern for various Mur distances. The cell size is $\lambda/40$. Mur distance of 10 space cells ($\lambda/40$ per cell) is acceptable, though more cells produce better results. So at initial stage of radiation pattern estimation, 10 cells are used and in the final design stage 20 cells or more are used until the radiation patterns converge to accurate values.

Then the radiation patterns of the antenna at 2GHz for various cell sizes and different Mur distances are calculated. Figure 3.8 shows the radiation pattern of the above antenna at 2GHz with different FDTD cell size. The solid line is obtained from NEC-2, where 41 segments, which corresponds to $\lambda/40$ per segments, were used. For the dashed line, the cell size is $\lambda/20$. For the dotted line, the cell size is $\lambda/40$. For the dash dotted line, the cell size is $\lambda/80$. The distance between the outer boundary and the antenna for all the cell size is 20 cells. It can be seen that keeping the Mur distance a constant number of cells (with varying absolute distance), the accuracy can still be improved a lot by reducing the cell size.

When the cell size is small enough, one can see the effects of Mur distance on the accuracy from Figure 3.9, where the cell size is $\lambda/80$ and the dashed and dotted line correspond to the Mur distance of 10 cells and 20 cells, respectively.

From the above numerical results, one can draw the following conclusion for the

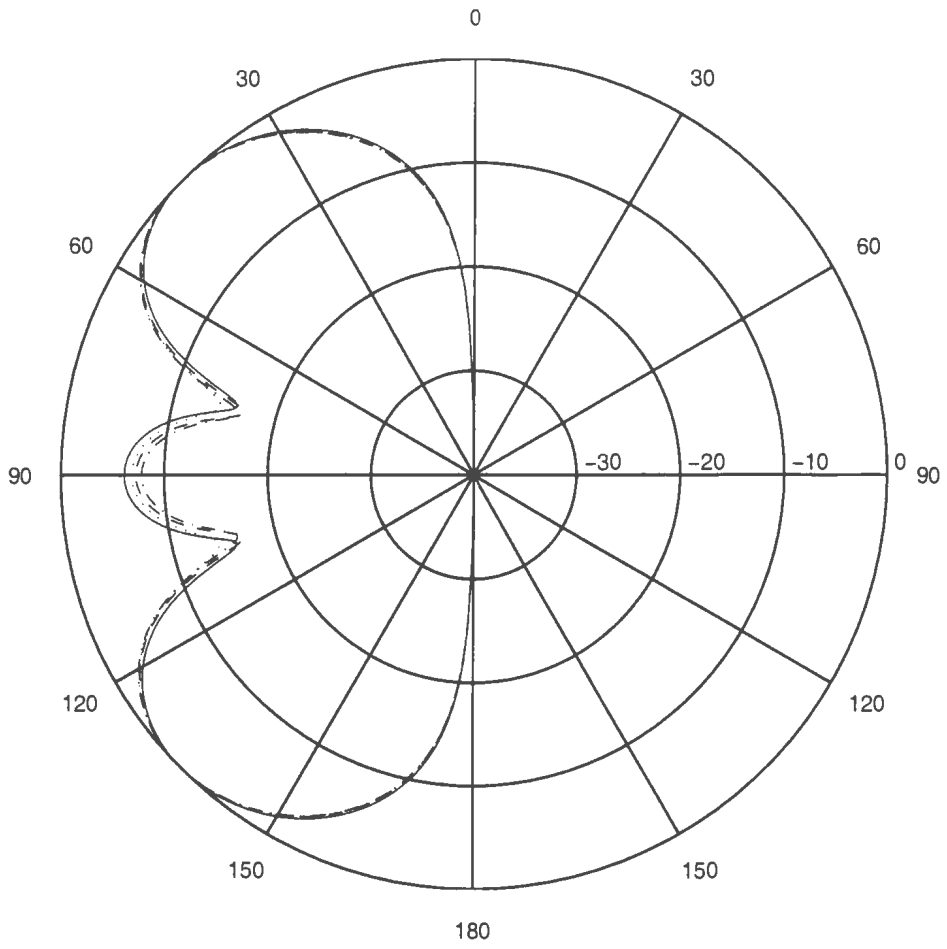


Figure 3.7: The radiation patterns of the dipole at 3GHz with different Mur distance

—— NEC-2, ····· 10 cells, — · — 20 cells, - - - 40 cells.

radiation pattern calculations: The cell size must be less than $\lambda/40$ in order to get very accurate result. The radiation patterns are not sensitive to the Mur distance. The steady-state controlling constant is chosen to be 1% to get good result, while not wasting computational time.

Actually, the radiation pattern of dipoles has been studied before [21]. But the results in [21] are not very good, though it coincides with that of sine current

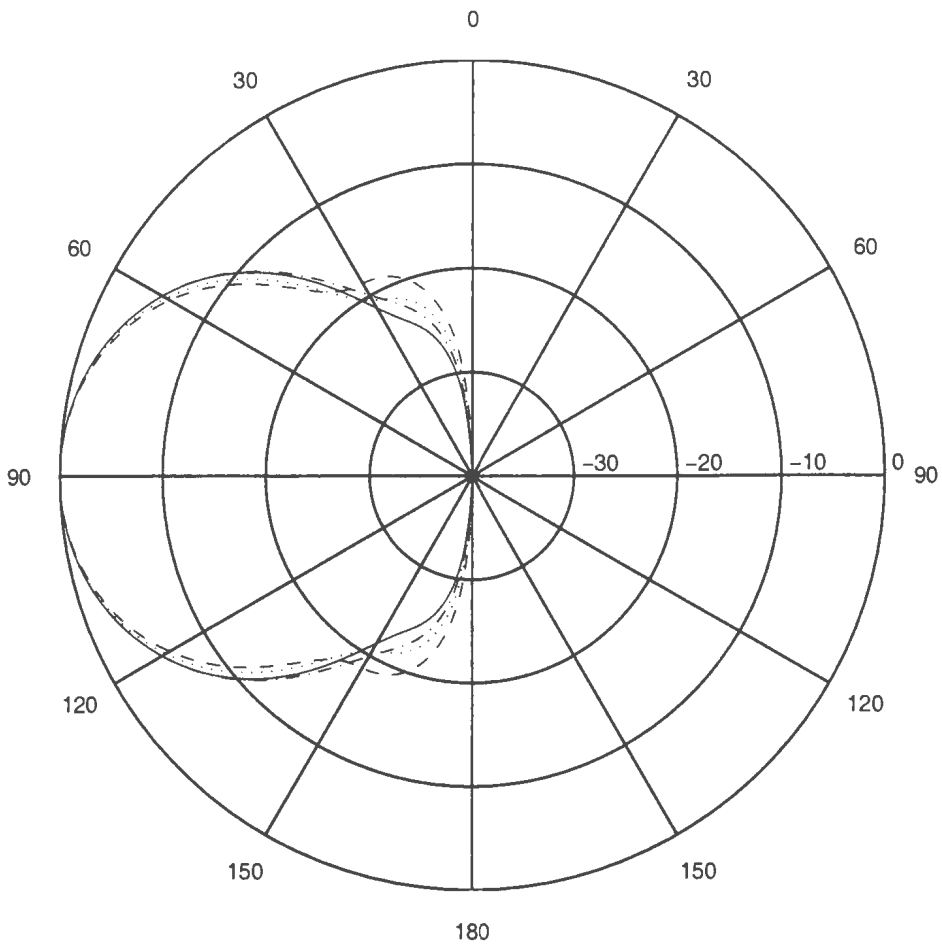


Figure 3.8: The radiation patterns of the dipole at 2GHz with different cell size

—— NEC-2; - - - $\lambda/20$; ····· $\lambda/40$; — · — $\lambda/80$;

approximation.

3.3.2 Input Impedance

First, the input impedance of the above dipole with a thin gap voltage feed was calculated by using Gaussian pulse and Rayleigh pulse as the source form, respectively, and no difference was observed. So we used only the Gaussian pulse to test the effects

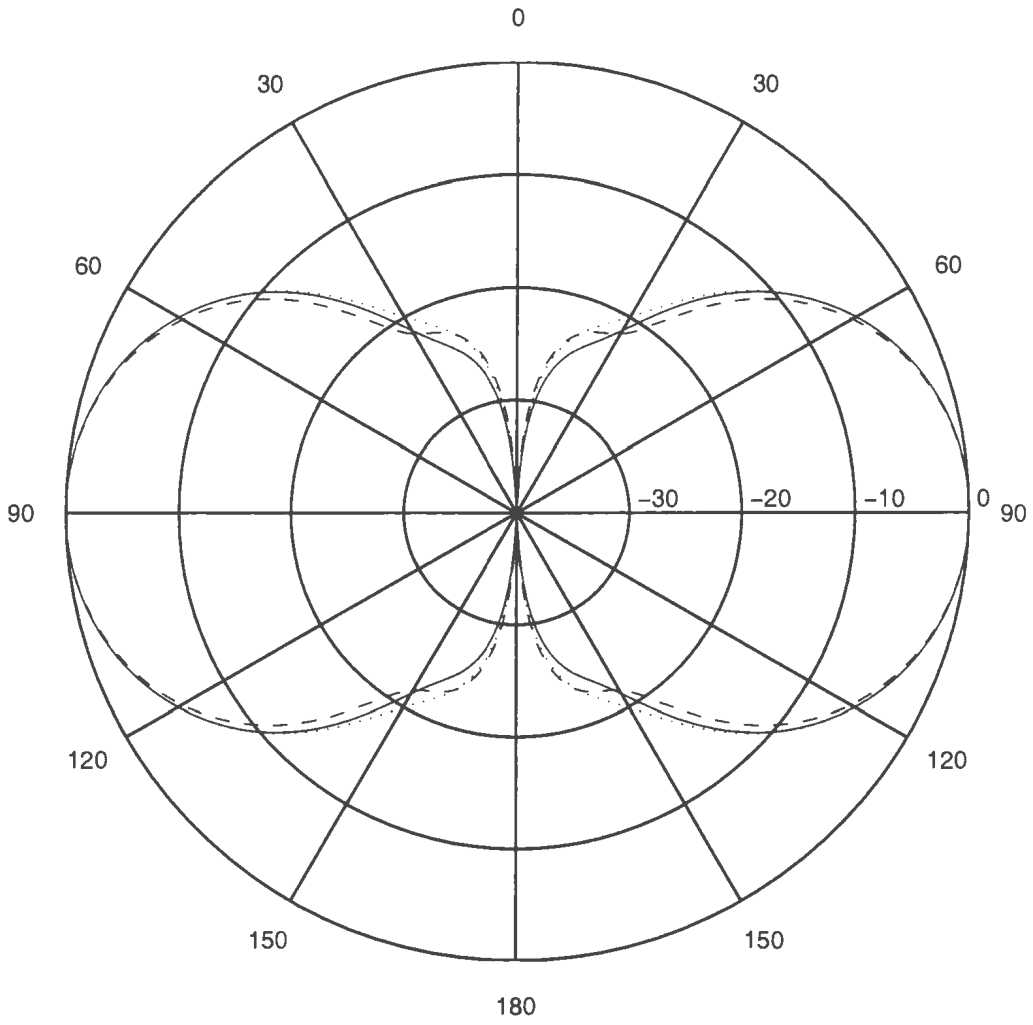


Figure 3.9: The effects of Mur distance on the accuracy of the radiation patterns

—— NEC-2; – – – 10cells ; ····· 20 cells.

of Mur distance and cell size unless specified otherwise.

Figure 3.10 shows the input impedance versus frequency, where the cell size is $\lambda/40$ and the dashed and dash dotted lines correspond to the Mur distance of 20 cells and 40 cells, respectively. From the figure, one can observe that the Mur distance affects the input impedance mainly near its peak values of the real and imaginary

parts.

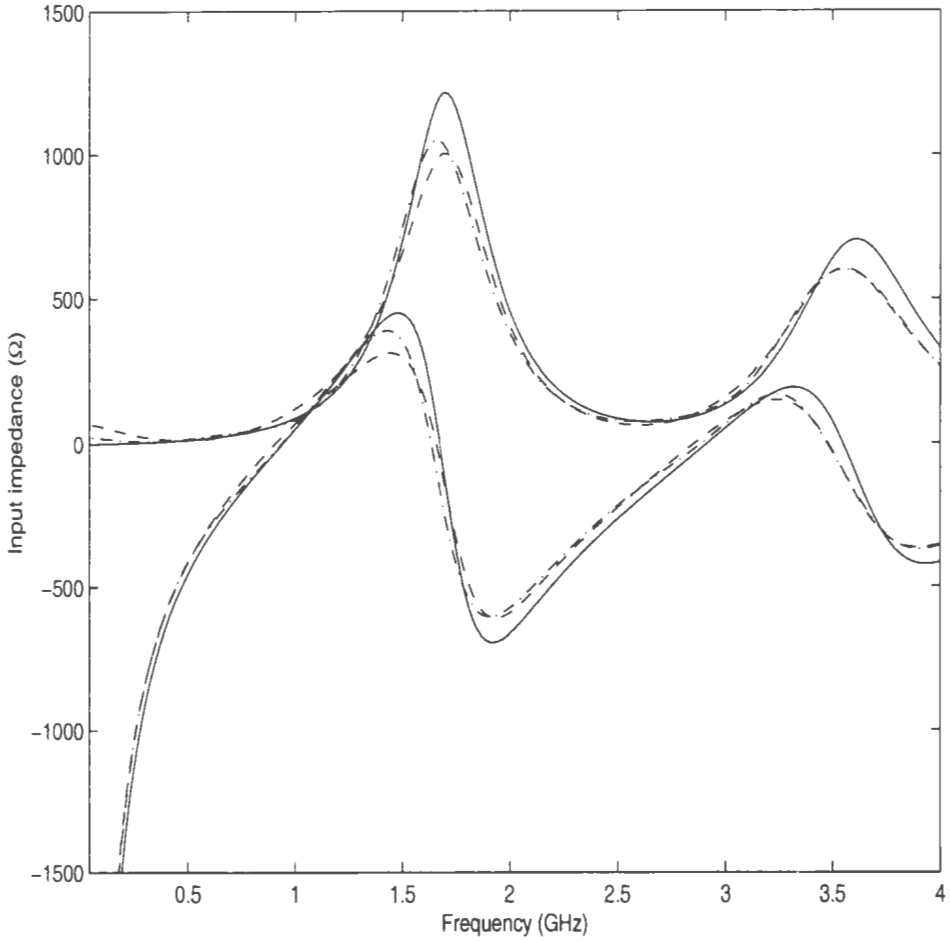


Figure 3.10: The effects of Mur distance on the accuracy of the input impedance

—— NEC-2; – – – 20 cells ; — · — 40cells.

Then the role of the cell size is explored. Figure 3.11 shows the effects of cell size, where the Mur distance is 40 cells and the dash dotted and dotted lines correspond to the cell size of $\lambda/40$ and $\lambda/50$, respectively. It is clear the smaller cell size produces more accurate results at the expense of much higher computational costs. The input impedance is more sensitive to cell size and Mur distance than radiation patterns. In

order to get meaningful data, cell size of $\lambda/40$ and Mur distance of at least 20 cells are to be used for input impedance calculation in final designing process.

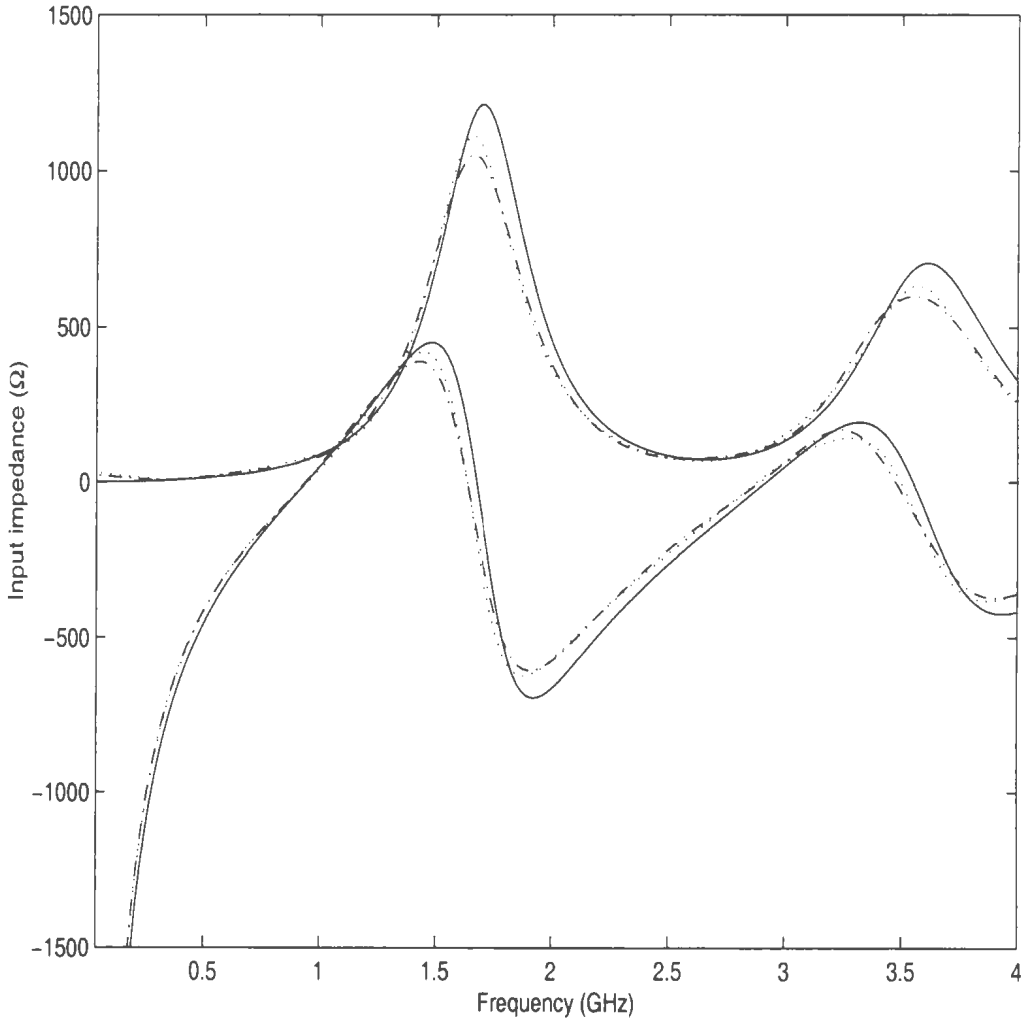


Figure 3.11: The effects of the cell size on the accuracy of the input impedance

—— NEC-2; — · — $\lambda/40$; ····· $\lambda/50$

3.4 Loop Antenna

Loop antennas are very important antennas in mobile communications. At present, they are widely used in pagers. Since the input impedance is very small at low frequencies, it has not been applied in transmitters such as cellular phones. But its resistance against noise has received much attention. The radiation pattern is shown in Figure 3.12. Clearly, the agreement between the result from FDTD and from NEC-2 is very good.

Then a thin gap voltage source with Gaussian pulse is used for input impedance calculation. It is found that the current in source region approaches a constant not equal to zero, as shown in Figure 3.13. The corresponding input impedance is shown in Figure 3.14. Obviously, the input resistance cannot be negative. Therefore the solution is not acceptable. It may be explained by what happens when a voltage source is shorted. When the load impedance is very very small, the current in the circuit cannot be measured accurately, the input impedance obtained from the ratio of the voltage to the current is not correct.

Then the source form is changed to Rayleigh pulse with the same thin gap feed. The resulting current wave is shown in Figure 3.15. and the input impedance in shown in 3.16. In this case, excellent agreement between our FDTD approach and NEC-2 is obtained.

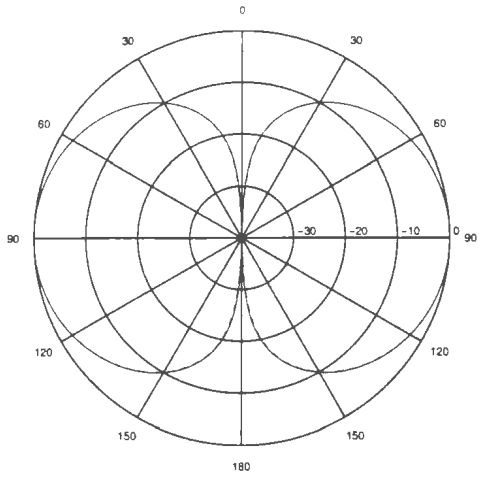
When a current source is used to feed the loop antenna, both Gaussian and Rayleigh pulse produce accurate impedance results.

Table 3.4: Feed forms and pulse forms for loop antennas

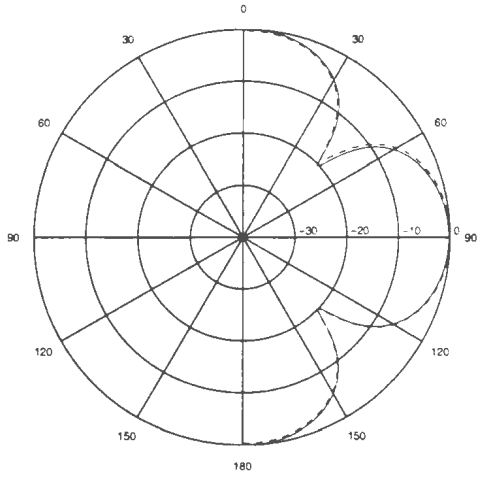
input impedance result		Feed Form		
		thin gap Voltage	Magnetic frill	Current
Pulse Form	Gaussian	not acceptable	not acceptable	good
	Rayleigh	good	good	good

Table 3.5: Feed forms and pulse forms for open circuit type antennas

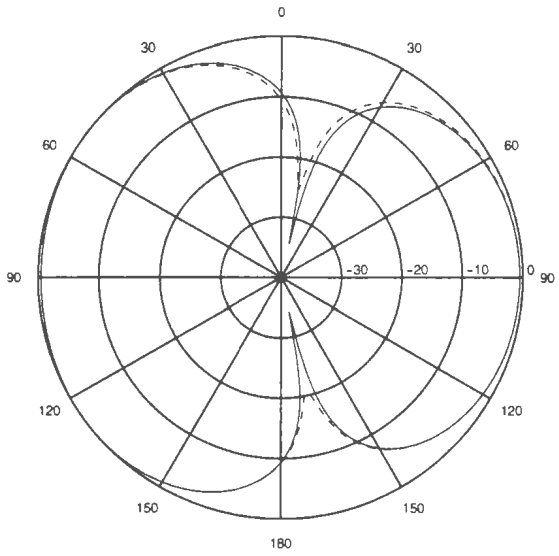
input impedance result		Feed Form		
		thin gap Voltage	Magnetic frill	Current
Pulse Form	Gaussian	good	good	not acceptable
	Rayleigh	good	good	good



(a) x-z plane



(b) y-z plane



(c) x-y plane

Figure 3.12: The radiation pattern of a loop antenna using delta-gap voltage source

—— NEC-2; – – – FDTD.

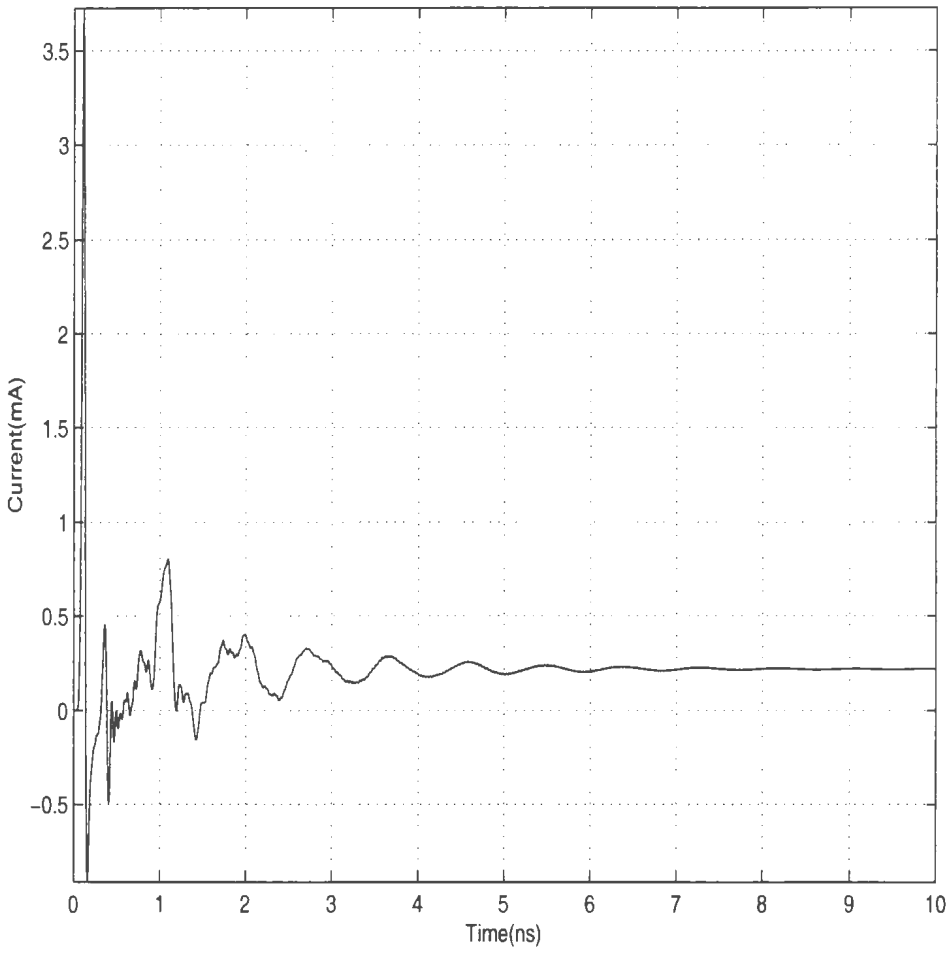


Figure 3.13: The resultant current in the source region of the loop antenna

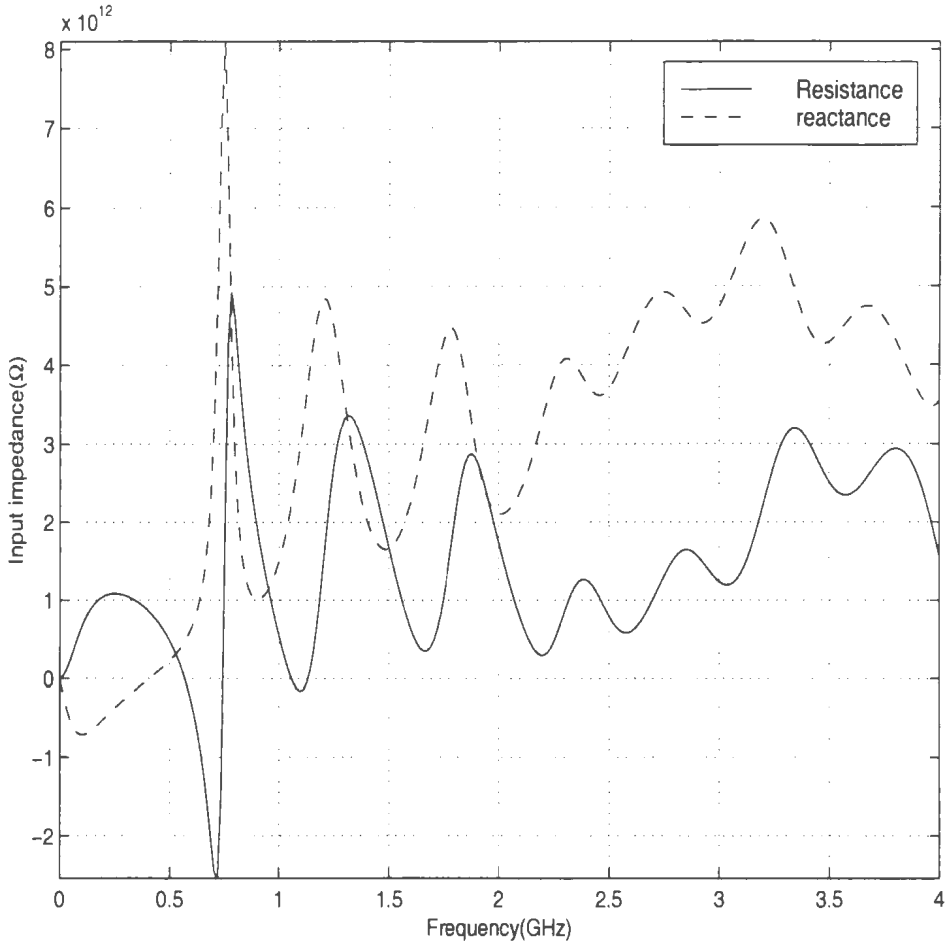


Figure 3.14: The input impedance of an loop antenna using delta-gap voltage source

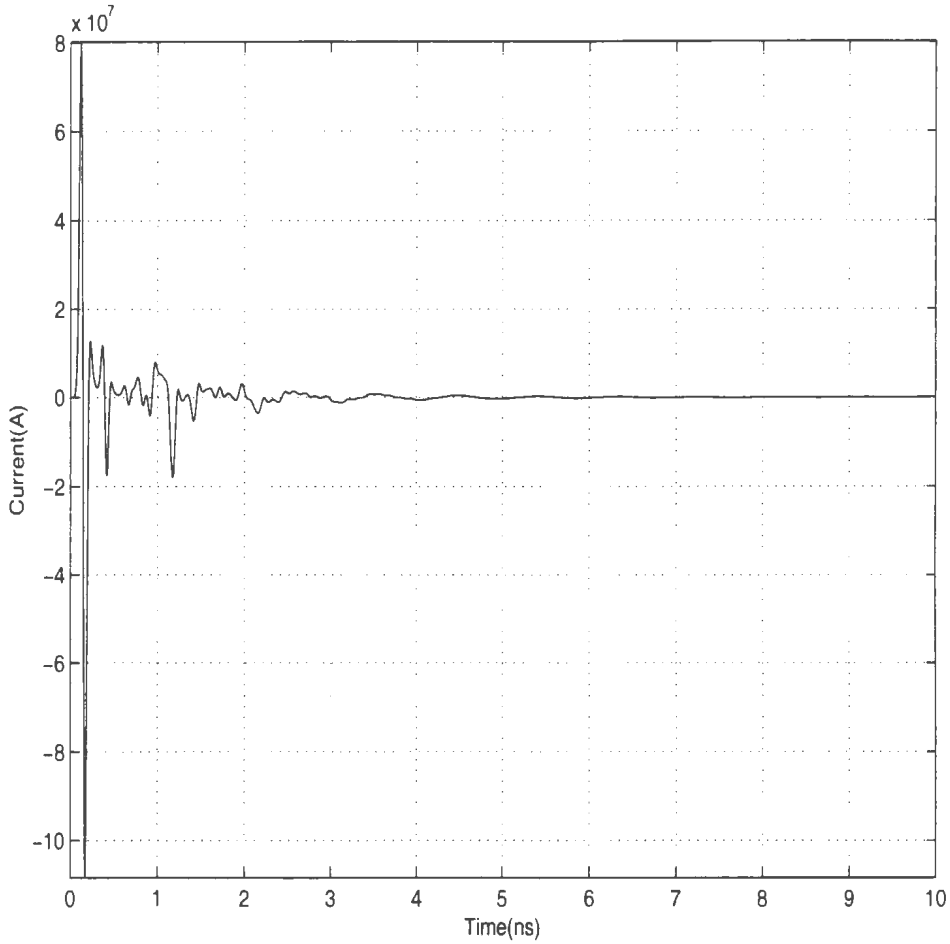


Figure 3.15: The resultant current in source region of loop antenna with Rayleigh pulse

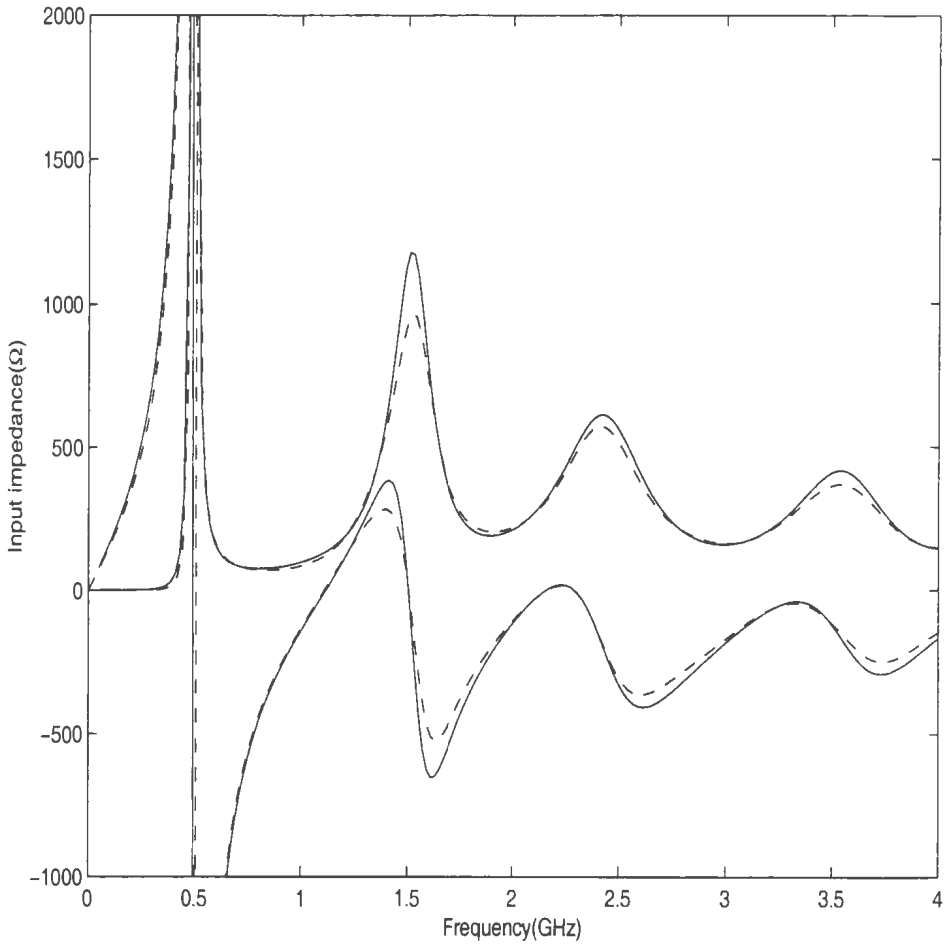


Figure 3.16: The input impedance of an loop antenna with Rayleigh pulse

—— NEC-2; - - - FDTD

3.5 A Monopole on a Finite Conducting Plate

In this section, the performance of a monopole on a finite conducting ground plane will be studied, which is an approximation of practical situation. The monopole is $7.5cm$ long, and the ground plane is $30cm \times 30cm$. The operation frequency is $1GHz$.

Figure 3.17 shows the normalized amplitude radiation pattern of the monopole. The FDTD grid resolution is $\lambda/40$. The agreement between the result from FDTD and that from NEC-2 is excellent.

Figure 3.18 shows the input impedance of the antenna at frequency band up to 4GHz. The agreement is not as good as for radiation patterns because the input impedance calculation procedure varies considerably between the two codes, i.e. FDTD and NEC-2. The wire-grid model is just an approximation of the solid surface, and the input impedance, unlike the radiation pattern, is very sensitive to the model. For the FDTD approach, it is perceived that the results are considered reliable if one gets the same result using different cell size.

Figure 3.19 shows the same antenna on a $60 \times 60cm^2$ plate, where the cell size is $\Delta x = \Delta y = \lambda/20$ and $\Delta z = \lambda/40$. Again, the agreement is very good, though the grid on the horizontal plane is not small.

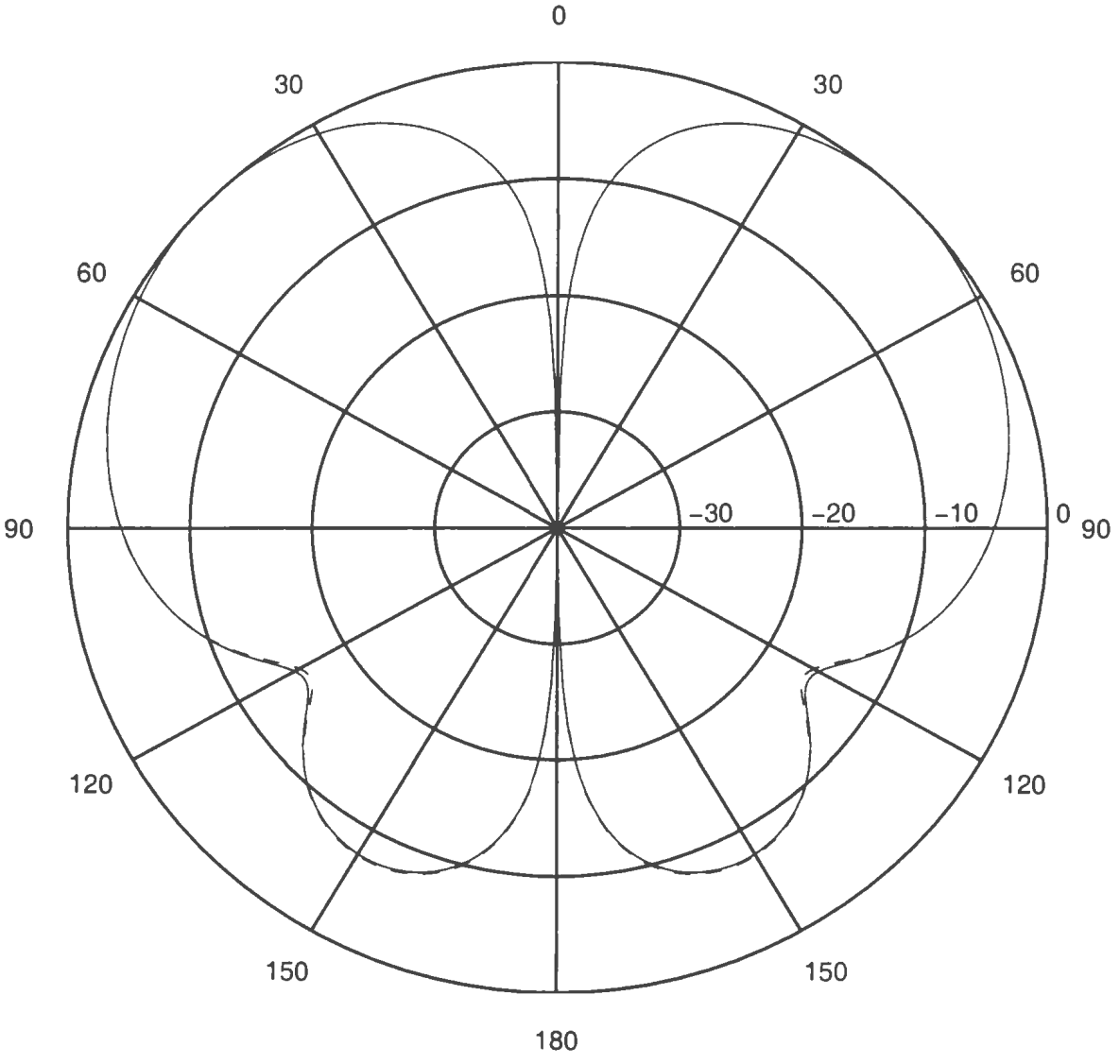


Figure 3.17: The radiation pattern of a monopole on $\lambda \times \lambda$ ground

—— NEC-2; - - - FDTD.

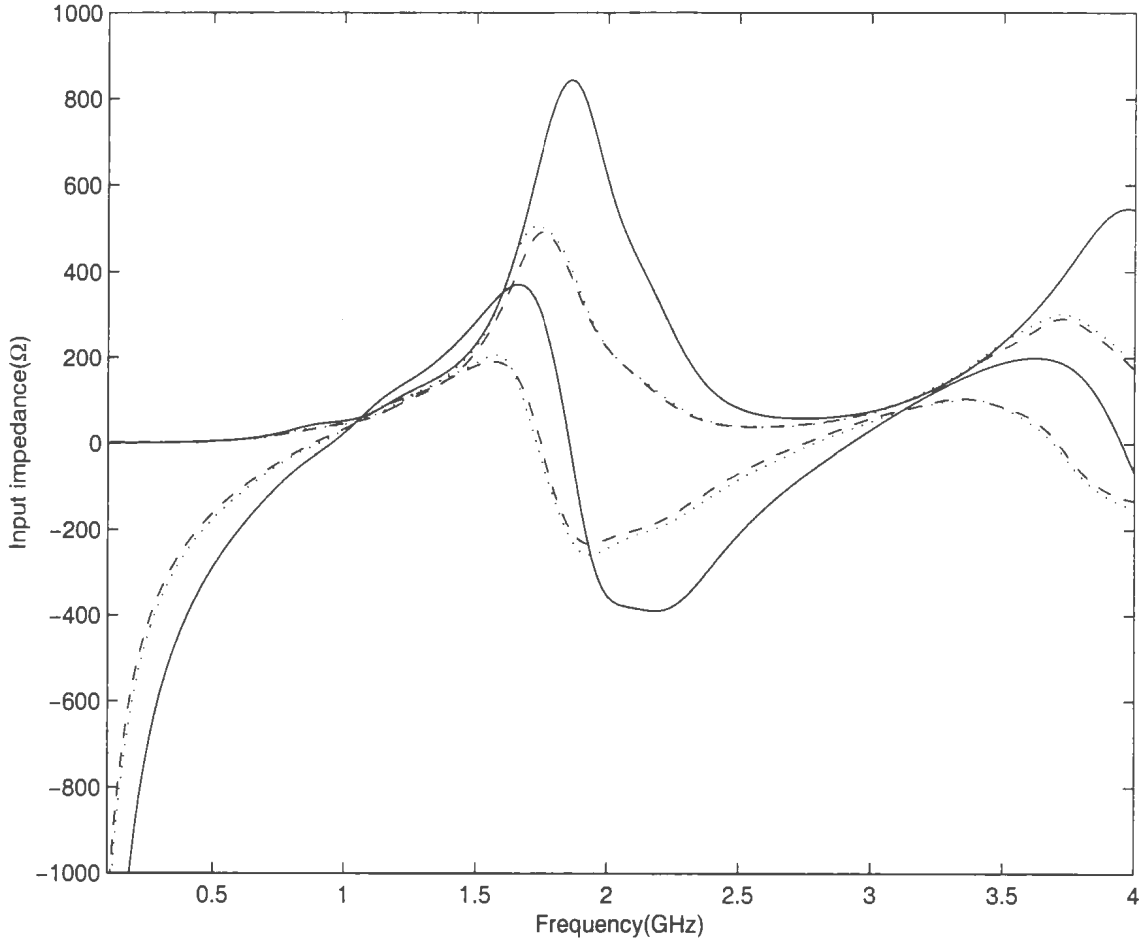


Figure 3.18: The input impedance of a 7.5cm monopole on a 30cm × 30cm ground

—— NEC-2; – – – cell size FDTD:$\lambda/40$; ····· cell size FDTD:$\lambda/50$.

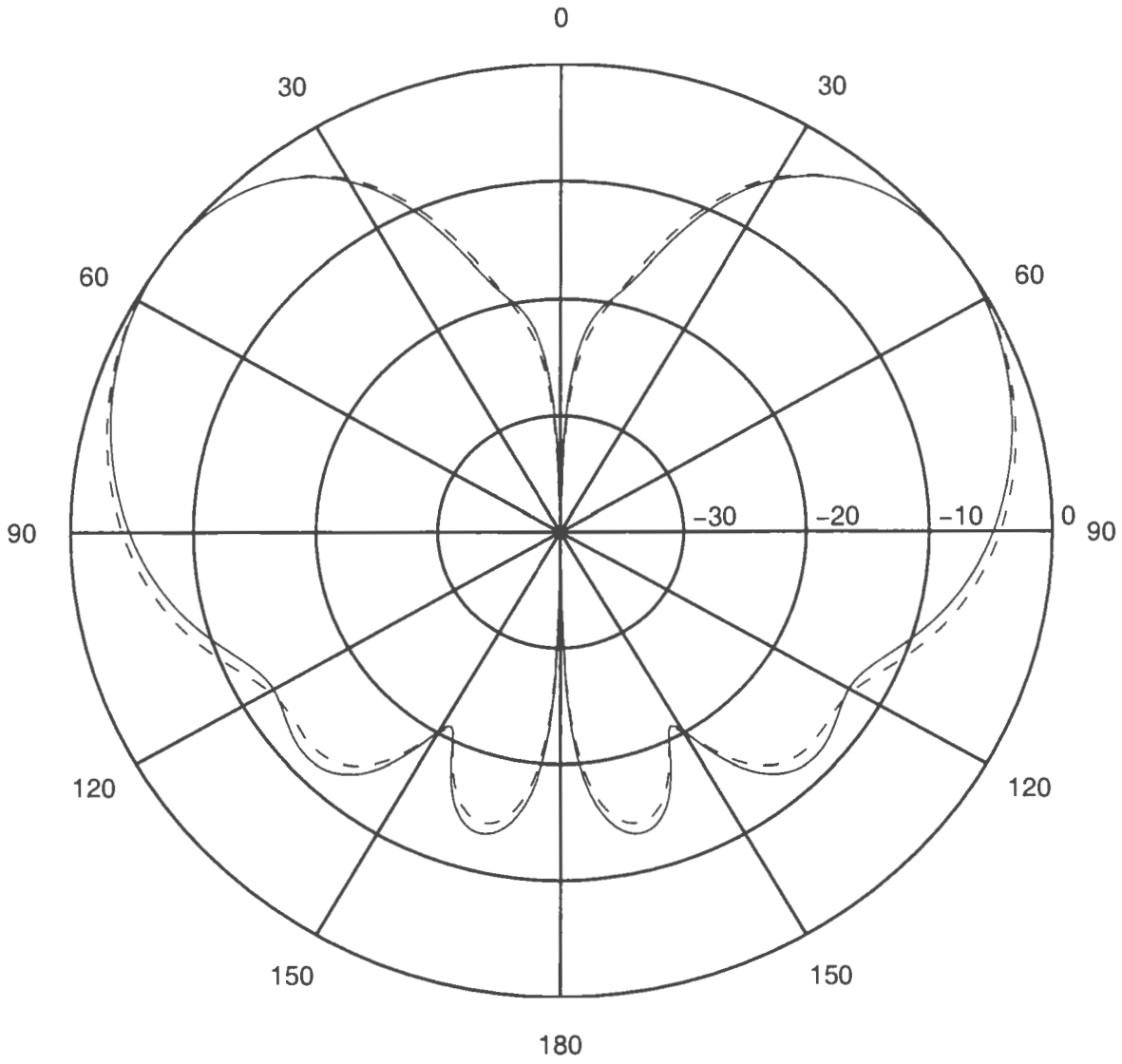


Figure 3.19: The radiation pattern of a monopole on $2\lambda \times 2\lambda$ ground

—— NEC-2; - - - FDTD.

Chapter 4

A Proposed Mobile Communication Antenna Design

In modern terrestrial-based wireless communications systems, the antenna for mobile radio telephone is a key element for system performance. The monopole and sleeve antennas have been the dominant radiating elements in mobile phones until now because of their wide bandwidth, omni-directional radiation pattern, and low cost. With higher requirement for small size, light weight, and low profile for modern mobile antennas, much research effort has turned to planar inverted-F antenna (PIFA) and microstrip antenna, which are low-profile and produce lower specific absorption rate(SAR) [3]. While the radiation patterns of PIFA and microstrip antennas satisfy the demand basically, much emphasis has been put on improving the impedance bandwidth recently [31]. Since the impedance bandwidth is not only limited by

antenna type itself but also sensitive to the radio case dimensions [14] as well as the existence of operators [3], it is particularly difficult to meet with the ever higher demands of modern communications.

As stated in [19], *"At present, the most widely adopted systems are the Global System for Mobile (GSM) Communications developed primarily in Europe and Asia and Interim Standard-54 (IS-54) developed in North America. The communication between the mobile station and base station is implemented through two links: uplink and downlink. The frequency bands for GSM are 890-915MHz and 935-960MHz for the uplink and downlink, respectively. While for IS-54 they are 869-894MHz for the uplink and 824-849MHz for the downlink. The new generation of personal communication services (PCS) such as DCS-1800 has frequency bands of 1.710-1.785GHz and 1.805-1.880 GHz for the uplink and downlink, respectively. The co-existence of GSM and DCS with a dual standard providing analogue and digital services in the same network means the corresponding antennas should have the capability of operating at dual frequency bands (824-960MHz and 1.71-1.88GHz)"*. Some efforts have already been put to the development of dual-frequency antennas [9, 10].

The planar monopole antenna has recently been proposed [11, 12], that has a very large impedance bandwidth. However, the application of this type of antenna to mobile communication has not been found in the literature yet. In this chapter, FDTD method is applied to obtain the characteristics of a planar monopole antenna, mounted on a conducting housing of limited size. A modified structure of planar

monopole is developed to provide dual-band operation while its radiation pattern is still approximately omni-directional in the horizontal plane.

4.1 A Planar Monopole on a Conducting Box

The input impedance of a planar monopole disc mounted on a large conducting plate was studied both numerically [11] and experimentally [12]. It was found that an elliptical planar monopole on a large conducting plane has very wide impedance bandwidth. However the performance of a planar monopole on a small conducting plate or box has not been reported. In order to apply this wide band antenna to mobile communications, the following model is set to study the antenna performance.

The geometry of the planar monopole mounted on a conducting box is shown in Figure 4.1. The conducting box is used to simulate a small hand-held portable telephone. The size of the box is $W \times L \times H$ in x, y, z direction as shown in Figure 4.1. The planar monopole antenna consists of an elliptic conducting disc with an extended short wire, which is connected to the central conductor of the feed on the top of the box. The length of the major and minor axes of the elliptical disc are $2a$ and $2b$ respectively. The parameters that affect the input impedance are the size of the monopole plate and the distance between the monopole plate and the conducting box, i.e. the height h of the extended wire and the box dimensions.

The box dimensions for our calculations are $2 \times 6 \times 10cm$ in x, y, z directions, respectively. The size of the elliptical planar monopole is $6 \times 4.8cm$. The return loss

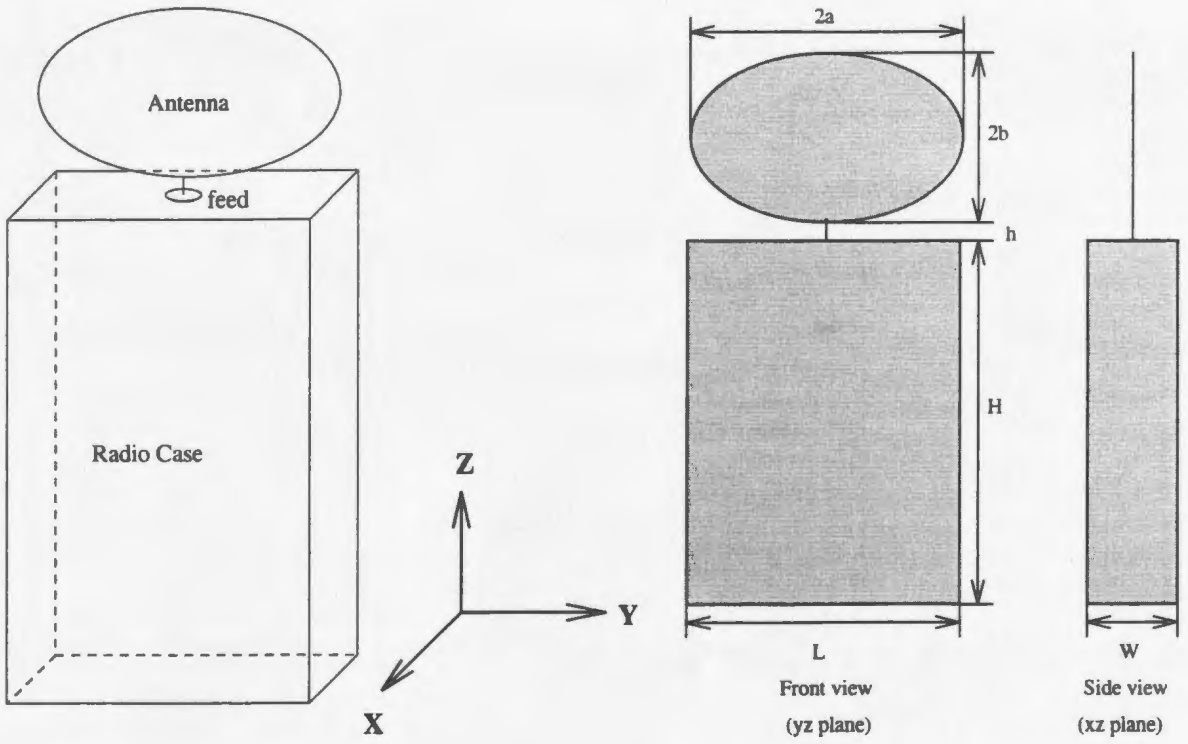


Figure 4.1: The geometry of planar monopole on a conducting box

of the planar monopole is shown in Figure 4.2

It is found that the frequency band of the antenna mounted on the small hand-held portable telephone is still very wide, though narrower than that on an infinite ground. However, the VSWR is greater than 2 when the frequency is lower than about 1.3GHz which means that the antenna cannot be use for existing analogue communication (around 900MHz) unless its size is increased or other measures are taken to create another resonant frequency around 900MHz. Obviously, the method of increasing the size the antenna should be avoided.

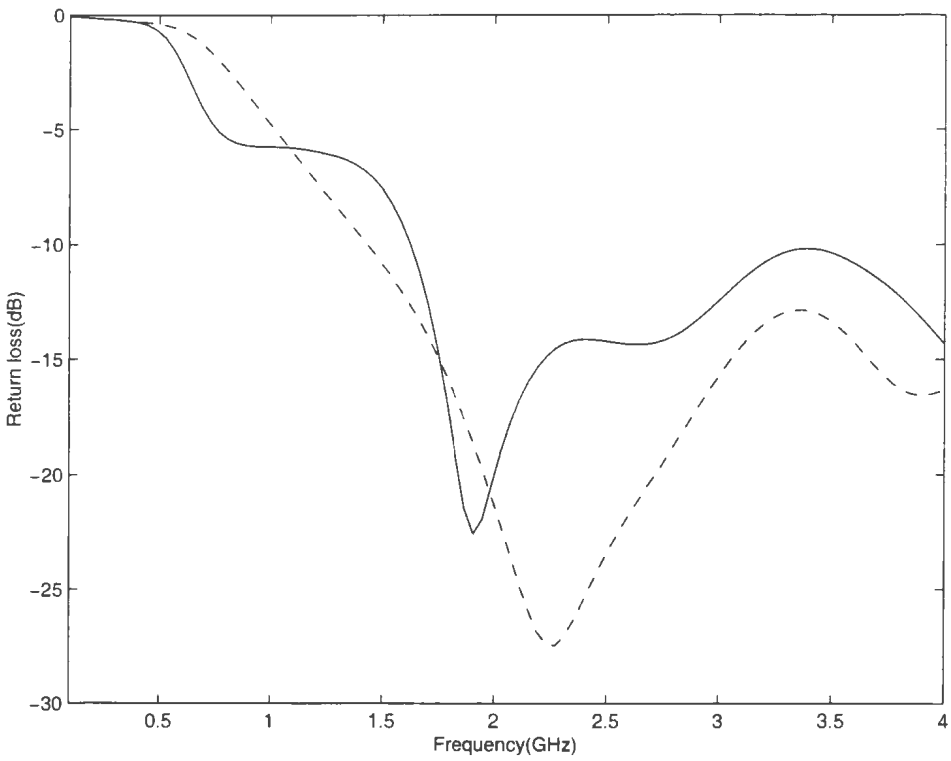


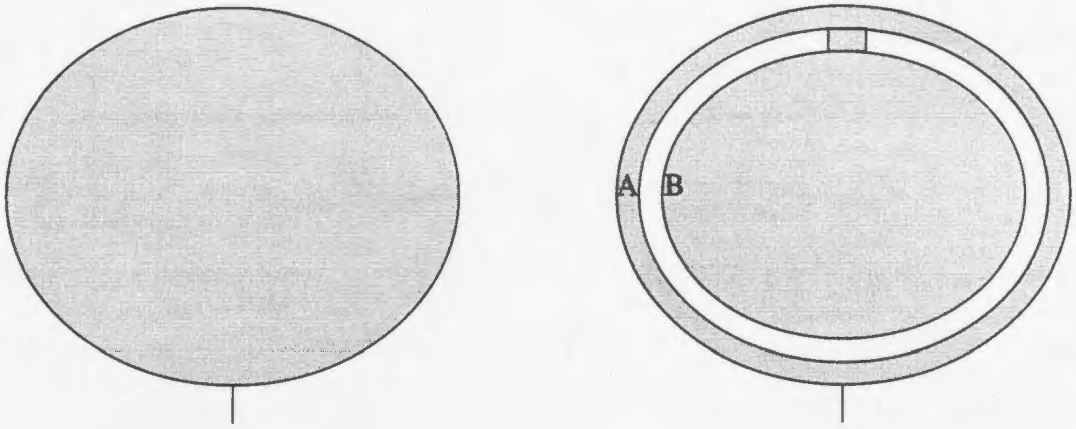
Figure 4.2: The return loss of a planar monopole mounted on ground and a box

——on a infinite ground; - - -on the box of finite size

4.2 The Implementation of Dual-Band Operation

The dual frequency operation was implemented by cutting an elliptical slot near the edge of the plate[23]. The antenna structure before and after the cutting is shown in Figure 4.3

The new antenna consists of two parts:



Original planar monopole

New dual-band antenna

Figure 4.3: The implementation of dual-band operation by cutting a slot

Part A is an elliptical ring. Its size is controlled by

$$\frac{(x-x_0)^2}{a^2}+\frac{(y-y_0)^2}{b^2}\leq 1 \tag{4.1}$$

$$\frac{(x-x_0)^2}{a_1^2}+\frac{(y-y_0)^2}{b_1^2}\geq 1 \tag{4.2}$$

Part B is a small elliptical disc with similar equations

$$\frac{(x-x_0)^2}{a_2^2}+\frac{(y-y_0)^2}{b_2^2}\leq 1 \tag{4.3}$$

When cutting the disc in this way, we actually changed the original disc into a planar monopole combined with a strip monopole. This can explain why the combined antenna operates at dual frequencies. While the sizes of the disc and the box have not changed, the width of the slot is about $4mm$ ($a1 = 2.6cm, a2 = 2.2cm, b1 = 2.0cm, b2 = 1.6cm$). Figure 4.4 shows the input impedance of the antenna for $h = 2mm$. The corresponding return loss at the input port is shown in Figure 4.5. One

resonant frequency is 800MHz and another resonant frequency is around 2100MHz. It is found that the reactance of the antenna around the resonant frequencies almost vanishes and the resistance approaches 50Ω, which is the characteristic impedance of the standard coaxial cable. Since very good impedance match is obtained, the complicated matching circuit can be avoided.

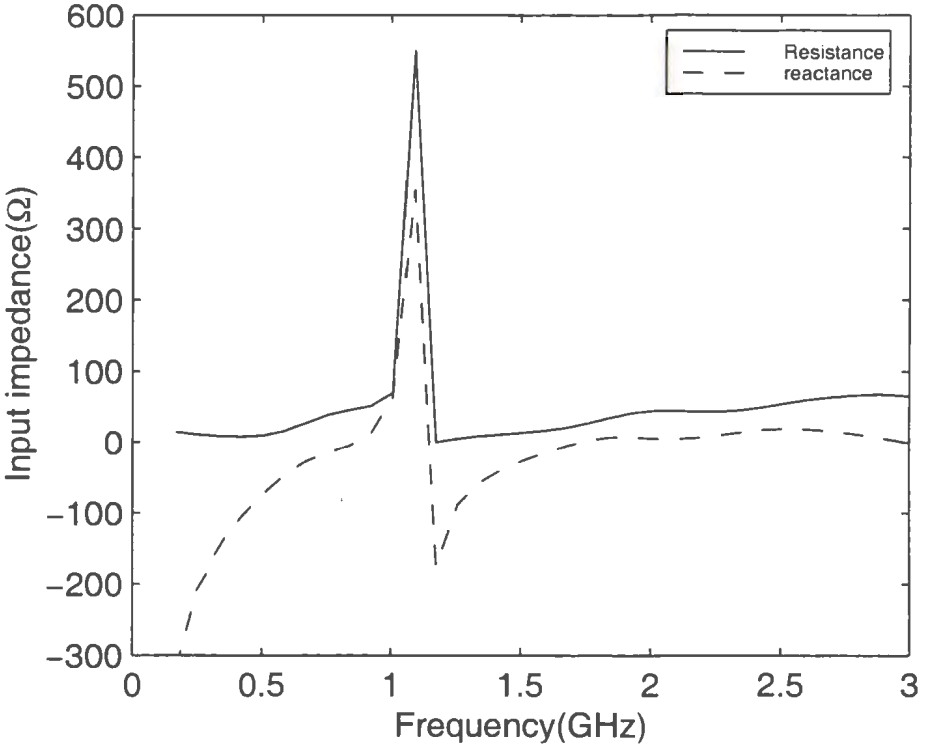


Figure 4.4: Input impedance of the antenna

Shown in Figure 4.5, the impedance bandwidth at 800MHz is about 30%. The bandwidth at 2100MHz is much wider.

Within each operating band, the radiation patterns do not change much. Figure 4.6 shows the radiation patterns of E_θ at 1900MHz. The solid line was obtained by

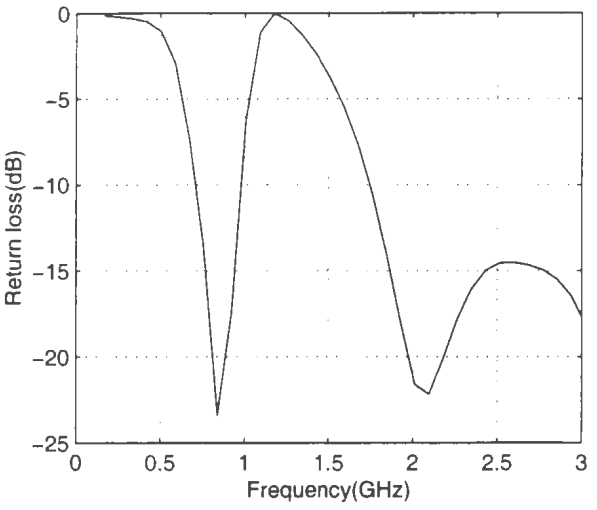


Figure 4.5: Return loss at the feed of the antenna

FDTD and the dash line was obtained by NEC-2.

4.3 Development of Controlling Two Specific Resonant Frequencies

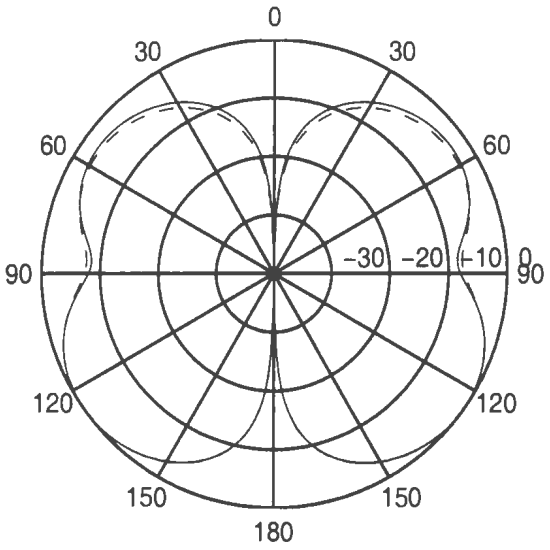
Instead of cutting an elliptical slot near the edge of the plate, one can cut the disc in the way [24] shown in Figure 4.7.

The new antenna consists of three parts:

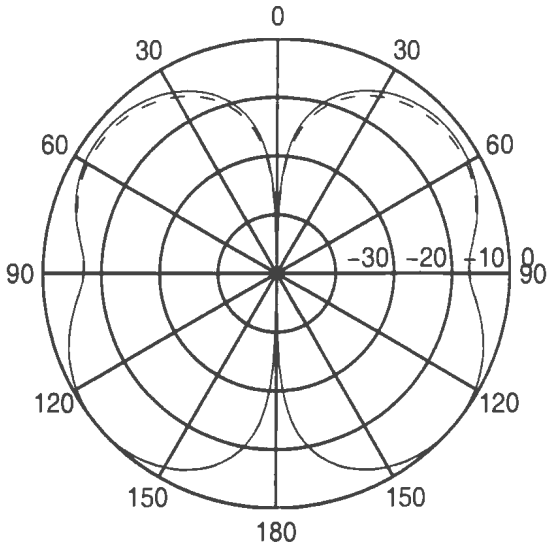
Part A is a big elliptical ring. Its size is controlled by

$$\frac{(x-x_0)^2}{a^2} + \frac{(y-y_0)^2}{b^2} \leq 1 \tag{4.4}$$

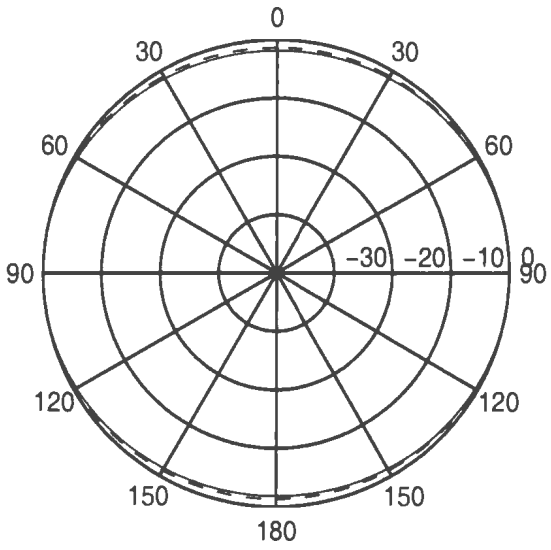
$$\frac{(x-x_0)^2}{a_1^2} + \frac{(y-y_0)^2}{b_1^2} \geq 1 \tag{4.5}$$



(a) x-z plane



(b) y-z plane



(c) x-y plane

Figure 4.6: Radiation pattern of the antenna

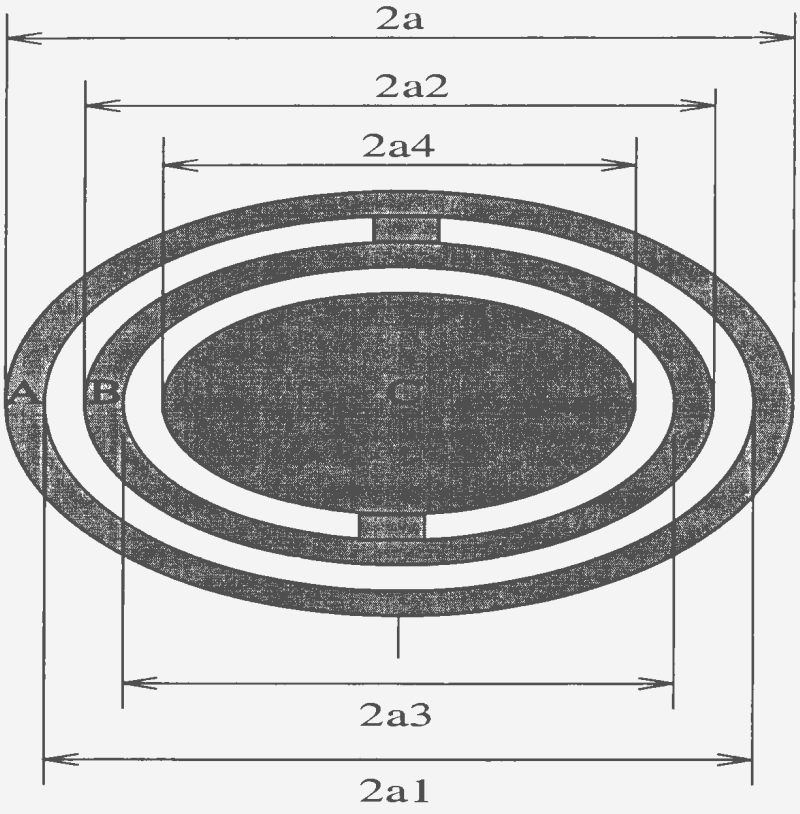


Figure 4.7: Geometry of the new planar monopole

Part B is a small elliptical ring with similar equations

$$\frac{(x-x_0)^2}{a_2^2}+\frac{(y-y_0)^2}{b_2^2}\leq 1 \tag{4.6}$$

$$\frac{(x-x_0)^2}{a_3^2}+\frac{(y-y_0)^2}{b_3^2}\geq 1 \tag{4.7}$$

Part C is an elliptical disc with an equation

$$\frac{(x-x_0)^2}{a_4^2}+\frac{(y-y_0)^2}{b_4^2}\leq 1 \tag{4.8}$$

The three parts are connected together in series, as shown in Figure 4.7. The purpose to do this is to lower the second resonant frequency without increasing the size

of the antenna. When the resonant frequency is adjusted to the operating frequency, the antenna performance will be enhanced.

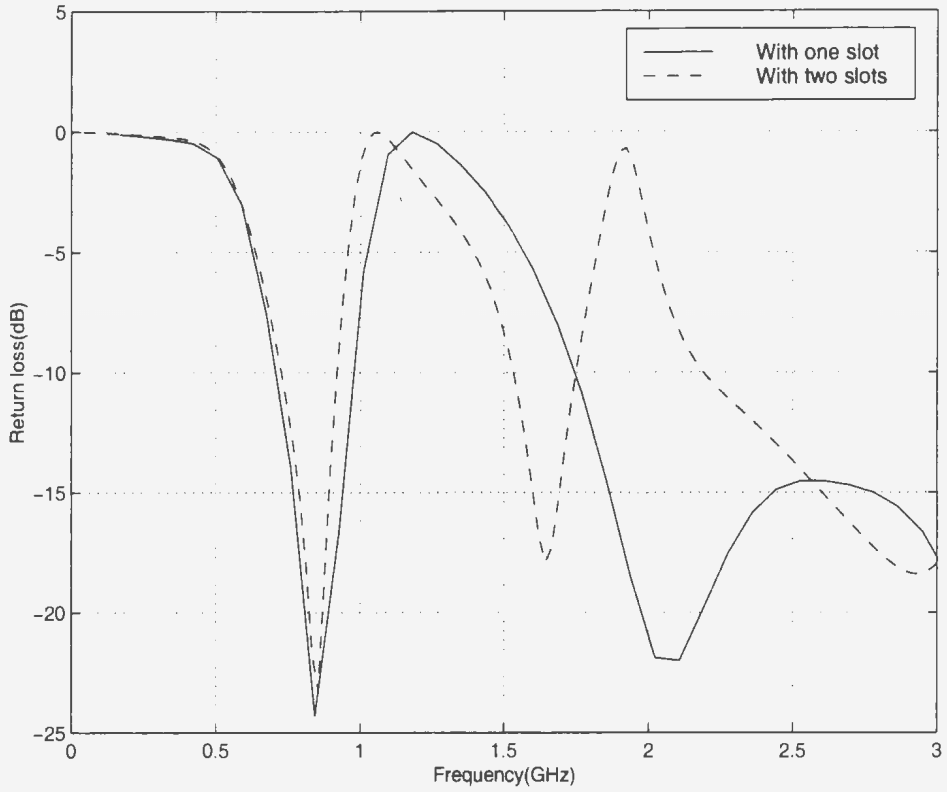


Figure 4.8: Return loss of the antenna with and without the second slot

The box dimensions are still $2 \times 6 \times 10cm$ in x, y, z directions, respectively. The size of the original elliptical planar monopole is $6 \times 4.8cm$ ($a = 3cm$, $b = 2.4cm$). The height of the extension wire is $h = 2mm$. In the rest of chapter, the unit for the dimensions of the disc will be in centimeters.

Figure 4.8 shows the effect of the second cutting on the return loss of the antenna. The parameters for different configurations are shown in Table 1. For the new cutting, $a_3 = 1.8$, $b_3 = 1.2$, $a_4 = 1.4$, $b_4 = 0.8$. The other parameters have not changed, i.e.

a, b, a_1, b_1, a_2, b_2 are the same as in previous shape. It is found that the resonant frequency at 845MHz has not changed. But the higher resonant frequency is lowered about 450MHz. The new resonant frequency is 1.65GHz, which is lower than the DCS operation frequency.

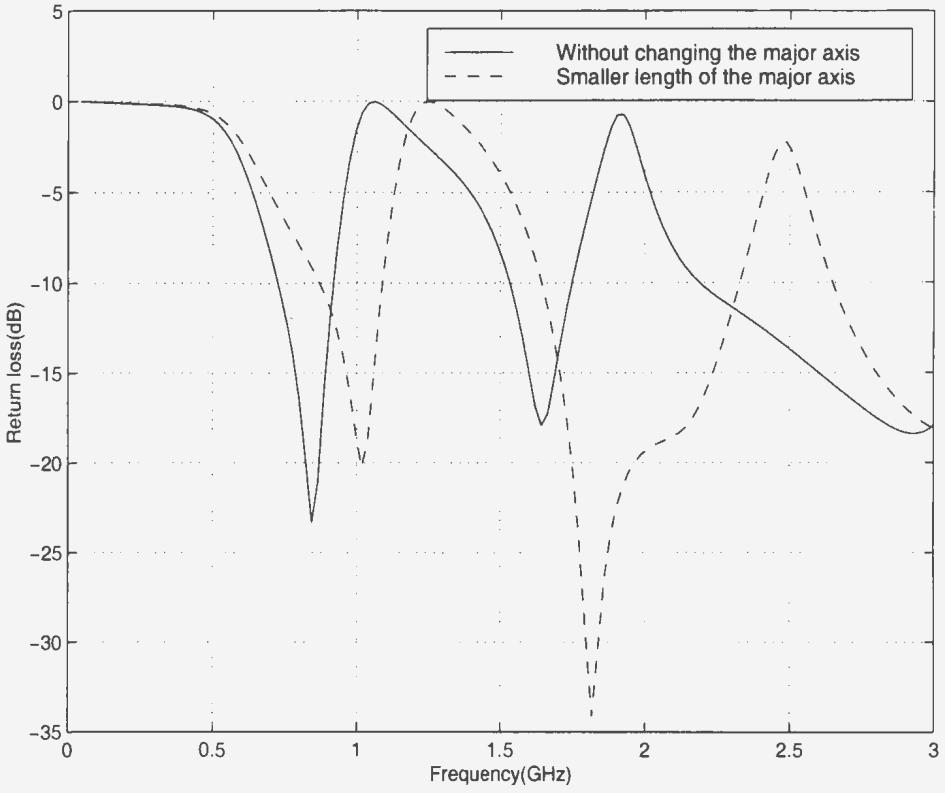


Figure 4.9: Return loss of the antenna for different major axis length

Next, one can reduce the major axis of the elliptical disc with two slots. The return loss is shown in Figure 4.9, where the dash line represents $a = 2.4$, $a_1 = 2.0$, $a_2 = 1.6$, $a_3 = 1.2$, $a_4 = 0.8$, with b_i(i=1, 2, 3, 4) unchanged. It is found that reduction of the axis let the resonant frequencies shift toward higher frequency. Therefore, the resonant frequency at 845MHz can be adjusted by changing the length of major axis.

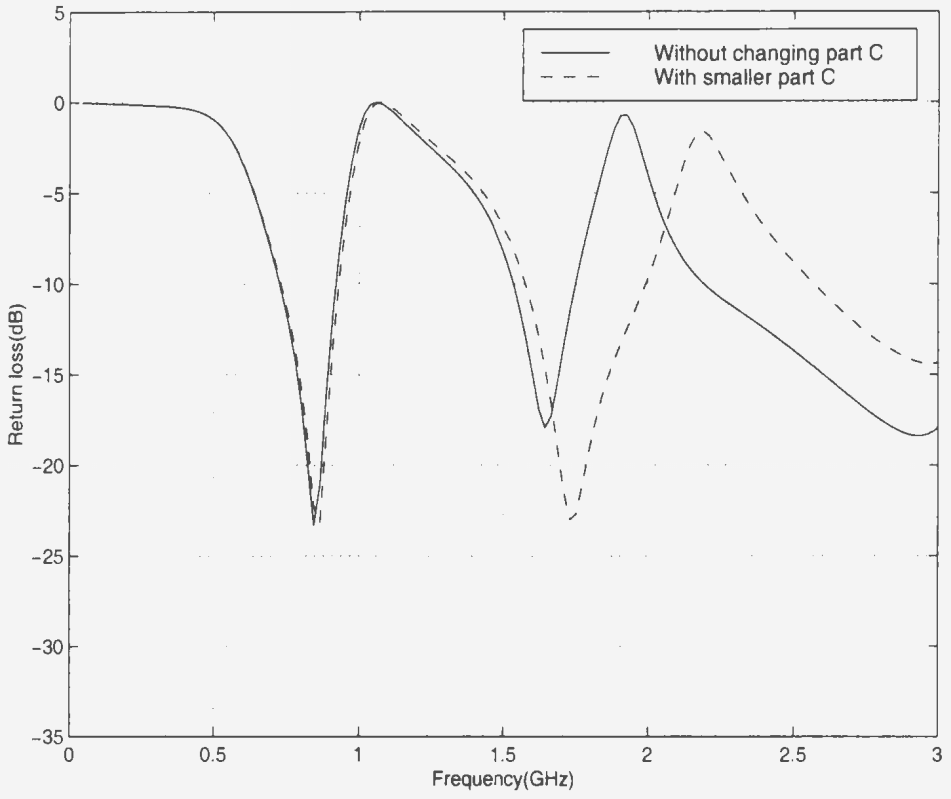


Figure 4.10: Return loss of the antenna for different size of the second slot

Then one can change the size of the small disc (part C). From Table 1, one can see $a_4 = 1.0$, $b_4 = 0.6$. The rest of the parameters are not changed. The resultant return loss is shown in Figure 4.10. From the figure one can see that the resonant frequency at 845MHz is not affected while the higher resonant frequency can be adjusted.

The return loss of the antenna covered with lossless dielectric material is shown in Figure 4.11. The conducting box is not covered by dielectric material. The cover of the antenna has the same height ($5cm$) and width($6cm$) as the antenna. For the thin cover, the thickness is $0.6cm$, while for the think one, the thickness is $1.0cm$. The parameters for the material are $\epsilon_r = 2.1$, $\mu = \mu_0$, The resonant frequencies are shifted

Table 1: The effect of the parameters of the disc with slots to the resonant frequencies

action	parameters controlling the size of the disc (cm)										resonant frequencies (MHz)	
	a	a_1	a_2	a_3	a_4	b	b_1	b_2	b_3	b_4	the first	the second
original (one slot)	3.0	2.6	2.2	-	-	2.4	2.0	1.6	-	-	845	2100
adding a slot	3.0	2.6	2.2	1.8	1.4	2.4	2.0	1.6	1.2	0.8	845	1650
reducing major axis length	2.4	2.0	1.6	1.2	0.8	2.4	2.0	1.6	1.2	0.8	1020	1850
changing part C	3.0	2.6	2.2	1.8	1.0	2.4	2.0	1.6	1.2	0.6	845	1740

downwards. The thicker the dielectric material, the more the resonant frequencies are shifted down. It is also found that the impedance bandwidth becomes much narrower when the antenna is covered with dielectric material. It is suggested that the dielectric cover should not be very thick in order to satisfy the bandwidth requirement.

From above numerical results, we can summarize our design procedure as follows:

1. change the size of the original disc so that the first resonant frequency is 900MHz(GSM operation frequency);

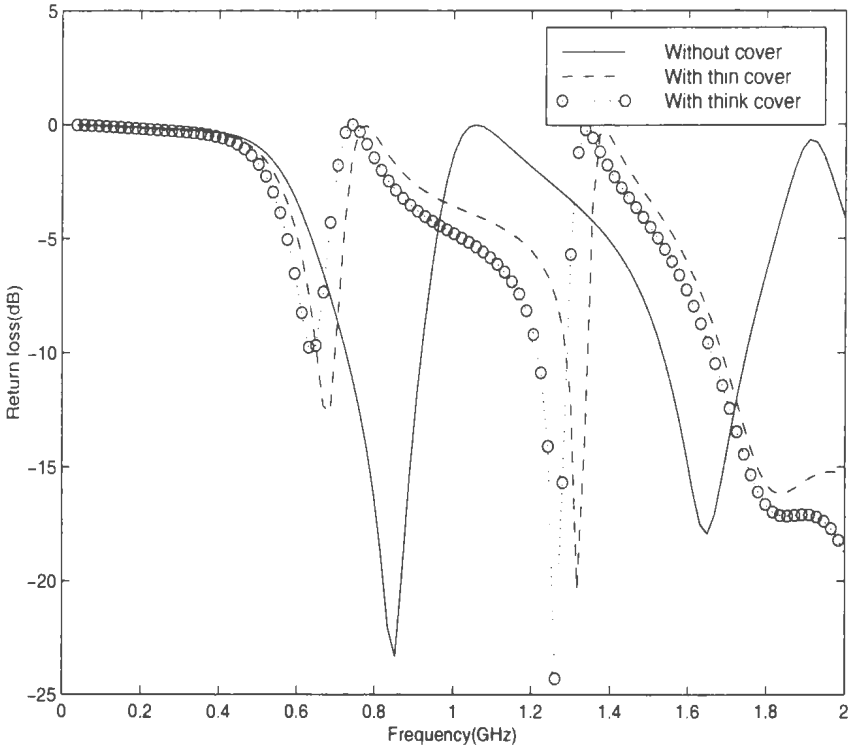


Figure 4.11: Return loss of the antenna with dielectric cover

2. change the size of part C (inner most part) so that the second resonant frequency is 1800MHz(DCS operation frequency);

3. change the major axis so that the first resonant frequency is the exact frequency expected;

4. repeat step 2 to control the second frequency.

In practical design, the dielectric constant should be taken into account before beginning the above steps.

In the above procedure, no obvious changes have been observed for the antenna

radiation patterns.

Chapter 5

Conclusion and Future Work

A FDTD code is developed for mobile antenna design purposes, with particular emphasis on the modeling of the source region. The feed forms of the source region include a thin gap voltage source and a current source, the latter can be integrated directly into the Maxwell's curl equation. For input impedance calculation, the Gaussian pulse or the Rayleigh pulse is chosen to be the function of the sources. It is found that whatever is the feed form (voltage source or current source), Rayleigh pulse always produces good results. The results from Gaussian pulse is dependent on the antenna type. If the antenna is dipole type (it is a open circuit as the operating frequency is low), the impedance results are good provided that the feed form is voltage source. The current source of Gaussian pulse will result in negative resistance (the real part of the input impedance). If the antenna is loop type (it is short circuit as the operating frequency is low), current source produces good impedance results,

while the results from a voltage source are not acceptable.

Several typical antenna structures including a dipole antenna, a loop antenna and a monopole mounted on a finite conducting plane are used to validate the developed FDTD code. The effects of the FDTD cell size and the Mur distance (the distance between the outer absorbing boundary and the antenna) on accuracy are explored. Some criteria for the choice of above parameters in FDTD calculation are given for practical antenna design. The results from the developed code are compared with those producing by a moment method based code *Numerical Electromagnetic Code* (NEC-2) and very good agreement is obtained when the criteria is satisfied.

Using the developed code, a new planar monopole antenna which operates at dual wide band (800MHz band and 1800MHz band) is developed by cutting slots and determining the geometrical parameters. This dual frequency performance is required for the existing and potential mobile communication system providing analogue and digital services.

The absorbing boundary condition used in this thesis is Mur boundary condition, which requires that the distance between the antenna and the computational boundary is more than 10 cells for radiation pattern calculation and 30 cells for input impedance calculation. Since the computational cost is directly related to the computational domain, the reduction of the cells between the antenna and the boundary will lower the computational cost greatly. Some other absorbing boundary conditions, which can simulate the free wave propagation in a better way, are needed.

Bibliography

[1] R. A. Burberry. *VHF and UHF Antennas.* Peter Peregrinus Ltd.. London. United Knigdom. 1992.

[2] P. A. Tirkas and C. A. Balanis, "Finite-difference time-domain method for antenna radiation." *IEEE Trans. on Ant. Prop.*, vol. 40, pp. 334–340. March 1992.

[3] M. A. Jensen and Y. Rahmat-Samii, "EM interaction of handset antennas and a human in personal communications," *Proc of IEEE*, vol. 83, pp. 7–17. January 1995.

[4] R. Luebbers, L. Chen. T. Uno, and S. Adachi. "FDTD calculation of radiation patterns, impedance, and gain for a monopole antenna on a conducting box." *IEEE Trans. Ant. Prop.*, vol. 40, pp. 1577–1583. December 1992.

[5] C. A. Balans, *Antenna Theory: Analysis and Design.* John Wiley & Sons, Inc.. 1982.

[6] K. Fujimoto and I. R. James, *Mobile Antenna System Handbook.* Artech House, Inc., 1994.

[7] S. Ramo, J. R. Whinnery, and T. V. Duzer. *Fields and Waves in Communication Electronics.* John Wiley & Sons Inc., 1994.

[8] C. R. Rowell and R. D. Murch. "A capacitively loaded PIFA for compact mobile telephone handsets." *IEEE Trans. Ant. Prop.*, vol. 45, pp. 837–842, May 1997.

[9] W. S. Chen. "Single-feed dual-frequency rectangular microstrip antenna with square slot." *Electronics Letters*, vol. 34, pp. 231–232, February 1998.

[10] A. Serrano-Vaello and D. Sanchez-Hernandez, "Printed antennas for dual-band GSM/DCS 1800 mobile handsets." *Electronics Letters*, vol. 34, pp. 140–141, January 1998.

[11] N. P. Agrawall, G. Kumar, and K. P. Ray, "A wide-band planar monopole antennas." *IEEE Trans. Ant. Prop.*, vol. 46, pp. 294–295, February 1998.

[12] M. Hammoud, P. Poey, and F. Colombel, "Matching the input impedance of a broadband disc monopole." *Electronics Letters*, vol. 29, pp. 406–407, February 1993.

[13] C. J. Burke and A. J. Poggio, *Numerical Electromagnetics Code (NEC): User's Manual.* Lawrence Livermore Nat. Lab., Livermore, CA, Jan., 1981.

[14] T. Taga and K. Tsunekawa, "Performance analysis of a built-in planar inverted F antenna for 800 mhz band portable radio units," *IEEE Journal on selected areas in communications*, vol. 5, pp. 921–929, June 1987.

[15] K. Sato, K. Matsumoto, K. Fujimoto, and K. Hirasawa, "Characteristics of a planar inverted F antenna on a rectangular conducting body," *Electronics and Communications in Japan*, vol. 72, no. 10, pp. 43–51, 1989.

[16] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's eqations in isotropic media," *IEEE Trans. Ant. Prop.*, vol. 14, p. 302, March 1966.

[17] G. Mur, "Absorbing boundary conditions for the finite-difference approximation of the time-domain electromagnetic-field equations," *IEEE Trans. Ant. Prop.*, vol. 39, pp. 429–433, April 1991.

[18] Z. P. Liao, H. L. Wong, and Y. F. Yuan, "A transmitting boundary for the transient wave analysis," *Scince Sinica*, vol. 27, no. 10, pp. 1063–1076, 1984.

[19] K. D. Katsibas, C. A. Balanis, P. A. Tirkas, and C. R. Birtcher, "Folded loop antenna for mobile hand-held units," *IEEE Trans. on Ant. Prop.*, vol. 46, pp. 260–266, February 1998.

[20] J. G. Maloney, G. S. Smith, and W. R. Scott, "Accurate computation of the radiation from simple antennas using the finite-difference time-domain method," *IEEE Trans. on Ant. Prop.*, vol. 38, pp. 1059–1068, July 1990.

[21] J. J. Boonzaaier and C. W. I. Pistorius, "Thin wire dipoles—a finite-difference time-domain approach," *Electronics Letters*, vol. 26, pp. 1891–1892, October 1990.

[22] K. S. Kunz and R. J. Luebbers. *The Finite Difference Time domain Method for Electromagnetics.* CRC Press. Inc.. 1993.

[23] M. Qiu. B. P. Sinha. and S. A. Saoudy. "A dual-band planar monopole antenna for mobile handsets." *10th International Symposium on Antennas.* November 17-19. 1998. Nice Acropolis. France.

[24] M. Qiu. B. P. Sinha. and S. A. Saoudy. "Performance optimization of planar monopole antenna for mobile telephone handsets," *1998 Asia-Pacific Microwave Conference.* December 8-11, 1998, Pacifico Yokohama, Yokohama, Japan.

[25] A. Taflove and M. E. Brodwin, "Numerical solution of steady-state electromagnetic scattering problems using the time-dependent Maxwell's equations," *IEEE Trans. on MTT.* vol. 23. pp. 623–630. August 1975.

[26] J. P. Berenger, "Improved PML for the FDTD solution of wave-structure interaction problems," *IEEE Trans. Ant. Prop.*, vol. 45, pp. 466–473, March 1997.

[27] R. J. Luebbers, K. S. Kunz, M. Schneider, and F. Hunsberger, "A finite-difference time-domain near zone to far zone transformation," *IEEE Trans. Ant. Prop.*, vol. 39, pp. 429–433, April 1991.

[28] O. M. Ramahi, "Near- and far-field calculation in FDTD simulations using kirchhoff surface integral representation," *IEEE Trans. Ant. Prop.*, vol. 45, pp. 753–759, May 1997.

[29] A. Taflove, K. R. Umashankar, B. Beker, F. Harfoush, and K. S. Yee, "Detailed FD-TD analysis of electromagnetic fields penetrating narrow slots and lapped joints in thick conducting screens," *IEEE Trans. on Ant. Prop.*, vol. 36, pp. 247–257, Febuary 1988.

[30] M. A. Jensen and Y. Rahmat-Samii, "Performance analysis of antennas for hand-held transivers using FDTD," *IEEE Trans. on Ant. Prop.*, vol. 42, pp. 260??–266, August 1994.

[31] K. L. Wong and Y. F. Lin, "Small broadband rectangular microstrip antenna with chip-resistor loading," *Electronics Letters*, vol. 33, pp. 1593–1594, February 1997.

Appendix A

A Input Data File Example

A.1 The Data File "ant.dat"

```
41 61 62
0.00375 0.00375 0.003571429
0.0005
1.0   32.0   0.1499
90000
material.dat
0
19 21 19 43 19 44
.010
21.0 30.5 31.0
4.0

!!!!!!!!!!!
line 1  nx ny nz
line 2  dx dy dz
line 3  wire radii
line 4  amp beta  wavelength
line 5  nstop
line 6  material file name
line 7  flag:A
            A :0>pulse; 1>sine wave  (the form of the source);
line 8  Ist Ind Jst Jnd Kst Knd; (integral surface size)
line 9  value_ERROR : condition to judge the stability
line 10 the origin for the far field
line 11 dielectric constant of the surrouding material
```

A.2 The Data File "wire.dat"

```
4
z 21 21 22 40 31
z 21 41 22 40  2
y 21 22 39 21  2
y 21 22 39 42  2

/**line 1 number of wires(single wire:1; two wires:2.......)
/**line 2 first wire: Ic Jc Kst Kend(starting point & ending point)
/**line 3 second wire:Ic Jc Kst Kend(starting point & ending point)
/**.....................................
```

A.3 A Program to Create "material.dat'

```
#include <iostream.h>
#include <math.h>
#include<stdio.h>
//*****************
//***material id***
//*****************
const int  LL= 21;
const int  L1= 43;
const int  L2= 57;
const int nx0=70, ny0=70, nz0=190;

int idone[nx0][ny0][nz0], idtwo[nx0][ny0][nz0], idthre[nx0][ny0][nz0];
main()
{int i,j,k;
int nx=51, ny=62, nz=182;
int mtype=1;

for( i=1; i<=nx;i++)
   for( j=1; j<=ny;j++)
       for( k=1; k<=nz;k++)
          {
          idone[i][j][k]=0;idtwo[i][j][k]=0; idthre[i][j][k]=0;
          if(i==21 && (j==21||j==41) && k>=21 && k<=41) idthre[i][j][k]=1;
          if(i==21 && (k==21||k==42) && j>=21 && j<=40) idtwo[i][j][k]=1;
          }

idthre[21][21][31]=-1;
```

```
FILE * fp;
fp= fopen("material.dat", "w");
for( i=1; i<=nx;i++)
   for( j=1; j<=ny;j++)
       for(k=1; k<=nz;k++)
           if(idone[i][j][k]!=0 || idtwo[i][j][k]!=0 || idthre[i][j][k]!=0)
           fprintf(fp, "%5i %5i %5i %5i %5i %5i\n", i,j,k, idone[i][j][k],\
                  idtwo[i][j][k], idthre[i][j][k]);
fclose(fp);
}
```

Appendix B

The FDTD Code

```
*************defin.h***************

#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <ctype.h>
#include "complex.h"
#define nx0 42
#define ny0 42
#define nz0 73
typedef double dou;
enum Mybool{false0,true1};
struct nxyz{int x,y,z;};
struct del0{dou x,y,z;};
struct dis0{dou x,y,z;};
struct coff0{dou ecrlx[10], ecrly[10], ecrlz[10],
                 esctc[10], eincc[10], edevcn[10];};
************************************

************antm.cpp***************

#include "defin.h"
coff0 coff;
del0 del;
dis0 disp;
nxyz nn, n1;
```

```
void init() ;
void geo(char *);
void par(int, dou, dou);
void exfld(), eyfld(), ezfld() ;
void hxfld(), hyfld(), hzfld() ;
void radeyx(), radezx();
void radezy(), radexy();
void radexz(), radeyz();
void fine_wire();
void wave_stable(double , int *, int *);
void find_max();
void find_phase( double );
void out_surface_field( double );
void out_surface_flow( double );
void current(int, fstream &);
void save_diag(int, dou,dou,dou,dou,dou,dou);
//void datsave(del0 , dou , int , int , int );
void far_field(int, int , char *);
void readFile(char *, fstream &);
void writeFile(char *, fstream &);

dou eps0=8.854e-12,xmu0=1.2566306e-6,eta0=376.733341;
dou C=1.0/sqrt(eps0*xmu0);
dou PI=4*atan(1.0);
const unsigned NAME_SIZE=64;

int idone[nx0][ny0][nz0],idtwo[nx0][ny0][nz0],idthre[nx0][ny0][nz0];
dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
    hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
dou eysx1[5][ny0][nz0], ezsx1[5][ny0][nz0],
    ezsy1[nx0][5][nz0], exsy1[nx0][5][nz0],
    exsz1[nx0][ny0][5], eysz1[nx0][ny0][5];  //radsav
dou eysx2[5][ny0][nz0], ezsx2[5][ny0][nz0],
    ezsy2[nx0][5][nz0], exsy2[nx0][5][nz0],
    exsz2[nx0][ny0][5], eysz2[nx0][ny0][5];  //radsav2

//*******************
dou ex_y[2][nx0][ny0],hx_y[4][nx0][ny0],
    ex_z[2][nx0][nz0],hx_z[4][nx0][nz0],

    ey_x[2][nx0][ny0],hy_x[4][nx0][ny0],
```

```
    ey_z[2][ny0][nz0],hy_z[4][ny0][nz0],

    ez_x[2][nx0][nz0],hz_x[4][nx0][nz0],
    ez_y[2][ny0][nz0],hz_y[4][ny0][nz0];   //surface_current

complex  ex_y_p[2][nx0][ny0],hx_y_p[4][nx0][ny0],
         ex_z_p[2][nx0][nz0],hx_z_p[4][nx0][nz0],

         ey_x_p[2][nx0][ny0],hy_x_p[4][nx0][ny0],
         ey_z_p[2][ny0][nz0],hy_z_p[4][ny0][nz0],

         ez_x_p[2][nx0][nz0],hz_x_p[4][nx0][nz0],
         ez_y_p[2][ny0][nz0],hz_y_p[4][ny0][nz0];   //surface_current_phase

complex exy[2][nx0][ny0],hxy[2][nx0][ny0],
        exz[2][nx0][nz0],hxz[2][nx0][nz0],

        eyx[2][nx0][ny0],hyx[2][nx0][ny0],
        eyz[2][ny0][nz0],hyz[2][ny0][nz0],

        ezx[2][nx0][nz0],hzx[2][nx0][nz0],
        ezy[2][ny0][nz0],hzy[2][ny0][nz0];   //surface_current
//********************



dou dtedx,dtedy,dtedz;
dou dtmdx,dtmdy,dtmdz;
dou cxd,cxu,cyd,cyu,czd,czu,      //mur 1
    cxx,cyy,czz,cxfyd,cxfzd,cyfxd,cyfzd,czfxd,czfyd; //mur2

dou alpha, betadt, period, off;
dou delay, ampx, ampy, ampz;

dou t,dt;
dou wl;
dou r0;

int flag_inc, flag_stable=0, flag_max=0, flag_phase=0;
int Ist, Ind, Jst, Jnd, Kst, Knd;
double value_ERROR;
double dielc;
```

```
double cen_x, cen_y, cen_z, fi;

//*****************
//***management***
//*****************
double SOURCE_OLD=0.0, SOURCE_NEW=0.0;
main()
{
int nstop;
dou thinc=0, phinc=0, ethinc=0, ephinc=0;
dou amp, beta;
int ntest=4;
char material_File[NAME_SIZE+1];

fstream fin ;
char inFile[NAME_SIZE+1]="ant.dat";
readFile(inFile, fin);
fin>>nn.x>>nn.y>>nn.z;
fin>>del.x>>del.y>>del.z;
fin>>r0;
fin>>amp>>beta>>wl;
fin >>nstop;
fin>>material_File;
fin>>flag_inc;
fin>>Ist >>Ind >>Jst >>Jnd >>Kst >>Knd;
fin>>value_ERROR;
fin>>cen_x>>cen_y>>cen_z;
fin>>dielc;
fi=90;
fin.close();

n1.x=nn.x-1; n1.y=nn.y-1; n1.z=nn.z-1;

cout<<"nstop="<<nstop<<"\n";

//_______________________________operation
cout << "qmd";
init();
par(nstop,amp, beta);
geo(material_File);

int step_add=-1;  // step -1: initialize; step 0 : begin
```

```
double ONE_cycle=wl/C/dt;

cout<<"cycle="<<ONE_cycle<<"\n";
for(int n0=1; n0<=nstop; n0++)
{ cout<<n0<< endl;
exfld(); eyfld(); ezfld();

radeyx();
radezx();
radezy();
radexy();
radexz();
radeyz();

t=t+dt/2.;
hxfld(); hyfld(); hzfld();

fine_wire();
t=t+dt/2.;

//************************************************
if(flag_inc==1)
  {
   if(flag_stable!=1)
      wave_stable(hxs[nn.x-10][nn.y-10][nn.z-10], &flag_stable, &n0) ;
   else
     {find_max();
      double _tn=step_add*(360/ONE_cycle);
      find_phase(_tn);
      if( (step_add++)==(int)(ONE_cycle+1) || n0==nstop)
        {out_surface_flow(ONE_cycle);
         out_surface_field(ONE_cycle);
         far_field(0,1, "Ectx.dat");
         far_field(90,1, "Ecty.dat");
         far_field(90,0, "Ectf.dat");
         break;
        }
     }
  }//** sin_inc wave , steady response.**
//************************************************

}  // for-loop  ends*******
```

```
return 0;
}// main end

//*****************
//***material id***
//*****************
void geo(char *inFile)
{
int i, j, k, id_x, id_y, id_z;
char c_tmp[160];
fstream fin ;
readFile(inFile, fin);
while( !fin.eof() )
  {fin>>c_tmp; if( !isdigit(c_tmp[0]) ) break;
   i=atoi(c_tmp);  fin>>j>>k>>id_x>>id_y>>id_z;
   idone[i][j][k]=id_x; idtwo[i][j][k]=id_y; idthre[i][j][k]=id_z;
   }
fin.close();
}
**********************************

**********init.cpp*************

#include "defin.h"
//**********************
//***initial value 0***
//**********************

extern int idone[nx0][ny0][nz0],idtwo[nx0][ny0][nz0],
           idthre[nx0][ny0][nz0];
extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],
           ezs[nx0][ny0][nz0],hxs[nx0][ny0][nz0],
           hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
extern dou eysx1[5][ny0][nz0], ezsx1[5][ny0][nz0],
           ezsy1[nx0][5][nz0], exsy1[nx0][5][nz0],
           exsz1[nx0][ny0][5], eysz1[nx0][ny0][5];  //radsav
extern dou eysx2[5][ny0][nz0], ezsx2[5][ny0][nz0],
           ezsy2[nx0][5][nz0], exsy2[nx0][5][nz0],
           exsz2[nx0][ny0][5], eysz2[nx0][ny0][5];  //radsav2
//******************
extern dou ex_y[2][nx0][ny0],hx_y[2][nx0][ny0],
```

```
            ex_z[2][nx0][nz0],hx_z[2][nx0][nz0],

            ey_x[2][nx0][ny0],hy_x[2][nx0][ny0],
            ey_z[2][ny0][nz0],hy_z[2][ny0][nz0],

            ez_x[2][nx0][nz0],hz_x[2][nx0][nz0],
            ez_y[2][ny0][nz0],hz_y[2][ny0][nz0];
//surface_current
//*******************

extern nxyz nn,n1;
extern coff0 coff;

void init()
{
int i,j,k,l;
for( k=1; k<=nn.z; k++)
   for( j=1; j<=nn.y; j++)
       for( i=1; i<=nn.x; i++)
   {exs[i][j][k]=0.0;  eys[i][j][k]=0.0;
            ezs[i][j][k]=0.0;  hxs[i][j][k]=0.0;
            hys[i][j][k]=0.0;  hzs[i][j][k]=0.0;
    idone[i][j][k]=0;  idtwo[i][j][k]=0;
            idthre[i][j][k]=0;
    }
for(k=1; k<=n1.z; k++)
   for( j=1; j<=n1.y; j++)
       for( i=1; i<=4; i++)
   {eysx1[i][j][k]=0.0; eysx2[i][j][k]=0.0;
    ezsx1[i][j][k]=0.0; ezsx2[i][j][k]=0.0;
   }
for(k=1; k<=n1.z; k++)
   for( j=1; j<=4; j++)
       for( i=1; i<=n1.x; i++)
   {exsy1[i][j][k]=0.0;  exsy2[i][j][k]=0.0;
    ezsy1[i][j][k]=0.0;  ezsy2[i][j][k]=0.0;
   }
for(k=1; k<=4; k++)
   for( j=1; j<=n1.y; j++)
       for( i=1; i<=n1.x; i++)
   {exsz1[i][j][k]=0.0;  exsz2[i][j][k]=0.0;
    eysz1[i][j][k]=0.0;  eysz2[i][j][k]=0.0;
```

```
   }
for( l=1; l<=9; l++)
   { coff.esctc[l]=0.0; coff.eincc[l]=0.0; coff.edevcn[l]=0.0;
     coff.ecrlx[l]=0.0; coff.ecrly[l]=0.0; coff.ecrlz[l]=0.0 ;
   }
//*************************
for( int ind=0; ind<=1; ind++)
   {
    for(i=1; i<=nn.x; i++)
       for(j=1; j<=nn.y; j++)
          {ex_y[ind][i][j]=0; hx_y[ind][i][j]=0;
           ey_x[ind][i][j]=0; hy_x[ind][i][j]=0;
          }

    for(i=1; i<=nn.x; i++)
       for(k=1; k<=nn.z; k++)
          {ex_z[ind][i][k]=0; hx_z[ind][i][k]=0;
           ez_x[ind][i][k]=0; hz_x[ind][i][k]=0;
          }

    for(j=1; j<=nn.y; j++)
       for(k=1; k<=nn.z; k++)
          {ey_z[ind][j][k]=0; hy_z[ind][j][k]=0;
           ez_y[ind][j][k]=0; hz_y[ind][j][k]=0;
          }
   }
}
*********************************

**********par.cpp*************

//*********************************
// setup to get initial values***
//*********************************
#include "defin.h"

extern del0 del;
extern dis0 disp;
extern coff0 coff;
extern nxyz nn,n1;
extern int nstop;
extern dou C, PI, eps0, xmu0, dielc;
```

```
extern dou amp,beta;
extern dou dtedx,dtedy,dtedz;
extern dou dtmdx,dtmdy,dtmdz;
extern dou cxd,cxu,cyd,cyu,czd,czu,     //mur 1
          cxx,cyy,czz,cxfyd,cxfzd,cyfxd,cyfzd,czfxd,czfyd; //mur2
extern dou alpha, betadt, period, off, dt, delay, ampx, ampy, ampz;

void par(int nstop,dou amp, dou beta)
{int i;
dou eps[10], sigma[10];
dou dtxi=C/del.x, dtyi=C/del.y, dtzi=C/del.z;

dt=1.0/sqrt(dtxi*dtxi+dtyi*dtyi+dtzi*dtzi);
//c*** parameter alpha is the decay rate determined by beta.
betadt = beta*dt;
period = 2.0*(betadt);
alpha =pow(4./(betadt),2);

off=1.0;

for( i=1; i<=9; i++)
   { eps[i]=eps0; sigma[i]=0.0;}

//c***  define eps and sigma for each material here
eps[2]=dielc*eps0;
sigma[2]=0.005;

//c***  generate multiplicative constants for field update equations
//c***  free space
dtedx=dt/(eps0*del.x);
dtedy=dt/(eps0*del.y);
dtedz=dt/(eps0*del.z);
dtmdx=dt/(xmu0*del.x);
dtmdy=dt/(xmu0*del.y);
dtmdz=dt/(xmu0*del.z);

//c***  lossy dielectrics
for(i=2; i<=9; i++)
   {coff.esctc[i]=eps[i]/(eps[i]+sigma[i] *dt);
    coff.eincc[i]=sigma[i] *dt/(eps[i]+sigma[i] *dt);
    coff.edevcn[i]=dt*(eps[i]-eps0)/(eps[i]+sigma[i] *dt);
    coff.ecrlx[i]=dt/((eps[i]+sigma[i] *dt)*del.x);
```

```
    coff.ecrly[i]=dt/((eps[i]+sigma[i] *dt)*del.y);
    coff.ecrlz[i]=dt/((eps[i]+sigma[i] *dt)*del.z);
   }

//c***  compute outer radiation boundary condition (orbc) constants
cxd=(C*dt-del.x)/(C*dt+del.x);
cyd=(C*dt-del.y)/(C*dt+del.y);
czd=(C*dt-del.z)/(C*dt+del.z);
cxu=cxd;    cyu=cyd;    czu=czd;

//c***  compute 2nd order orbc constants
cxx=2.*del.x/(C*dt+del.x);
cyy=2.*del.y/(C*dt+del.y);
czz=2.*del.z/(C*dt+del.z);
cxfyd=del.x*C*dt*C*dt/(2.*del.y*del.y*(C*dt+del.x));
cxfzd=del.x*C*dt*C*dt/(2.*del.z*del.z*(C*dt+del.x));
cyfzd=del.y*C*dt*C*dt/(2.*del.z*del.z*(C*dt+del.y));
cyfxd=del.y*C*dt*C*dt/(2.*del.x*del.x*(C*dt+del.y));
czfxd=del.z*C*dt*C*dt/(2.*del.x*del.x*(C*dt+del.z));
czfyd=del.z*C*dt*C*dt/(2.*del.y*del.y*(C*dt+del.z));

}
*****************************************

**************e_field.cpp***************

//***************
//*** E field ***
//***************

#include "defin.h"
extern coff0 coff;
extern del0 del;
extern nxyz n1;
extern int flag_inc;
extern int idone[nx0][ny0][nz0],idtwo[nx0][ny0][nz0],idthre[nx0][ny0][nz0];
extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
           hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
extern dou dtedx,dtedy,dtedz;
extern dou wl;
extern dou SOURCE_OLD, SOURCE_NEW;
dou  ez_source0();
```

```
dou  ez_source1();

void exfld()
{
for(int k=2; k<=n1.z; k++)
   for(int j=2; j<=n1.y; j++)
      for(int i=1; i<=n1.x; i++)
 { //   determine material type
  if(idone[i][j][k]==0) // free space
    exs[i][j][k]+=(hzs[i][j][k]-hzs[i][j-1][k])*dtedy
                 -(hys[i][j][k]-hys[i][j][k-1])*dtedz;
  else if(idone[i][j][k]==1)  //  perfect conductor
    exs[i][j][k]=0.0;
          else if(idone[i][j][k]==-1) // source;
              {if(flag_inc==0)  exs[i][j][k]=-ez_source0()/del.x;
               else if(flag_inc==1) { dou TMP;TMP=ez_source1();
                                      exs[i][j][k]=-TMP/del.x;
                                      SOURCE_OLD=SOURCE_NEW; SOURCE_NEW=TMP;}
               else {cout<<"flag error in ex field flag_inc="
                         <<flag_inc<<"\n"; exit(0);}
               cout<<"ex"<<exs[i][j][k]<<endl;
              }
}
}

void eyfld()
{
for(int k=2; k<=n1.z; k++)
   for(int j=1; j<=n1.y; j++)
      for(int i=2; i<=n1.x; i++)
 { //   determine material type
  if(idtwo[i][j][k]==0) // free space
     eys[i][j][k]=eys[i][j][k]+(hxs[i][j][k]-hxs[i][j][k-1])*dtedz
      -(hzs[i][j][k]-hzs[i-1][j][k])*dtedx;
  else if(idtwo[i][j][k]==1)    //  perfect conductor
     eys[i][j][k]=0.0;
          else if(idtwo[i][j][k]==-1) // source;
              {if(flag_inc==0)  eys[i][j][k]=-ez_source0()/del.y;
               else if(flag_inc==1) { dou TMP;TMP=ez_source1();
                                      eys[i][j][k]=-TMP/del.y;
                                      SOURCE_OLD=SOURCE_NEW; SOURCE_NEW=TMP;}
               else {cout<<"flag error in ey field flag_inc="
```

```
                        <<flag_inc<<"\n"; exit(0);}
                cout<<"ey"<<eys[i][j][k]<<endl;
               }
}
}

void ezfld()
{
//dou  ez_source0();
//dou  ez_source1();
for(int k=1; k<=n1.z; k++)
   for(int j=2; j<=n1.y; j++)
      for(int i=2; i<=n1.x; i++)
 { //   determine material type
   if(idthre[i][j][k]==0) // free space
      ezs[i][j][k]=ezs[i][j][k]+(hys[i][j][k]-hys[i-1][j][k])*dtedx
       -(hxs[i][j][k]-hxs[i][j-1][k])*dtedy;
   else if(idthre[i][j][k]==1) //  perfect conductor
      ezs[i][j][k]=0.0;
   else if(idthre[i][j][k]==-1) // source point of the antenna***
             {if(flag_inc==0)  ezs[i][j][k]=-ez_source0()/del.z;
        // Rayleigh pulse {if(flag_inc==0)  ezs[i][j][k]=-ez_source0()/del.z;
        // if(flag_inc==0) ezs[i][j][k]+=(hys[i][j][k]-hys[i-1][j][k])*dtedx
        //                               -(hxs[i][j][k]-hxs[i][j-1][k])*dtedy
        //current source!!               -ez_source0()*dtedx/del.y;
             //if(flag_inc==0)
             //  {ezs[i][j][k]=0;
             //  exs[i][j][k]=-ez_source0()*2/del.x/log(del.x/0.0005);
             //  eys[i][j][k]=-ez_source0()*2/del.y/log(del.y/0.0005);
             //  exs[i-1][j][k]=ez_source0()*2/del.x/log(del.x/0.0005);
             //  eys[i][j-1][k]=ez_source0()*2/del.y/log(del.y/0.0005);
             //  } for magnetic frill source!!

              else if(flag_inc==1) { dou TMP;TMP=ez_source1();
                                     ezs[i][j][k]=-TMP/del.z;
                                     SOURCE_OLD=SOURCE_NEW; SOURCE_NEW=TMP;}
              else {cout<<"flag error in ez field flag_inc="
                        <<flag_inc<<"\n"; exit(0);}
              cout<<"ez"<<ezs[i][j][k]<<"\n";
             }
   }
}
```

```
************************************

***********source.cpp*************

#include "defin.h"
extern del0 del;
extern dis0 disp;

extern dou PI, C;
extern dou wl,t;

extern int i,j,k;
extern dou t,alpha,betadt;

dou ez_source0()
//*** Gaussian pulse***
{dou v1;
 if(t<0||t>2*betadt) v1=0;
 else v1=1.0*exp( -alpha*pow(t-betadt,2) );
 return(v1);
}

dou ez_source0R()
//*** Rayleigh pulse --the derivative of Gaussian pulse***
{dou v1;
 if(t<0||t>2*betadt) v1=0;
 else v1=-2*alpha*(t-betadt)*exp( -alpha*pow(t-betadt,2) );
 return(v1);
}

dou ez_source1() //*** sine wave source***
{dou v1=1.0*sin(2*PI*C/wl*t );
 return(v1);
}
************************************

***********Mur.cpp****************

//*************************
//*** Boundary condition***
//*************************
#include "defin.h"
```

```
extern nxyz nn;
extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
          hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
extern dou eysx1[5][ny0][nz0], ezsx1[5][ny0][nz0],
          ezsy1[nx0][5][nz0], exsy1[nx0][5][nz0],
          exsz1[nx0][ny0][5], eysz1[nx0][ny0][5];  //radsav
extern dou eysx2[5][ny0][nz0], ezsx2[5][ny0][nz0],
          ezsy2[nx0][5][nz0], exsy2[nx0][5][nz0],
          exsz2[nx0][ny0][5], eysz2[nx0][ny0][5];  //radsav2
extern dou cxd,cxu,cyd,cyu,czd,czu,     //mur 1
          cxx,cyy,czz,cxfyd,cxfzd,cyfxd,cyfzd,czfxd,czfyd; //mur2

int i, j, k;
void radeyx()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
        //c     do edges with first order orbc
for( k=2; k<=nz1; k++)
   { j=1;
    eys[1][j][k] =eysx1[2][j][k]+cxd*(eys[2][j][k]-eysx1[1][j][k]);
    eys[nn.x][j][k]=eysx1[3][j][k]+cxu*(eys[nx1][j][k]-eysx1[4][j][k]);
    j=ny1;
    eys[1][j][k] =eysx1[2][j][k]+cxd*(eys[2][j][k]-eysx1[1][j][k]) ;
    eys[nn.x][j][k]=eysx1[3][j][k]+cxu*(eys[nx1][j][k]-eysx1[4][j][k]);
   }
for( j=2; j<=ny1-1; j++)
   {k=2;
    eys[1][j][k] =eysx1[2][j][k]+cxd*(eys[2][j][k]-eysx1[1][j][k]);
    eys[nn.x][j][k]=eysx1[3][j][k]+cxu*(eys[nx1][j][k]-eysx1[4][j][k]);
    k=nz1;
    eys[1][j][k] =eysx1[2][j][k]+cxd*(eys[2][j][k]-eysx1[1][j][k])    ;
    eys[nn.x][j][k]=eysx1[3][j][k]+cxu*(eys[nx1][j][k]-eysx1[4][j][k]);
   }
        //c     now do 2nd order orbc on remaining portions of faces
for(k=3; k<=nz1-1; k++)
   for(j=2; j<=ny1-1; j++)
      {eys[1][j][k]=-eysx2[2][j][k]+cxd*(eys[2][j][k]+eysx2[1][j][k])
                    +cxx*(eysx1[1][j][k]+eysx1[2][j][k])
                    +cxfyd*(eysx1[1][j+1][k]-2.*eysx1[1][j][k]
                           +eysx1[1][j-1][k]+eysx1[2][j+1][k]
                           -2.*eysx1[2][j][k]+eysx1[2][j-1][k])
                    +cxfzd*(eysx1[1][j][k+1]-2.*eysx1[1][j][k]
                           +eysx1[1][j][k-1]+eysx1[2][j][k+1]
```

```
                        -2.*eysx1[2][j][k]+eysx1[2][j][k-1]);
      eys[nn.x][j][k]=-eysx2[3][j][k]+cxd*(eys[nx1][j][k]+eysx2[4][j][k])
                      +cxx*(eysx1[4][j][k]+eysx1[3][j][k])
                      +cxfyd*(eysx1[4][j+1][k]-2.*eysx1[4][j][k]
                             +eysx1[4][j-1][k]+eysx1[3][j+1][k]
                             -2.*eysx1[3][j][k]+eysx1[3][j-1][k])
                      +cxfzd*(eysx1[4][j][k+1]-2.*eysx1[4][j][k]
                             +eysx1[4][j][k-1]+eysx1[3][j][k+1]
                             -2.*eysx1[3][j][k]+eysx1[3][j][k-1]);
       }
        // now save past values
for(k=2; k<=nz1; k++)
   for(j=1; j<=ny1; j++)
      {   eysx2[1][j][k]=eysx1[1][j][k];  eysx2[2][j][k]=eysx1[2][j][k];
          eysx2[3][j][k]=eysx1[3][j][k];  eysx2[4][j][k]=eysx1[4][j][k];
          eysx1[1][j][k]=eys[1][j][k]  ;  eysx1[2][j][k]=eys[2][j][k]  ;
          eysx1[3][j][k]=eys[nx1][j][k];  eysx1[4][j][k]=eys[nn.x][j][k] ;
      }
}


void radezx()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
for( k=1; k<=nz1; k++)
   { j=2;
     ezs[1][j][k]=ezsx1[2][j][k]+cxd*(ezs[2][j][k]-ezsx1[1][j][k]);
     ezs[nn.x][j][k]=ezsx1[3][j][k]+cxu*(ezs[nx1][j][k]-ezsx1[4][j][k]);
     j=ny1;
     ezs[1][j][k]=ezsx1[2][j][k]+cxd*(ezs[2][j][k]-ezsx1[1][j][k]);
     ezs[nn.x][j][k]=ezsx1[3][j][k]+cxu*(ezs[nx1][j][k]-ezsx1[4][j][k]);

   }
for( j=3; j<=ny1-1; j++)
   {k=1;
    ezs[1][j][k]=ezsx1[2][j][k]+cxd*(ezs[2][j][k]-ezsx1[1][j][k]);
    ezs[nn.x][j][k]=ezsx1[3][j][k]+cxu*(ezs[nx1][j][k]-ezsx1[4][j][k]);
    k=nz1;
    ezs[1][j][k]=ezsx1[2][j][k]+cxd*(ezs[2][j][k]-ezsx1[1][j][k]);
    ezs[nn.x][j][k]=ezsx1[3][j][k]+cxu*(ezs[nx1][j][k]-ezsx1[4][j][k]);
   }
        //c     now do 2nd order orbc on remaining portions of faces
for( k=2; k<=nz1-1; k++)
```

```
   for( j=3; j<=ny1-1; j++)
      { ezs[1][j][k]=-ezsx2[2][j][k]+cxd*(ezs[2][j][k]+ezsx2[1][j][k])
                    +cxx*(ezsx1[1][j][k]+ezsx1[2][j][k])
                    +cxfyd*(ezsx1[1][j+1][k]-2.*ezsx1[1][j][k]
                           +ezsx1[1][j-1][k]+ezsx1[2][j+1][k]
                           -2.*ezsx1[2][j][k]+ezsx1[2][j-1][k])
                    +cxfzd*(ezsx1[1][j][k+1]-2.*ezsx1[1][j][k]
                           +ezsx1[1][j][k-1]+ezsx1[2][j][k+1]
                           -2.*ezsx1[2][j][k]+ezsx1[2][j][k-1])  ;
        ezs[nn.x][j][k]=-ezsx2[3][j][k]+cxd*(ezs[nx1][j][k]+ezsx2[4][j][k])
                       +cxx*(ezsx1[4][j][k]+ezsx1[3][j][k])
                       +cxfyd*(ezsx1[4][j+1][k]-2.*ezsx1[4][j][k]
                              +ezsx1[4][j-1][k]+ezsx1[3][j+1][k]
                              -2.*ezsx1[3][j][k]+ezsx1[3][j-1][k])
                       +cxfzd*(ezsx1[4][j][k+1]-2.*ezsx1[4][j][k]
                              +ezsx1[4][j][k-1]+ezsx1[3][j][k+1]
                              -2.*ezsx1[3][j][k]+ezsx1[3][j][k-1]) ;
        }
        //c     now save past values
for(k=1; k<=nz1; k++)
    for(j=2; j<=ny1; j++)
       {  ezsx2[1][j][k]=ezsx1[1][j][k];  ezsx2[2][j][k]=ezsx1[2][j][k];
          ezsx2[3][j][k]=ezsx1[3][j][k];  ezsx2[4][j][k]=ezsx1[4][j][k];
          ezsx1[1][j][k]=ezs[1][j][k]  ;  ezsx1[2][j][k]=ezs[2][j][k]  ;
          ezsx1[3][j][k]=ezs[nx1][j][k];  ezsx1[4][j][k]=ezs[nn.x][j][k] ;
        }
}


void radezy()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
        //c     do edges with first order orbc
for(  k=1;k<=nz1;k++)
   { i=2;
    ezs[i][1][k]=ezsy1[i][2][k]+cyd*(ezs[i][2][k]-ezsy1[i][1][k]);
    ezs[i][nn.y][k]=ezsy1[i][3][k]+cyd*(ezs[i][ny1][k]-ezsy1[i][4][k]) ;
    i=nx1;
    ezs[i][1][k]=ezsy1[i][2][k]+cyd*(ezs[i][2][k]-ezsy1[i][1][k]);
    ezs[i][nn.y][k]=ezsy1[i][3][k]+cyd*(ezs[i][ny1][k]-ezsy1[i][4][k]);
   }
for( i=3; i<=nx1-1; i++)
   { k=1 ;
```

```
    ezs[i][1][k]=ezsy1[i][2][k]+cyd*(ezs[i][2][k]-ezsy1[i][1][k]);
    ezs[i][nn.y][k]=ezsy1[i][3][k]+cyd*(ezs[i][ny1][k]-ezsy1[i][4][k]);
    k=nz1;
    ezs[i][1][k]=ezsy1[i][2][k]+cyd*(ezs[i][2][k]-ezsy1[i][1][k]);
    ezs[i][nn.y][k]=ezsy1[i][3][k]+cyd*(ezs[i][ny1][k]-ezsy1[i][4][k]);
   }
        //c     now do 2nd order orbc on remaining portions of faces
for(k=2;k<=nz1-1; k++)
   for(i=3;i<=nx1-1; i++)
      { ezs[i][1][k]=-ezsy2[i][2][k]+cyd*(ezs[i][2][k]+ezsy2[i][1][k])
                    +cyy*(ezsy1[i][1][k]+ezsy1[i][2][k])
                    +cyfxd*(ezsy1[i+1][1][k]-2.*ezsy1[i][1][k]
                           +ezsy1[i-1][1][k]+ezsy1[i+1][2][k]
                           -2.*ezsy1[i][2][k]+ezsy1[i-1][2][k])
                    +cyfzd*(ezsy1[i][1][k+1]-2.*ezsy1[i][1][k]
                           +ezsy1[i][1][k-1]+ezsy1[i][2][k+1]
                           -2.*ezsy1[i][2][k]+ezsy1[i][2][k-1]) ;
        ezs[i][nn.y][k]=-ezsy2[i][3][k]+cyd*(ezs[i][ny1][k]+ezsy2[i][4][k])
                       +cyy*(ezsy1[i][4][k]+ezsy1[i][3][k])
                       +cyfxd*(ezsy1[i+1][4][k]-2.*ezsy1[i][4][k]
                              +ezsy1[i-1][4][k]+ezsy1[i+1][3][k]
                              -2.*ezsy1[i][3][k]+ezsy1[i-1][3][k])
                       +cyfzd*(ezsy1[i][4][k+1]-2.*ezsy1[i][4][k]
                              +ezsy1[i][4][k-1]+ezsy1[i][3][k+1]
                              -2.*ezsy1[i][3][k]+ezsy1[i][3][k-1])   ;
   }
        //c     now save past values
for(k=1;k<=nz1; k++)
   for(i=2;i<=nx1; i++)
      {ezsy2[i][1][k]=ezsy1[i][1][k] ;  ezsy2[i][2][k]=ezsy1[i][2][k] ;
       ezsy2[i][3][k]=ezsy1[i][3][k] ;  ezsy2[i][4][k]=ezsy1[i][4][k] ;
       ezsy1[i][1][k]=ezs[i][1][k]   ;  ezsy1[i][2][k]=ezs[i][2][k]   ;
       ezsy1[i][3][k]=ezs[i][ny1][k] ;  ezsy1[i][4][k]=ezs[i][nn.y][k]  ;
      }
}

void radexy()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
        //c     do edges with first order orbc
for( k=2;k<=nz1; k++)
   {i=1;
    exs[i][1][k]=exsy1[i][2][k]+cyd*(exs[i][2][k]-exsy1[i][1][k]);
```

```
    exs[i][nn.y][k]=exsy1[i][3][k]+cyd*(exs[i][ny1][k]-exsy1[i][4][k]) ;
    i=nx1;
    exs[i][1][k]=exsy1[i][2][k]+cyd*(exs[i][2][k]-exsy1[i][1][k]);
    exs[i][nn.y][k]=exsy1[i][3][k]+cyd*(exs[i][ny1][k]-exsy1[i][4][k]);
   }
for( i=2; i<=nx1-1; i++)
   { k=2;
    exs[i][1][k]=exsy1[i][2][k]+cyd*(exs[i][2][k]-exsy1[i][1][k]);
    exs[i][nn.y][k]=exsy1[i][3][k]+cyd*(exs[i][ny1][k]-exsy1[i][4][k]);
    k=nz1;
    exs[i][1][k]=exsy1[i][2][k]+cyd*(exs[i][2][k]-exsy1[i][1][k]);
    exs[i][nn.y][k]=exsy1[i][3][k]+cyd*(exs[i][ny1][k]-exsy1[i][4][k]);
   }
        //c     now do 2nd order orbc on remaining portions of faces
for(k=3;k<=nz1-1; k++)
    for(i=2;i<=nx1-1; i++)
       {exs[i][1][k]=-exsy2[i][2][k]+cyd*(exs[i][2][k]+exsy2[i][1][k])
                     +cyy*(exsy1[i][1][k]+exsy1[i][2][k])
                     +cyfxd*(exsy1[i+1][1][k]-2.*exsy1[i][1][k]
                            +exsy1[i-1][1][k]+exsy1[i+1][2][k]
                            -2.*exsy1[i][2][k]+exsy1[i-1][2][k])
                     +cyfzd*(exsy1[i][1][k+1]-2.*exsy1[i][1][k]
                            +exsy1[i][1][k-1]+exsy1[i][2][k+1]
                            -2.*exsy1[i][2][k]+exsy1[i][2][k-1])  ;

        exs[i][nn.y][k]=-exsy2[i][3][k]+cyd*(exs[i][ny1][k]+exsy2[i][4][k])
                        +cyy*(exsy1[i][4][k]+exsy1[i][3][k])
                        +cyfxd*(exsy1[i+1][4][k]-2.*exsy1[i][4][k]
                               +exsy1[i-1][4][k]+exsy1[i+1][3][k]
                               -2.*exsy1[i][3][k]+exsy1[i-1][3][k])
                        +cyfzd*(exsy1[i][4][k+1]-2.*exsy1[i][4][k]
                               +exsy1[i][4][k-1] +exsy1[i][3][k+1]
                               -2.*exsy1[i][3][k]+exsy1[i][3][k-1])   ;
       }
        //c     now save past values
for(k=2;k<=nz1; k++)
   for(i=1;i<=nx1; i++)
      {   exsy2[i][1][k]=exsy1[i][1][k] ;   exsy2[i][2][k]=exsy1[i][2][k] ;
          exsy2[i][3][k]=exsy1[i][3][k] ;   exsy2[i][4][k]=exsy1[i][4][k] ;
          exsy1[i][1][k]=exs[i][1][k]   ;   exsy1[i][2][k]=exs[i][2][k]   ;
          exsy1[i][3][k]=exs[i][ny1][k] ;   exsy1[i][4][k]=exs[i][nn.y][k]  ;
       }
```

```
}

void radexz()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
        //c     do edges with first order orbc
for( j=2;j<=ny1; j++)
   { i=1;
        exs[i][j][1]=exsz1[i][j][2]+czd*(exs[i][j][2]-exsz1[i][j][1]);
        exs[i][j][nn.z]=exsz1[i][j][3]+czd*(exs[i][j][nz1]-exsz1[i][j][4]);
        i=nx1;
        exs[i][j][1]=exsz1[i][j][2]+czd*(exs[i][j][2]-exsz1[i][j][1]);
        exs[i][j][nn.z]=exsz1[i][j][3]+czd*(exs[i][j][nz1]-exsz1[i][j][4]);
   }
for( i=2;i<=nx1-1; i++)
{ int   j=2;
        exs[i][j][1]=exsz1[i][j][2]+czd*(exs[i][j][2]-exsz1[i][j][1]);
        exs[i][j][nn.z]=exsz1[i][j][3]+czd*(exs[i][j][nz1]-exsz1[i][j][4]);
        j=ny1;
        exs[i][j][1]=exsz1[i][j][2]+czd*(exs[i][j][2]-exsz1[i][j][1]);
        exs[i][j][nn.z]=exsz1[i][j][3]+czd*(exs[i][j][nz1]-exsz1[i][j][4]);
}
        //c     now do 2nd order orbc on remaining portions of faces
for( j=3; j<=ny1-1; j++)
   for(i=2; i<=nx1-1; i++)
      { exs[i][j][1]=-exsz2[i][j][2]+czd*(exs[i][j][2]+exsz2[i][j][1])
                     +czz*(exsz1[i][j][1]+exsz1[i][j][2])
                     +czfxd*(exsz1[i+1][j][1]-2.*exsz1[i][j][1]
                            +exsz1[i-1][j][1]+exsz1[i+1][j][2]
                            -2.*exsz1[i][j][2]+exsz1[i-1][j][2])
                     +czfyd*(exsz1[i][j+1][1]-2.*exsz1[i][j][1]
                            +exsz1[i][j-1][1]+exsz1[i][j+1][2]
                            -2.*exsz1[i][j][2]+exsz1[i][j-1][2]) ;
        exs[i][j][nn.z]=-exsz2[i][j][3]+czd*(exs[i][j][nz1]+exsz2[i][j][4])
                        +czz*(exsz1[i][j][4]+exsz1[i][j][3])
                        +czfxd*(exsz1[i+1][j][4]-2.*exsz1[i][j][4]
                               +exsz1[i-1][j][4]+exsz1[i+1][j][3]
                               -2.*exsz1[i][j][3]+exsz1[i-1][j][3])
                        +czfyd*(exsz1[i][j+1][4]-2.*exsz1[i][j][4]
                               +exsz1[i][j-1][4]+exsz1[i][j+1][3]
                               -2.*exsz1[i][j][3]+exsz1[i][j-1][3]) ;
      }
        //c     now save past values
```

```
for(j=2;j<=ny1; j++)
   for(i=1; i<=nx1; i++)
      {   exsz2[i][j][1]=exsz1[i][j][1];  exsz2[i][j][2]=exsz1[i][j][2];
          exsz2[i][j][3]=exsz1[i][j][3];  exsz2[i][j][4]=exsz1[i][j][4];
          exsz1[i][j][1]=exs[i][j][1]  ;  exsz1[i][j][2]=exs[i][j][2]  ;
          exsz1[i][j][3]=exs[i][j][nz1];  exsz1[i][j][4]=exs[i][j][nn.z] ;
      }
}


void radeyz()
{int nx1=nn.x-1, ny1=nn.y-1, nz1=nn.z-1;
        //c     do edges with first order orbc
for( j=1; j<=ny1; j++)
   { i=2;
     eys[i][j][1]=eysz1[i][j][2]+czd*(eys[i][j][2]-eysz1[i][j][1]);
     eys[i][j][nn.z]=eysz1[i][j][3]+czd*(eys[i][j][nz1]-eysz1[i][j][4]);
     i=nx1;
     eys[i][j][1]=eysz1[i][j][2]+czd*(eys[i][j][2]-eysz1[i][j][1]);
     eys[i][j][nn.z]=eysz1[i][j][3]+czd*(eys[i][j][nz1]-eysz1[i][j][4]);
    }
for( i=3; i<=nx1-1; i++)
   { j=1;
     eys[i][j][1]=eysz1[i][j][2]+czd*(eys[i][j][2]-eysz1[i][j][1]);
     eys[i][j][nn.z]=eysz1[i][j][3]+czd*(eys[i][j][nz1]-eysz1[i][j][4]);
     j=ny1;
     eys[i][j][1]=eysz1[i][j][2]+czd*(eys[i][j][2]-eysz1[i][j][1]);
     eys[i][j][nn.z]=eysz1[i][j][3]+czd*(eys[i][j][nz1]-eysz1[i][j][4]);
   }
        //c     now do 2]nd order orbc on remaining portions of faces
for(j=2; j<= ny1-1; j++)
   for(i=3; i<=nx1-1; i++)
      {eys[i][j][1]=-eysz2[i][j][2]+czd*(eys[i][j][2]+eysz2[i][j][1])
                  +czz*(eysz1[i][j][1]+eysz1[i][j][2])
                  +czfxd*(eysz1[i+1][j][1]-2.*eysz1[i][j][1]
                         +eysz1[i-1][j][1]+eysz1[i+1][j][2]
                         -2.*eysz1[i][j][2]+eysz1[i-1][j][2])
                  +czfyd*(eysz1[i][j+1][1]-2.*eysz1[i][j][1]
                         +eysz1[i][j-1][1]+eysz1[i][j+1][2]
                         -2.*eysz1[i][j][2]+eysz1[i][j-1][2]);
       eys[i][j][nn.z]=-eysz2[i][j][3]+czd*(eys[i][j][nz1]+eysz2[i][j][4])
                     +czz*(eysz1[i][j][4]+eysz1[i][j][3])
```

```
                        +czfxd*(eysz1[i+1][j][4]-2.*eysz1[i][j][4]
                               +eysz1[i-1][j][4]+eysz1[i+1][j][3]
                               -2.*eysz1[i][j][3]+eysz1[i-1][j][3])
                        +czfyd*(eysz1[i][j+1][4]-2.*eysz1[i][j][4]
                               +eysz1[i][j-1][4]+eysz1[i][j+1][3]
                               -2.*eysz1[i][j][3]+eysz1[i][j-1][3]);
      }
        //c     now save past values
for( j=1; j<=ny1;j++)
   for(i=2; i<=nx1; i++)
      {eysz2[i][j][1]=eysz1[i][j][1];       eysz2[i][j][2]=eysz1[i][j][2];
       eysz2[i][j][3]=eysz1[i][j][3];       eysz2[i][j][4]=eysz1[i][j][4];
       eysz1[i][j][1]=eys[i][j][1]  ;       eysz1[i][j][2]=eys[i][j][2]  ;
       eysz1[i][j][3]=eys[i][j][nz1];       eysz1[i][j][4]=eys[i][j][nn.z] ;
      }
}
************************************

*********h_field.cpp***************

//*************************
//*** H field component ***
//*************************

#include "defin.h"
extern nxyz n1;
extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
           hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
extern dou dtmdx,dtmdy,dtmdz;

void hxfld()
{
for(int k=1; k<=n1.z;k++)
   for(int j=1; j<=n1.y;j++)
      for(int i=2; i<=n1.x;i++)
          hxs[i][j][k]=hxs[i][j][k]-(ezs[i][j+1][k]-ezs[i][j][k])*dtmdy
                                   +(eys[i][j][k+1]-eys[i][j][k])*dtmdz;
}

void hyfld()
{
for(int k=1; k<=n1.z;k++)
```

```
   for(int j=2; j<=n1.y;j++)
      for(int i=1; i<=n1.x;i++)
          hys[i][j][k]=hys[i][j][k]-(exs[i][j][k+1]-exs[i][j][k])*dtmdz
                                   +(ezs[i+1][j][k]-ezs[i][j][k])*dtmdx;
}

void hzfld()
{
for(int k=2; k<=n1.z;k++)
   for(int j=1; j<=n1.y;j++)
      for(int i=1; i<=n1.x;i++)
          hzs[i][j][k]=hzs[i][j][k]-(eys[i+1][j][k]-eys[i][j][k])*dtmdx
                                   +(exs[i][j+1][k]-exs[i][j][k])*dtmdy ;
}
*******************************

************wires.cpp********

#include "defin.h"
#include <fstream.h>
extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
           hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];
extern del0 del;
extern coff0 coff;
extern dou dtmdx, dtmdy, dtmdz, r0, t;
void writeFile(char *, fstream &);
void current(int, fstream &);
static int  flag=0;
static char FL[20];
static int  A[20][6];
static int  num_wires;
void fine_wire()
{fstream op;
 int i,j,k;
 int IA, JA, KA, IA1, IA2, JA1, JA2, KA1, KA2;
 if(flag==0)
   {op.open("wire.dat", ios::in);
    op>>num_wires;
    for(i=1; i<=num_wires; i++)
      op>>FL[i]>>A[i][1]>>A[i][2]>>A[i][3]>>A[i][4]>>A[i][5];
    op.close();
    cout<<FL[1]<<" "<<A[1][1]<<" "<<A[1][2]<<" "<<A[1][3]<<" "
```

```
                    <<A[1][4]<<" " <<A[1][5]<<endl;
   }
 //*****************************************************************
 fstream fout1, fout2;
for(i=1; i<=num_wires; i++)
  {switch(FL[i]){
   case 'z': IA=A[i][1]; JA=A[i][2]; KA1=A[i][3]; KA2=A[i][4];
             for(k=KA1+1; k<KA2; k++)
                {int IT=IA-1,JT=JA-1;
                 hys[IA][JA][k]+= (2/log(del.x/r0)-1)*dtmdx*(ezs[IA+1][JA][k]
                                  -ezs[IA][JA][k]);
                 hys[IT][JA][k]+= (2/log(del.x/r0)-1)*dtmdx*(ezs[IT+1][JA][k]
                                  -ezs[IT][JA][k]);
                 hxs[IA][JA][k]+=-(2/log(del.y/r0)-1)*dtmdy*(ezs[IA][JA+1][k]
                                  -ezs[IA][JA][k]);
                 hxs[IA][JT][k]+=-(2/log(del.y/r0)-1)*dtmdy*(ezs[IA][JT+1][k]
                                  -ezs[IA][JT][k]);
                }break;
   case 'y': IA=A[i][1]; JA1=A[i][2]; JA2=A[i][3]; KA=A[i][4];
             for(j=JA1+1; j<JA2; j++)
                {int IT=IA-1,KT=KA-1;
                 hzs[IA][j][KA]+=-(2/log(del.x/r0)-1)*dtmdx*(eys[IA+1][j][KA]
                                  -eys[IA][j][KA]);
                 hzs[IT][j][KA]+=-(2/log(del.x/r0)-1)*dtmdx*(eys[IT+1][j][KA]
                                  -eys[IT][j][KA]);
                 hxs[IA][j][KA]+= (2/log(del.z/r0)-1)*dtmdz*(eys[IA][j][KA+1]
                                  -eys[IA][j][KA]);
                 hxs[IA][j][KT]+= (2/log(del.z/r0)-1)*dtmdz*(eys[IA][j][KT+1]
                                  -eys[IA][j][KT]);
                }break;
   case 'x': IA1=A[i][1]; IA2=A[i][2]; JA=A[i][3]; KA=A[i][4];
             for(i=IA1+1; i<IA2; i++)
                {int KT=KA-1,JT=JA-1;
                 hzs[i][JA][KA]+= (2/log(del.y/r0)-1)*dtmdy*(exs[i][JA+1][KA]
                                  -exs[i][JA][KA]);
                 hzs[i][JT][KA]+= (2/log(del.y/r0)-1)*dtmdy*(exs[i][JT+1][KA]
                                  -exs[i][JT][KA]);
                 hys[i][JA][KA]+=-(2/log(del.z/r0)-1)*dtmdz*(exs[i][JA][KA+1]
                                  -exs[i][JA][KA]);
                 hys[i][JA][KT]+=-(2/log(del.z/r0)-1)*dtmdz*(exs[i][JA][KT+1]
                                  -exs[i][JA][KT]);
                }break;
```

```
                }
  }
 if(flag==0)
   {flag++;
    fout1.open("cura.dat",ios::out); fout2.open("curb.dat",ios::out);}
 else
   {fout1.open("cura.dat",ios::app); fout2.open("curb.dat",ios::app);}
 current(1, fout1);
 fout1.close();
 fout2.close();
}

//*********************************
//*** save the data for current***
//*********************************
void current(int flag, fstream & fout)
{int k,j,i,IA, JA, KA;
dou ca;
//cout<<A[1][5]<<endl;
fout<<t<<" " ;
if(flag==1)
  {switch(FL[1]){
     case 'z': IA=A[1][1]; JA=A[1][2];
               for( k=A[1][5]; k<=A[1][4]; k++)
                {ca=(hxs[IA][JA-1][k]-hxs[IA][JA][k])*del.x+ \
                    (hys[IA][JA][k]-hys[IA-1][JA][k])*del.y;
                 fout<<ca<<"  ";
                }fout<<"\n";
                break;
     case 'y': IA=A[1][1]; KA=A[1][4];
               for( j=A[1][5]; j<=A[1][3]; j++)
                {ca=(hxs[IA][j][KA]-hxs[IA][j][KA-1])*del.x+ \
                    (hzs[IA-1][j][KA]-hzs[IA][j][KA])*del.z;
                 fout<<ca<<"  ";
                }fout<<"\n";
                break;
     case 'x': JA=A[1][3]; KA=A[1][4];
               for( i=A[1][5]; i<=A[1][2]; i++)
                {ca=(hzs[i][JA][KA]-hzs[i][JA-1][KA])*del.z+ \
                    (hys[i][JA][KA-1]-hys[i][JA][KA])*del.y;
                 fout<<ca<<"  ";
                }fout<<"\n";
```

```
                break;
      }
  }

else;
}
************************************

**********wave_stable.cpp************

#include "defin.h"

extern double value_ERROR;
extern double SOURCE_NEW, SOURCE_OLD;
void wave_stable(double t2, int *flag_stable, int *n0)
{
static double t1=0;
static unsigned stable[5]={0,0,0,0,0};      //used as flags *********
static double maxium_f[5]={-1,-2,-3,-4,-5}; //4 maxiums needed*******
static enum max_psss_flag
           { pass0,pass1,pass2,pass3,pass4,stay } max_pass=pass0;

double error_st1, error_st2,error_st3, error_st4;

   //if ( *flag_stable!=1 ) // not reached the stable status.
   switch(max_pass)
     {
      case pass0: if( t2>t1 && t1>=0 && t2>maxium_f[1] )
                    { maxium_f[1]=t2; stable[1]=-1; }
                  else if( t2<t1 && t1<0  && stable[1]==-1 )
                    max_pass=pass1;
                  break;
      case pass1: if( t2>t1 && t1>=0 && t2>maxium_f[2] )
                    { maxium_f[2]=t2; stable[2]=-1; }
                  else if( t2<t1 && t1<0  && stable[2]==-1 )
                    max_pass=pass2;
                  break;
      case pass2: if( t2>t1 && t1>=0 && t2>maxium_f[3] )
                    { maxium_f[3]=t2; stable[3]=-1; }
                  else if( t2<t1 && t1<0  && stable[3]==-1 )
                    max_pass=pass3;
                  break;
```

```
        case pass3: if( t2>t1 && t1>=0 && t2>maxium_f[4] )
                      { maxium_f[4]=t2; stable[4]=-1; }
                    else if( t2<t1 && t1<0  && stable[4]==-1 )
                      { max_pass=pass4; stable[4]=0; }
                    break;
        case pass4:
                    error_st1=fabs(maxium_f[1]-maxium_f[2])/maxium_f[2];
                    error_st2=fabs(maxium_f[2]-maxium_f[3])/maxium_f[3];
                    error_st3=fabs(maxium_f[3]-maxium_f[4])/maxium_f[4];
                    error_st4=fabs(maxium_f[1]-maxium_f[4])/maxium_f[4];
                    if( error_st1<value_ERROR && error_st2<value_ERROR \
                      &&error_st3<value_ERROR && error_st4<value_ERROR )
                       max_pass=stay; // but not found the 'zero' point*****
                    else
                      {max_pass=pass3;
                       maxium_f[1]=maxium_f[2];
                       maxium_f[2]=maxium_f[3];
                       maxium_f[3]=maxium_f[4];
                       maxium_f[4]=0;
                      }
                    break;
        case stay : if(SOURCE_NEW>=0 &&SOURCE_OLD<0)
      {*flag_stable=1;  // found the starting point***.
               *n0-=2;      // go back two step for phase finding
      }

                    else break;
        default: ;
      };//switch ends!

t1=t2;
}
**********************************

***********find_amp.cpp*********

#include "defin.h"

#define In1 Ind-1
#define Jn1 Jnd-1
#define Kn1 Knd-1
#define Is1 Ist-1
```

```
#define Js1 Jst-1
#define Ks1 Kst-1

#define MAX(a,b) a>b?a:b

extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
           hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];

//*************surface_field amplitude(maxium value*****
extern dou ex_y[2][nx0][ny0],hx_y[4][nx0][ny0], // X-component
           ex_z[2][nx0][nz0],hx_z[4][nx0][nz0], //************

           ey_x[2][nx0][ny0],hy_x[4][nx0][ny0], // Y-component
   ey_z[2][ny0][nz0],hy_z[4][ny0][nz0], //************

           ez_x[2][nx0][nz0],hz_x[4][nx0][nz0], // Z-component
           ez_y[2][ny0][nz0],hz_y[4][ny0][nz0]; //************

extern int Ist, Ind, Jst, Jnd, Kst, Knd;
//********************

void find_max()
{
int i,j,k;

for(i=Ist; i<=Ind; i++)      // X-component**************
   {for(j=Jst; j<=Jnd; j++) // z-constant plane*********
       {ex_y[0][i][j]=MAX(ex_y[0][i][j],exs[i][j][Kst]);
        ex_y[1][i][j]=MAX(ex_y[1][i][j],exs[i][j][Knd]);
        hx_y[0][i][j]=MAX(hx_y[0][i][j],hxs[i][j][Kst]);
        hx_y[1][i][j]=MAX(hx_y[1][i][j],hxs[i][j][Kn1]);
        hx_y[2][i][j]=MAX(hx_y[2][i][j],hxs[i][j][Ks1]);
        hx_y[3][i][j]=MAX(hx_y[3][i][j],hxs[i][j][Knd]);


       }
    for(k=Kst; k<=Knd; k++) //y-constant plane*********
       {ex_z[0][i][k]=MAX(ex_z[0][i][k],exs[i][Jst][k]);
        ex_z[1][i][k]=MAX(ex_z[1][i][k],exs[i][Jnd][k]);
        hx_z[0][i][k]=MAX(hx_z[0][i][k],hxs[i][Jst][k]);
        hx_z[1][i][k]=MAX(hx_z[1][i][k],hxs[i][Jn1][k]);
        hx_z[2][i][k]=MAX(hx_z[2][i][k],hxs[i][Js1][k]);
```

```
        hx_z[3][i][k]=MAX(hx_z[3][i][k],hxs[i][Jnd][k]);
       }
   }

for(j=Jst; j<=Jnd; j++)      // Y-component**************
   {for(i=Ist; i<=Ind; i++) // z-constant plane*********
       {ey_x[0][i][j]=MAX(ey_x[0][i][j],eys[i][j][Kst]);
        ey_x[1][i][j]=MAX(ey_x[1][i][j],eys[i][j][Knd]);
        hy_x[0][i][j]=MAX(hy_x[0][i][j],hys[i][j][Kst]);
        hy_x[1][i][j]=MAX(hy_x[1][i][j],hys[i][j][Kn1]);
        hy_x[2][i][j]=MAX(hy_x[2][i][j],hys[i][j][Ks1]);
        hy_x[3][i][j]=MAX(hy_x[3][i][j],hys[i][j][Knd]);
       }
    for(k=Kst; k<=Knd; k++) // x-constant plane********
       {ey_z[0][j][k]=MAX(ey_z[0][j][k],eys[Ist][j][k]);
        ey_z[1][j][k]=MAX(ey_z[1][j][k],eys[Ind][j][k]);
hy_z[0][j][k]=MAX(hy_z[0][j][k],hys[Ist][j][k]);
        hy_z[1][j][k]=MAX(hy_z[1][j][k],hys[In1][j][k]);
hy_z[2][j][k]=MAX(hy_z[2][j][k],hys[Is1][j][k]);
        hy_z[3][j][k]=MAX(hy_z[3][j][k],hys[Ind][j][k]);
       }
   }

for(k=Kst; k<=Knd; k++)      // Z-component***************
   {for(i=Ist; i<=Ind; i++) // y-constnat plane***********
       {ez_x[0][i][k]=MAX(ez_x[0][i][k],ezs[i][Jst][k]);
        ez_x[1][i][k]=MAX(ez_x[1][i][k],ezs[i][Jnd][k]);
        hz_x[0][i][k]=MAX(hz_x[0][i][k],hzs[i][Jst][k]);
        hz_x[1][i][k]=MAX(hz_x[1][i][k],hzs[i][Jn1][k]);
        hz_x[2][i][k]=MAX(hz_x[2][i][k],hzs[i][Js1][k]);
        hz_x[3][i][k]=MAX(hz_x[3][i][k],hzs[i][Jnd][k]);
       }

    for(j=Jst; j<=Jnd; j++) //x-constant plane*************
       {ez_y[0][j][k]=MAX(ez_y[0][j][k],ezs[Ist][j][k]);
        ez_y[1][j][k]=MAX(ez_y[1][j][k],ezs[Ind][j][k]);
        hz_y[0][j][k]=MAX(hz_y[0][j][k],hzs[Ist][j][k]);
        hz_y[1][j][k]=MAX(hz_y[1][j][k],hzs[In1][j][k]);
        hz_y[2][j][k]=MAX(hz_y[2][j][k],hzs[Is1][j][k]);
        hz_y[3][j][k]=MAX(hz_y[3][j][k],hzs[Ind][j][k]);
       }
   }
```

```
}
*******************************

*********find_phase.cpp***********

#include "defin.h"

#define In1 Ind-1
#define Jn1 Jnd-1
#define Kn1 Knd-1
#define Is1 Ist-1
#define Js1 Jst-1
#define Ks1 Kst-1

extern dou exs[nx0][ny0][nz0],eys[nx0][ny0][nz0],ezs[nx0][ny0][nz0],
           hxs[nx0][ny0][nz0],hys[nx0][ny0][nz0],hzs[nx0][ny0][nz0];

//*****************relative phase of the surface current**********
extern complex  ex_y_p[2][nx0][ny0],hx_y_p[4][nx0][ny0], //***X-component**
                ex_z_p[2][nx0][nz0],hx_z_p[4][nx0][nz0], //*****************

                ey_x_p[2][nx0][ny0],hy_x_p[4][nx0][ny0], //***Y-component**
                ey_z_p[2][ny0][nz0],hy_z_p[4][ny0][nz0], //*****************

                ez_x_p[2][nx0][nz0],hz_x_p[4][nx0][nz0], //***Z-component**
                ez_y_p[2][ny0][nz0],hz_y_p[4][ny0][nz0]; //*****************
//*****************************************************************
extern int Ist, Ind, Jst, Jnd, Kst, Knd;
extern nxyz nn;
extern int n0;

void find_phase(double phase_n)
{phase_n+=0.0000001; // give a very small value for later judgement
complex jjc=complex(0,phase_n);
int i,j,k;
static flag_first=1;

if( (flag_first--)==1 ) //******initial amplitude for comparision************
  {for(i=Ist; i<=Ind; i++)     // X-component**************
      {for(j=Jst; j<=Jnd; j++) // z-constant plane*********
          {ex_y_p[0][i][j]=exs[i][j][Kst]; ex_y_p[1][i][j]=exs[i][j][Knd];
           hx_y_p[0][i][j]=hxs[i][j][Kst]; hx_y_p[1][i][j]=hxs[i][j][Kn1];
```

```
            hx_y_p[2][i][j]=hxs[i][j][Ks1]; hx_y_p[3][i][j]=hxs[i][j][Knd];
           }
        for(k=Kst; k<=Knd; k++) //y-constant plane*********
           {ex_z_p[0][i][k]=exs[i][Jst][k]; ex_z_p[1][i][k]=exs[i][Jnd][k];
            hx_z_p[0][i][k]=hxs[i][Jst][k]; hx_z_p[1][i][k]=hxs[i][Jn1][k];
            hx_z_p[2][i][k]=hxs[i][Js1][k]; hx_z_p[3][i][k]=hxs[i][Jnd][k];
           }
      }

   for(j=Jst; j<=Jnd; j++)      // Y-component**************
      {for(i=Ist; i<=Ind; i++) // z-constant plane*********
           {ey_x_p[0][i][j]=eys[i][j][Kst]; ey_x_p[1][i][j]=eys[i][j][Knd];
            hy_x_p[0][i][j]=hys[i][j][Kst]; hy_x_p[1][i][j]=hys[i][j][Kn1];
            hy_x_p[2][i][j]=hys[i][j][Ks1]; hy_x_p[3][i][j]=hys[i][j][Knd];
           }
        for(k=Kst; k<=Knd; k++) // x-constant plane********
           {ey_z_p[0][j][k]=eys[Ist][j][k]; ey_z_p[1][j][k]=eys[Ind][j][k];
            hy_z_p[0][j][k]=hys[Ist][j][k]; hy_z_p[1][j][k]=hys[In1][j][k];
            hy_z_p[2][j][k]=hys[Is1][j][k]; hy_z_p[3][j][k]=hys[Ind][j][k];
           }
      }

   for(k=Kst; k<=Knd; k++)      // Z-component***************
      {for(i=Ist; i<=Ind; i++) // y-constnat plane***********
           {ez_x_p[0][i][k]=ezs[i][Jst][k]; ez_x_p[1][i][k]=ezs[i][Jnd][k];
            hz_x_p[0][i][k]=hzs[i][Jst][k]; hz_x_p[1][i][k]=hzs[i][Jn1][k];
            hz_x_p[2][i][k]=hzs[i][Js1][k]; hz_x_p[3][i][k]=hzs[i][Jnd][k];
           }
        for(j=Jst; j<=Jnd; j++) //x-component*****************
           {ez_y_p[0][j][k]=ezs[Ist][j][k]; ez_y_p[1][j][k]=ezs[Ind][j][k];
            hz_y_p[0][j][k]=hzs[Ist][j][k]; hz_y_p[1][j][k]=hzs[In1][j][k];
            hz_y_p[2][j][k]=hzs[Is1][j][k]; hz_y_p[3][j][k]=hzs[Ind][j][k];
           }
      }
  }// end if ********

//Till now, the imaginary parts of the phases are 0!

else
{
for(i=Ist; i<=Ind; i++)      // X-component***************
   {for(j=Jst; j<=Jnd; j++) // z-constant plane**********
```

```
      {
if(imag(ex_y_p[0][i][j])==.0 && real(ex_y_p[0][i][j])<.0
          && exs[i][j][Kst]>=.0)
         ex_y_p[0][i][j]=exs[i][j][Kst]+jjc; //    just found the point!
       else if(imag(ex_y_p[0][i][j])!=.0); // already found : do nothing!
else ex_y_p[0][i][j]=exs[i][j][Kst];

       if(imag(ex_y_p[1][i][j])==.0 && real(ex_y_p[1][i][j])<.0
          && exs[i][j][Knd]>=.0)
          ex_y_p[1][i][j]=exs[i][j][Knd]+jjc; //    just found the point!
else if(imag(ex_y_p[1][i][j])!=.0); // already found : do nothing!
else ex_y_p[1][i][j]=exs[i][j][Knd];

if(imag(hx_y_p[0][i][j])==.0 && real(hx_y_p[0][i][j])<.0
          && hxs[i][j][Kst]>=.0)
          hx_y_p[0][i][j]=hxs[i][j][Kst]+jjc; //    just found the point!
else if(imag(hx_y_p[0][i][j])!=.0); // already found : do nothing!
else hx_y_p[0][i][j]=hxs[i][j][Kst];

if(imag(hx_y_p[1][i][j])==.0 && real(hx_y_p[1][i][j])<.0
          && hxs[i][j][Kn1]>=.0)
          hx_y_p[1][i][j]=hxs[i][j][Kn1]+jjc; //    just found the point!
else if(imag(hx_y_p[1][i][j])!=.0); // already found : do nothing!
else         hx_y_p[1][i][j]=hxs[i][j][Kn1];

if(imag(hx_y_p[2][i][j])==.0 && real(hx_y_p[2][i][j])<.0
          && hxs[i][j][Ks1]>=.0)
          hx_y_p[2][i][j]=hxs[i][j][Ks1]+jjc; //    just found the point!
else if(imag(hx_y_p[2][i][j])!=.0); // already found : do nothing!
else         hx_y_p[2][i][j]=hxs[i][j][Ks1];

if(imag(hx_y_p[3][i][j])==.0 && real(hx_y_p[3][i][j])<.0
          && hxs[i][j][Knd]>=.0)
          hx_y_p[3][i][j]=hxs[i][j][Knd]+jjc; //    just found the point!
else if(imag(hx_y_p[3][i][j])!=.0); // already found : do nothing!
else         hx_y_p[3][i][j]=hxs[i][j][Knd];
       }
    for(k=Kst; k<=Knd; k++) //y-constant plane************
       {if(imag(ex_z_p[0][i][k])==.0 && real(ex_z_p[0][i][k])<.0
          && exs[i][Jst][k]>=.0)
          ex_z_p[0][i][k]=exs[i][Jst][k]+jjc; //    just found the point!
else if(imag(ex_z_p[0][i][k])!=.0); // already found : do nothing!
```

```
else          ex_z_p[0][i][k]=exs[i][Jst][k];

if(imag(ex_z_p[1][i][k])==.0 && real(ex_z_p[1][i][k])<.0
           && exs[i][Jnd][k]>=.0)
           ex_z_p[1][i][k]=exs[i][Jnd][k]+jjc; //    just found the point!
else if(imag(ex_z_p[1][i][k])!=.0); // already found : do nothing!
else          ex_z_p[1][i][k]=exs[i][Jnd][k];

if(imag(hx_z_p[0][i][k])==.0 && real(hx_z_p[0][i][k])<.0
           && hxs[i][Jst][k]>=.0)
           hx_z_p[0][i][k]=hxs[i][Jst][k]+jjc;//    just found the point!
else if(imag(hx_z_p[0][i][k])!=.0); // already found : do nothing!
else          hx_z_p[0][i][k]=hxs[i][Jst][k];

if(imag(hx_z_p[1][i][k])==.0 && real(hx_z_p[1][i][k])<.0
           && hxs[i][Jn1][k]>=.0)
           hx_z_p[1][i][k]=hxs[i][Jn1][k]+jjc; //    just found the point!
else if(imag(hx_z_p[1][i][k])!=.0); // already found : do nothing!
else          hx_z_p[1][i][k]=hxs[i][Jn1][k];

if(imag(hx_z_p[2][i][k])==.0 && real(hx_z_p[2][i][k])<.0
           && hxs[i][Js1][k]>=.0)
           hx_z_p[2][i][k]=hxs[i][Js1][k]+jjc;//    just found the point!
else if(imag(hx_z_p[2][i][k])!=.0); // already found : do nothing!
else          hx_z_p[2][i][k]=hxs[i][Js1][k];

if(imag(hx_z_p[3][i][k])==.0 && real(hx_z_p[3][i][k])<.0
           && hxs[i][Jnd][k]>=.0)
           hx_z_p[3][i][k]=hxs[i][Jnd][k]+jjc; //    just found the point!
else if(imag(hx_z_p[3][i][k])!=.0); // already found : do nothing!
else          hx_z_p[3][i][k]=hxs[i][Jnd][k];
      }
   }

for(j=Jst; j<=Jnd; j++)     // Y-component***************
   {for(i=Ist; i<=Ind; i++) // z-constant plane***********
       {if(imag(ey_x_p[0][i][j])==.0 && real(ey_x_p[0][i][j])<.0
           && eys[i][j][Kst]>=.0)
           ey_x_p[0][i][j]=eys[i][j][Kst]+jjc; //    just found the point!
else if(imag(ey_x_p[0][i][j])!=.0); // already found : do nothing!
        else          ey_x_p[0][i][j]=eys[i][j][Kst];
```

```
        if(imag(ey_x_p[1][i][j])==.0 && real(ey_x_p[1][i][j])<.0
           && eys[i][j][Knd]>=.0)
           ey_x_p[1][i][j]=eys[i][j][Knd]+jjc;  //    just found the point!
else if(imag(ey_x_p[1][i][j])!=.0); // already found : do nothing!
else         ey_x_p[1][i][j]=eys[i][j][Knd];

        if(imag(hy_x_p[0][i][j])==.0 && real(hy_x_p[0][i][j])<.0
           && hys[i][j][Kst]>=.0)
           hy_x_p[0][i][j]=hys[i][j][Kst]+jjc;  //    just found the point!
else if(imag(hy_x_p[0][i][j])!=.0); // already found : do nothing!
else         hy_x_p[0][i][j]=hys[i][j][Kst];

        if(imag(hy_x_p[1][i][j])==.0 && real(hy_x_p[1][i][j])<.0
           && hys[i][j][Kn1]>=.0)
           hy_x_p[1][i][j]=hys[i][j][Kn1]+jjc;  //    just found the point!
else if(imag(hy_x_p[1][i][j])!=.0); // already found : do nothing!
else         hy_x_p[1][i][j]=hys[i][j][Kn1];

        if(imag(hy_x_p[2][i][j])==.0 && real(hy_x_p[2][i][j])<.0
           && hys[i][j][Ks1]>=.0)
           hy_x_p[2][i][j]=hys[i][j][Ks1]+jjc;  //    just found the point!
else if(imag(hy_x_p[2][i][j])!=.0); // already found : do nothing!
else         hy_x_p[2][i][j]=hys[i][j][Ks1];

        if(imag(hy_x_p[3][i][j])==.0 && real(hy_x_p[3][i][j])<.0
           && hys[i][j][Knd]>=.0)
           hy_x_p[3][i][j]=hys[i][j][Knd]+jjc;  //    just found the point!
else if(imag(hy_x_p[3][i][j])!=.0); // already found : do nothing!
else         hy_x_p[3][i][j]=hys[i][j][Knd];
      }
   for(k=Kst; k<=Knd; k++) // x-constant plane***********
      {if(imag(ey_z_p[0][j][k])==.0 && real(ey_z_p[0][j][k])<.0
           && eys[Ist][j][k]>=.0)
           ey_z_p[0][j][k]=eys[Ist][j][k]+jjc; //    just found the point!
else if(imag(ey_z_p[0][j][k])!=.0); // already found : do nothing!
else         ey_z_p[0][j][k]=eys[Ist][j][k];

        if(imag(ey_z_p[1][j][k])==.0 && real(ey_z_p[1][j][k])<.0
           && eys[Ind][j][k]>=.0)
           ey_z_p[1][j][k]=eys[Ind][j][k]+jjc; //    just found the point!
else if(imag(ey_z_p[1][j][k])!=.0); // already found : do nothing!
else         ey_z_p[1][j][k]=eys[Ind][j][k];
```

```
        if(imag(hy_z_p[0][j][k])==.0 && real(hy_z_p[0][j][k])<.0
           && hys[Ist][j][k]>=.0)
           hy_z_p[0][j][k]=hys[Ist][j][k]+jjc; //    just found the point!
else if(imag(hy_z_p[0][j][k])!=.0); // already found : do nothing!
else         hy_z_p[0][j][k]=hys[Ist][j][k];

        if(imag(hy_z_p[1][j][k])==.0 && real(hy_z_p[1][j][k])<.0
           && hys[In1][j][k]>=.0)
           hy_z_p[1][j][k]=hys[In1][j][k]+jjc; //    just found the point!
else if(imag(hy_z_p[1][j][k])!=.0); // already found : do nothing!
else         hy_z_p[1][j][k]=hys[In1][j][k];

        if(imag(hy_z_p[2][j][k])==.0 && real(hy_z_p[2][j][k])<.0
           && hys[Is1][j][k]>=.0)
           hy_z_p[2][j][k]=hys[Is1][j][k]+jjc; //    just found the point!
else if(imag(hy_z_p[2][j][k])!=.0); // already found : do nothing!
else         hy_z_p[2][j][k]=hys[Is1][j][k];

        if(imag(hy_z_p[3][j][k])==.0 && real(hy_z_p[3][j][k])<.0
           && hys[Ind][j][k]>=.0)
           hy_z_p[3][j][k]=hys[Ind][j][k]+jjc; //    just found the point!
else if(imag(hy_z_p[3][j][k])!=.0); // already found : do nothing!
else         hy_z_p[3][j][k]=hys[Ind][j][k];
      }
   }

for(k=Kst; k<=Knd; k++)      // Z-component****************
   {for(i=Ist; i<=Ind; i++) // y-constnat plane***********
       {if(imag(ez_x_p[0][i][k])==.0 && real(ez_x_p[0][i][k])<.0
           && ezs[i][Jst][k]>=.0)
           ez_x_p[0][i][k]=ezs[i][Jst][k]+jjc; //    just found the point!
else if(imag(ez_x_p[0][i][k])!=.0); // already found : do nothing!
        else         ez_x_p[0][i][k]=ezs[i][Jst][k];

if(imag(ez_x_p[1][i][k])==.0 && real(ez_x_p[1][i][k])<.0
           && ezs[i][Jnd][k]>=.0)
           ez_x_p[1][i][k]=ezs[i][Jnd][k]+jjc; //    just found the point!
else if(imag(ez_x_p[1][i][k])!=.0); // already found : do nothing!
else         ez_x_p[1][i][k]=ezs[i][Jnd][k];

if(imag(hz_x_p[0][i][k])==.0 && real(hz_x_p[0][i][k])<.0
```

```
          && hzs[i][Jst][k]>=.0)
          hz_x_p[0][i][k]=hzs[i][Jst][k]+jjc; //    just found the point!
else if(imag(hz_x_p[0][i][k])!=.0); // already found : do nothing!
else         hz_x_p[0][i][k]=hzs[i][Jst][k];

if(imag(hz_x_p[1][i][k])==.0 && real(hz_x_p[1][i][k])<.0
          && hzs[i][Jn1][k]>=.0)
          hz_x_p[1][i][k]=hzs[i][Jn1][k]+jjc; //    just found the point!
else if(imag(hz_x_p[1][i][k])!=.0); // already found : do nothing!
else         hz_x_p[1][i][k]=hzs[i][Jn1][k];

if(imag(hz_x_p[2][i][k])==.0 && real(hz_x_p[2][i][k])<.0
          && hzs[i][Js1][k]>=.0)
          hz_x_p[2][i][k]=hzs[i][Js1][k]+jjc; //    just found the point!
else if(imag(hz_x_p[2][i][k])!=.0); // already found : do nothing!
else         hz_x_p[2][i][k]=hzs[i][Js1][k];

if(imag(hz_x_p[3][i][k])==.0 && real(hz_x_p[3][i][k])<.0
          && hzs[i][Jnd][k]>=.0)
          hz_x_p[3][i][k]=hzs[i][Jnd][k]+jjc; //    just found the point!
else if(imag(hz_x_p[3][i][k])!=.0); // already found : do nothing!
else         hz_x_p[3][i][k]=hzs[i][Jnd][k];
      }
   for(j=Jst; j<=Jnd; j++) //x-constant plane***********
      {if(imag(ez_y_p[0][j][k])==.0 && real(ez_y_p[0][j][k])<.0
          && ezs[Ist][j][k]>=.0)
          ez_y_p[0][j][k]=ezs[Ist][j][k]+jjc; //    just found the point!
else if(imag(ez_y_p[0][j][k])!=.0); // already found : do nothing!
else         ez_y_p[0][j][k]=ezs[Ist][j][k];

if(imag(ez_y_p[1][j][k])==.0 && real(ez_y_p[1][j][k])<.0
          && ezs[Ind][j][k]>=.0)
          ez_y_p[1][j][k]=ezs[Ind][j][k]+jjc; //    just found the point!
else if(imag(ez_y_p[1][j][k])!=.0); // already found : do nothing!
else         ez_y_p[1][j][k]=ezs[Ind][j][k];

if(imag(hz_y_p[0][j][k])==.0 && real(hz_y_p[0][j][k])<.0
          && hzs[Ist][j][k]>=.0)
          hz_y_p[0][j][k]=hzs[Ist][j][k]+jjc; //    just found the point!
else if(imag(hz_y_p[0][j][k])!=.0); // already found : do nothing!
else         hz_y_p[0][j][k]=hzs[Ist][j][k];
```

```
if(imag(hz_y_p[1][j][k])==.0 && real(hz_y_p[1][j][k])<.0
           && hzs[In1][j][k]>=.0)
           hz_y_p[1][j][k]=hzs[In1][j][k]+jjc; //    just found the point!
else if(imag(hz_y_p[1][j][k])!=.0); // already found : do nothing!
else         hz_y_p[1][j][k]=hzs[In1][j][k];

if(imag(hz_y_p[2][j][k])==.0 && real(hz_y_p[2][j][k])<.0
           && hzs[Is1][j][k]>=.0)
           hz_y_p[2][j][k]=hzs[Is1][j][k]+jjc; //    just found the point!
else if(imag(hz_y_p[2][j][k])!=.0); // already found : do nothing!
else         hz_y_p[2][j][k]=hzs[Is1][j][k];

if(imag(hz_y_p[3][j][k])==.0 && real(hz_y_p[3][j][k])<.0
           && hzs[Ind][j][k]>=.0)
           hz_y_p[3][j][k]=hzs[Ind][j][k]+jjc; //    just found the point!
else if(imag(hz_y_p[3][j][k])!=.0); // already found : do nothing!
else         hz_y_p[3][j][k]=hzs[Ind][j][k];
      }
   }
}// ends if*****
}//ends main**********
*************************

************out_surface_field.cpp*********

#include "defin.h"
#include <iostream.h>
//*************surface_field*****
extern dou ex_y[2][nx0][ny0],hx_y[4][nx0][ny0], // X-component
           ex_z[2][nx0][nz0],hx_z[4][nx0][nz0], //************

           ey_x[2][nx0][ny0],hy_x[4][nx0][ny0], // Y-component
           ey_z[2][ny0][nz0],hy_z[4][ny0][nz0], //************

           ez_x[2][nx0][nz0],hz_x[4][nx0][nz0], // Z-component
           ez_y[2][ny0][nz0],hz_y[4][ny0][nz0];
         //**surface_current_amplitude**

extern complex  ex_y_p[2][nx0][ny0],hx_y_p[4][nx0][ny0],
                ex_z_p[2][nx0][nz0],hx_z_p[4][nx0][nz0],

                ey_x_p[2][nx0][ny0],hy_x_p[4][nx0][ny0],
```

```
                ey_z_p[2][ny0][nz0],hy_z_p[4][ny0][nz0],

                ez_x_p[2][nx0][nz0],hz_x_p[4][nx0][nz0],
                ez_y_p[2][ny0][nz0],hz_y_p[4][ny0][nz0];
                //**surface_current_phase

extern complex exy[2][nx0][ny0],hxy[2][nx0][ny0], // X-component
               exz[2][nx0][nz0],hxz[2][nx0][nz0], //************

               eyx[2][nx0][ny0],hyx[2][nx0][ny0], // Y-component
               eyz[2][ny0][nz0],hyz[2][ny0][nz0], //************

               ezx[2][nx0][nz0],hzx[2][nx0][nz0], // Z-component
               ezy[2][ny0][nz0],hzy[2][ny0][nz0]; //**final surface_field**

extern int Ist, Ind, Jst, Jnd, Kst, Knd;
extern dou PI, ONE_cycle;
//*******************
void  out_surface_field(double ONE_cycle)
{double dtor; dtor=PI/180;
//***********************************************
int i,j,k;
complex jj; jj=complex(0.0,-1.0);
double e0,e1,h0,h1,h2,h3;
double dp; dp=PI/ONE_cycle;  //The delayed phase angle***
for(i=Ist; i<=Ind; i++)      // X-component**************
   for(j=Jst; j<=Jnd; j++) // z-constant plane*********
      {e0=dtor*imag(ex_y_p[0][i][j]);
       e1=dtor*imag(ex_y_p[1][i][j]);
       h0=dtor*imag(hx_y_p[0][i][j]);
       h1=dtor*imag(hx_y_p[1][i][j]);
       h2=dtor*imag(hx_y_p[2][i][j]);
       h3=dtor*imag(hx_y_p[3][i][j]);
       if(ex_y[0][i][j]!=0.) e0-=asin(real(ex_y_p[0][i][j])/ex_y[0][i][j]);
       if(ex_y[1][i][j]!=0.) e1-=asin(real(ex_y_p[1][i][j])/ex_y[1][i][j]);
       if(hx_y[0][i][j]!=0.) h0-=asin(real(hx_y_p[0][i][j])/hx_y[0][i][j])-dp;
       if(hx_y[1][i][j]!=0.) h1-=asin(real(hx_y_p[1][i][j])/hx_y[1][i][j])-dp;
       if(hx_y[2][i][j]!=0.) h2-=asin(real(hx_y_p[2][i][j])/hx_y[2][i][j])-dp;
       if(hx_y[3][i][j]!=0.) h3-=asin(real(hx_y_p[3][i][j])/hx_y[3][i][j])-dp;
       exy[0][i][j]= ex_y[0][i][j]*exp(jj*e0);
       exy[1][i][j]= ex_y[1][i][j]*exp(jj*e1);
       hxy[0][i][j]=(hx_y[0][i][j]*exp(jj*h0)+hx_y[2][i][j]*exp(jj*h2) )/2.0;
```

```
        hxy[1][i][j]=(hx_y[1][i][j]*exp(jj*h1)+hx_y[3][i][j]*exp(jj*h3) )/2.0;
       }

for(i=Ist; i<=Ind; i++)      // X-component**************
   for(k=Kst; k<=Knd; k++) //y-constant plane*********
      {double e0=dtor*imag(ex_z_p[0][i][k]);
       double e1=dtor*imag(ex_z_p[1][i][k]);
       double h0=dtor*imag(hx_z_p[0][i][k]);
       double h1=dtor*imag(hx_z_p[1][i][k]);
       double h2=dtor*imag(hx_z_p[2][i][k]);
       double h3=dtor*imag(hx_z_p[3][i][k]);
       if(ex_z[0][i][k]!=0.) e0-=asin(real(ex_z_p[0][i][k])/ex_z[0][i][k]);
       if(ex_z[1][i][k]!=0.) e1-=asin(real(ex_z_p[1][i][k])/ex_z[1][i][k]);
       if(hx_z[0][i][k]!=0.) h0-=asin(real(hx_z_p[0][i][k])/hx_z[0][i][k])-dp;
       if(hx_z[1][i][k]!=0.) h1-=asin(real(hx_z_p[1][i][k])/hx_z[1][i][k])-dp;
       if(hx_z[2][i][k]!=0.) h2-=asin(real(hx_z_p[2][i][k])/hx_z[2][i][k])-dp;
       if(hx_z[3][i][k]!=0.) h3-=asin(real(hx_z_p[3][i][k])/hx_z[3][i][k])-dp;
       exz[0][i][k]= ex_z[0][i][k]*exp(jj*e0);
       exz[1][i][k]= ex_z[1][i][k]*exp(jj*e1);
       hxz[0][i][k]=(hx_z[0][i][k]*exp(jj*h0)+hx_z[2][i][k]*exp(jj*h2) )/2.0;
       hxz[1][i][k]=(hx_z[1][i][k]*exp(jj*h1)+hx_z[3][i][k]*exp(jj*h3) )/2.0;
      }
//*****************************************************

for(j=Jst; j<=Jnd; j++)      // Y-component**************
   for(i=Ist; i<=Ind; i++) // z-constant plane*********
      {double e0=dtor*imag(ey_x_p[0][i][j]);
       double e1=dtor*imag(ey_x_p[1][i][j]);
       double h0=dtor*imag(hy_x_p[0][i][j]);
       double h1=dtor*imag(hy_x_p[1][i][j]);
       double h2=dtor*imag(hy_x_p[2][i][j]);
       double h3=dtor*imag(hy_x_p[3][i][j]);
       if(ey_x[0][i][j]!=0.) e0-=asin(real(ey_x_p[0][i][j])/ey_x[0][i][j]);
       if(ey_x[1][i][j]!=0.) e1-=asin(real(ey_x_p[1][i][j])/ey_x[1][i][j]);
       if(hy_x[0][i][j]!=0.) h0-=asin(real(hy_x_p[0][i][j])/hy_x[0][i][j])-dp;
       if(hy_x[1][i][j]!=0.) h1-=asin(real(hy_x_p[1][i][j])/hy_x[1][i][j])-dp;
       if(hy_x[2][i][j]!=0.) h2-=asin(real(hy_x_p[2][i][j])/hy_x[2][i][j])-dp;
       if(hy_x[3][i][j]!=0.) h3-=asin(real(hy_x_p[3][i][j])/hy_x[3][i][j])-dp;
       eyx[0][i][j]= ey_x[0][i][j]*exp(jj*e0);
       eyx[1][i][j]= ey_x[1][i][j]*exp(jj*e1);
       hyx[0][i][j]=(hy_x[0][i][j]*exp(jj*h0)+hy_x[2][i][j]*exp(jj*h2) )/2.0;
       hyx[1][i][j]=(hy_x[1][i][j]*exp(jj*h1)+hy_x[3][i][j]*exp(jj*h3) )/2.0;
```

```
      }

for(j=Jst; j<=Jnd; j++)      // Y-component**************
   for(k=Kst; k<=Knd; k++) // x-constant plane*********
      {double e0=dtor*imag(ey_z_p[0][j][k]);
       double e1=dtor*imag(ey_z_p[1][j][k]);
       double h0=dtor*imag(hy_z_p[0][j][k]);
       double h1=dtor*imag(hy_z_p[1][j][k]);
       double h2=dtor*imag(hy_z_p[2][j][k]);
       double h3=dtor*imag(hy_z_p[3][j][k]);
       if(ey_z[0][j][k]!=0.) e0-=asin(real(ey_z_p[0][j][k])/ey_z[0][j][k]);
       if(ey_z[1][j][k]!=0.) e1-=asin(real(ey_z_p[1][j][k])/ey_z[1][j][k]);
       if(hy_z[0][j][k]!=0.) h0-=asin(real(hy_z_p[0][j][k])/hy_z[0][j][k])-dp;
       if(hy_z[1][j][k]!=0.) h1-=asin(real(hy_z_p[1][j][k])/hy_z[1][j][k])-dp;
       if(hy_z[2][j][k]!=0.) h2-=asin(real(hy_z_p[2][j][k])/hy_z[2][j][k])-dp;
       if(hy_z[3][j][k]!=0.) h3-=asin(real(hy_z_p[3][j][k])/hy_z[3][j][k])-dp;
       eyz[0][j][k]= ey_z[0][j][k]*exp(jj*e0);
       eyz[1][j][k]= ey_z[1][j][k]*exp(jj*e1);
       hyz[0][j][k]=(hy_z[0][j][k]*exp(jj*h0)+hy_z[2][j][k]*exp(jj*h2) )/2.0;
       hyz[1][j][k]=(hy_z[1][j][k]*exp(jj*h1)+hy_z[3][j][k]*exp(jj*h3) )/2.0;
      }
//*****************************************************

for(k=Kst; k<=Knd; k++)      // Z-component***************
   for(i=Ist; i<=Ind; i++) // y-constnat plane***********
      {double e0=dtor*imag(ez_x_p[0][i][k]);
       double e1=dtor*imag(ez_x_p[1][i][k]);
       double h0=dtor*imag(hz_x_p[0][i][k]);
       double h1=dtor*imag(hz_x_p[1][i][k]);
       double h2=dtor*imag(hz_x_p[2][i][k]);
       double h3=dtor*imag(hz_x_p[3][i][k]);
       if(ez_x[0][i][k]!=0.) e0-=asin(real(ez_x_p[0][i][k])/ez_x[0][i][k]);
       if(ez_x[1][i][k]!=0.) e1-=asin(real(ez_x_p[1][i][k])/ez_x[1][i][k]);
       if(hz_x[0][i][k]!=0.) h0-=asin(real(hz_x_p[0][i][k])/hz_x[0][i][k])-dp;
       if(hz_x[1][i][k]!=0.) h1-=asin(real(hz_x_p[1][i][k])/hz_x[1][i][k])-dp;
       if(hz_x[2][i][k]!=0.) h2-=asin(real(hz_x_p[2][i][k])/hz_x[2][i][k])-dp;
       if(hz_x[3][i][k]!=0.) h3-=asin(real(hz_x_p[3][i][k])/hz_x[3][i][k])-dp;
       ezx[0][i][k]= ez_x[0][i][k]*exp(jj*e0);
       ezx[1][i][k]= ez_x[1][i][k]*exp(jj*e1);
       hzx[0][i][k]=(hz_x[0][i][k]*exp(jj*h0)+hz_x[2][i][k]*exp(jj*h2) )/2.0;
       hzx[1][i][k]=(hz_x[1][i][k]*exp(jj*h1)+hz_x[3][i][k]*exp(jj*h3) )/2.0;
      }
```

```
for(k=Kst; k<=Knd; k++)     // Z-component****************
   for(j=Jst; j<=Jnd; j++) //x-constant plane ***********
      {double e0=dtor*imag(ez_y_p[0][j][k]);
       double e1=dtor*imag(ez_y_p[1][j][k]);
       double h0=dtor*imag(hz_y_p[0][j][k]);
       double h1=dtor*imag(hz_y_p[1][j][k]);
       double h2=dtor*imag(hz_y_p[2][j][k]);
       double h3=dtor*imag(hz_y_p[3][j][k]);
       if(ez_y[0][j][k]!=0.) e0-=asin(real(ez_y_p[0][j][k])/ez_y[0][j][k]);
       if(ez_y[1][j][k]!=0.) e1-=asin(real(ez_y_p[1][j][k])/ez_y[1][j][k]);
       if(hz_y[0][j][k]!=0.) h0-=asin(real(hz_y_p[0][j][k])/hz_y[0][j][k])-dp;
       if(hz_y[1][j][k]!=0.) h1-=asin(real(hz_y_p[1][j][k])/hz_y[1][j][k])-dp;
       if(hz_y[2][j][k]!=0.) h2-=asin(real(hz_y_p[2][j][k])/hz_y[2][j][k])-dp;
       if(hz_y[3][j][k]!=0.) h3-=asin(real(hz_y_p[3][j][k])/hz_y[3][j][k])-dp;
       ezy[0][j][k]= ez_y[0][j][k]*exp(jj*e0);
       ezy[1][j][k]= ez_y[1][j][k]*exp(jj*e1);
       hzy[0][j][k]=(hz_y[0][j][k]*exp(jj*h0)+hz_y[2][j][k]*exp(jj*h2) )/2.0;
       hzy[1][j][k]=(hz_y[1][j][k]*exp(jj*h1)+hz_y[3][j][k]*exp(jj*h3) )/2.0;
      }
}//end
*********************************

*********far_field.cpp*********

#include "defin.h"
#include "complex.h"
void readFile(char *, fstream &);
void writeFile(char *, fstream &);
extern dou PI, eta0;
extern del0 del;
extern double wl;
extern double cen_x, cen_y, cen_z;
extern int Ist, Ind , Jst , Jnd , Kst, Knd;
extern complex exy[2][nx0][ny0],hxy[2][nx0][ny0], // X-component
               exz[2][nx0][nz0],hxz[2][nx0][nz0], //*************

               eyx[2][nx0][ny0],hyx[2][nx0][ny0], // Y-component
               eyz[2][ny0][nz0],hyz[2][ny0][nz0], //*************

               ezx[2][nx0][nz0],hzx[2][nx0][nz0], // Z-component
               ezy[2][ny0][nz0],hzy[2][ny0][nz0]; //*************
```

```
const double dtor=0.0174532925199; // degree to radian
const double rtod=57.295779513082; // radian to degree
complex Nx[362], Ny[362], Nz[362],
        Lx[362], Ly[362], Lz[362];
fstream fout;
int qaz;
//************************************************
void far_field(int angle, int ID_ang, char *OUTPUT_name )
{void find_Nx(double ), find_Ny(double ), find_Nz(double ),
      find_Lx(double ), find_Ly(double ), find_Lz(double );
 void find_Nx1(double ), find_Ny1(double ), find_Nz1(double ),
      find_Lx1(double ), find_Ly1(double ), find_Lz1(double );
 double fi, ct;
 complex Ect[362], Efi[362];
 complex Nct, Lct, Nfi, Lfi;
 double cosfi, sinfi, cosct, sinct;
 if(ID_ang==1)
   {fi=angle*dtor;
    find_Nx(fi); find_Ny(fi); find_Nz(fi);
    find_Lx(fi); find_Ly(fi); find_Lz(fi);
    cosfi=cos(fi);
    sinfi=sin(fi);
    writeFile(OUTPUT_name, fout);
    for(int i=1; i<=180; i++)
       {cosct=cos(i*dtor);
        sinct=sin(i*dtor);
        Nct=(Nx[i]*cosfi+Ny[i]*sinfi)*cosct-Nz[i]*sinct;
        Lct=(Lx[i]*cosfi+Ly[i]*sinfi)*cosct-Lz[i]*sinct;
        Nfi=-Nx[i]*sinfi+Ny[i]*cosfi;
        Lfi=-Lx[i]*sinfi+Ly[i]*cosfi;

        Ect[i]=Lfi+Nct*eta0;
        Efi[i]=Lct-Nfi*eta0;
        fout<<i<<" " << abs(Ect[i])<<endl;
       }
    fout.close();
   }
 else
  {ct=angle*dtor;
    find_Nx1(ct); find_Ny1(ct); find_Nz1(ct);
    find_Lx1(ct); find_Ly1(ct); find_Lz1(ct);
    cosct=cos(ct);
```

```
     sinct=sin(ct);
     writeFile(OUTPUT_name, fout);
     for(int i=1; i<=360; i++)
        { cosfi=cos(i*dtor); sinfi=sin(i*dtor);
          Nct=(Nx[i]*cosfi+Ny[i]*sinfi)*cosct-Nz[i]*sinct;
          Lct=(Lx[i]*cosfi+Ly[i]*sinfi)*cosct-Lz[i]*sinct;
          Nfi=-Nx[i]*sinfi+Ny[i]*cosfi;
          Lfi=-Lx[i]*sinfi+Ly[i]*cosfi;

          Ect[i]=Lfi+Nct*eta0;
          Efi[i]=Lct-Nfi*eta0;
          fout<<i<<" " << abs(Ect[i])<<endl;
        }
     fout.close();
    }
}
//*************************************************
//*** exp(jkr'cos()) ****************************
complex exp_phase(double x1, double y1, double z1, double ct, double fi)
{double cosf;
 complex y ;
 cosf=z1*cos(ct)+( x1*cos(fi)+y1*sin(fi) )*sin(ct);
 y=exp(complex(0.0,1.0)*2*PI/wl*cosf);
 return(y);
}
//***********************over*****************
//*************************************************
void find_Nx(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
complex Jx, Nx1, Nx2;
writeFile("NX.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Nx1=0; Nx2=0;
    for(i=Ist; i<=Ind-1; i++)
       {x1=(i-cen_x+0.5)*del.x;
     //***************************
        for(j=Jst; j<=Jnd; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y)*del.y;
               if(m==0) {z1=(Kst-cen_z)*del.z; Jx= hyx[0][i][j];}
```

```
                else     {z1=(Knd-cen_z)*del.z; Jx=-hyx[1][i][j];}
                if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
                Nx1+=Jx*exp_phase(x1,y1,z1,ct_r,fi)*coff;
               }
     //**************************
        for(k=Kst; k<=Knd; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z)*del.z;
               if(m==0) {y1=(Jst-cen_y)*del.y; Jx=-hzx[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Jx= hzx[1][i][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Nx2+=Jx*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
       }
    Nx[ct]=del.x*(Nx1*del.y+Nx2*del.z);
     fout<<ct<<"," << real(Nx[ct])<<", " <<imag(Nx[ct])<<endl;
   }
fout.close();
}
//***********************over*****************
//************************************************
void find_Ny(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
complex Jy, Ny1, Ny2;
writeFile("NY.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Ny1=0; Ny2=0;
    for(j=Jst; j<=Jnd-1; j++)
       {y1=(j-cen_y+0.5)*del.y;
     //**************************
        for(i=Ist; i<=Ind; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x)*del.x;
               if(m==0) {z1=(Kst-cen_z)*del.z; Jy=-hxy[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; Jy= hxy[1][i][j];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Ny1+=Jy*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
     //**************************
        for(k=Kst; k<=Knd; k++)
```

```
            for(m=0; m<=1; m++)
               {z1=(k-cen_z)*del.z;
                if(m==0) {x1=(Ist-cen_x)*del.x; Jy= hzy[0][j][k];}
                else     {x1=(Ind-cen_x)*del.x; Jy=-hzy[1][j][k];}
                if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
                Ny2+=Jy*exp_phase(x1,y1,z1,ct_r,fi)*coff;
               }
         }
     Ny[ct]=del.y*(Ny1*del.x+Ny2*del.z);
      fout<<ct<<"," << real(Ny[ct])<<", " <<imag(Ny[ct])<<endl;
    }
fout.close();
}
//**********************over*****************
//***************************************************
void find_Nz(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
complex Jz, Nz1, Nz2;
writeFile("NZ.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Nz1=0; Nz2=0;
    for(k=Kst; k<=Knd-1; k++)
       {z1=(k-cen_z+0.5)*del.z;
     //**************************
        for(j=Jst; j<=Jnd; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y)*del.y;
               if(m==0) {x1=(Ist-cen_x)*del.x; Jz=-hyz[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; Jz= hyz[1][j][k];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Nz2+=Jz*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
     //**************************
        for(i=Ist; i<=Ind; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x)*del.x;
               if(m==0) {y1=(Jst-cen_y)*del.y; Jz= hxz[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Jz=-hxz[1][i][k];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Nz1+=Jz*exp_phase(x1,y1,z1,ct_r,fi)*coff;
```

```
                }
        }
     Nz[ct]=del.z*(Nz1*del.x+Nz2*del.y);
      fout<<ct<<"," << real(Nz[ct])<<", " <<imag(Nz[ct])<<endl;
    }
fout.close();
}
//**********************over*****************
//***************************************************
void find_Lx(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
complex Mx, Lx1, Lx2;
writeFile("LX.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Lx1=0; Lx2=0;
    for(i=Ist; i<=Ind; i++)
       {x1=(i-cen_x)*del.x;
     //**************************
        for(j=Jst; j<=Jnd-1; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y+0.5)*del.y;
               if(m==0) {z1=(Kst-cen_z)*del.z; Mx= eyx[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; Mx=-eyx[1][i][j];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Lx1-=Mx*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
     //**************************
        for(k=Kst; k<=Knd-1; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z+0.5)*del.z;
               if(m==0) {y1=(Jst-cen_y)*del.y; Mx=-ezx[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Mx= ezx[1][i][k];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Lx2-=Mx*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
       }
    Lx[ct]=del.x*(Lx1*del.y+Lx2*del.z);
     fout<<ct<<"," << real(Lx[ct])<<", " <<imag(Lx[ct])<<endl;
   }
fout.close();
```

```
}
//**********************over*****************
//***************************************************
void find_Ly(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
complex My, Ly1, Ly2;
writeFile("LY.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Ly1=0; Ly2=0;
    for(j=Jst; j<=Jnd; j++)
       {y1=(j-cen_y)*del.y;
     //***************************
        for(i=Ist; i<=Ind-1; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x+0.5)*del.x;
               if(m==0) {z1=(Kst-cen_z)*del.z; My=-exy[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; My= exy[1][i][j];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Ly1-=My*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
     //***************************
        for(k=Kst; k<=Knd-1; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z+0.5)*del.z;
               if(m==0) {x1=(Ist-cen_x)*del.x; My= ezy[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; My=-ezy[1][j][k];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Ly2-=My*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
       }
    Ly[ct]=del.y*(Ly1*del.x+Ly2*del.z);
     fout<<ct<<"," << real(Ly[ct])<<", " <<imag(Ly[ct])<<endl;
   }
fout.close();
}
//**********************over*****************
//***************************************************
void find_Lz(double fi)
{int ct, i,j,k,m;
double x1, y1, z1, ct_r, coff;
```

```
complex Mz, Lz1, Lz2;
writeFile("LZ.dat", fout);
for(ct=1;ct<=180;ct++)
   {ct_r=ct*dtor;
    Lz1=0; Lz2=0;
    for(k=Kst; k<=Knd; k++)
       {z1=(k-cen_z)*del.z;
     //**************************
        for(j=Jst; j<=Jnd-1; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y+0.5)*del.y;
               if(m==0) {x1=(Ist-cen_x)*del.x; Mz=-eyz[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; Mz= eyz[1][j][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Lz2-=Mz*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
     //**************************
        for(i=Ist; i<=Ind-1; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x+0.5)*del.x;
               if(m==0) {y1=(Jst-cen_y)*del.y; Mz= exz[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Mz=-exz[1][i][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Lz1-=Mz*exp_phase(x1,y1,z1,ct_r,fi)*coff;
              }
       }
    Lz[ct]=del.z*(Lz1*del.x+Lz2*del.y);
     fout<<ct<<"," << real(Lz[ct])<<", " <<imag(Lz[ct])<<endl;
   }
fout.close();
}
//**********************over*****************
//**********************over*****************
//************************************************
void find_Nx1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex Jx, Nx1, Nx2;
writeFile("NX.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Nx1=0; Nx2=0;
```

```
    for(i=Ist; i<=Ind-1; i++)
       {x1=(i-cen_x+0.5)*del.x;
     //**************************
        for(j=Jst; j<=Jnd; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y)*del.y;
               if(m==0) {z1=(Kst-cen_z)*del.z; Jx= hyx[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; Jx=-hyx[1][i][j];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Nx1+=Jx*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
     //**************************
        for(k=Kst; k<=Knd; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z)*del.z;
               if(m==0) {y1=(Jst-cen_y)*del.y; Jx=-hzx[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Jx= hzx[1][i][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Nx2+=Jx*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
       }
    Nx[fi]=del.x*(Nx1*del.y+Nx2*del.z);
     fout<<fi<<"," << real(Nx[fi])<<", " <<imag(Nx[fi])<<endl;
   }
fout.close();
}
//************************over*****************
//**************************************************
void find_Ny1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex Jy, Ny1, Ny2;
writeFile("NY.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Ny1=0; Ny2=0;
    for(j=Jst; j<=Jnd-1; j++)
       {y1=(j-cen_y+0.5)*del.y;
     //**************************
        for(i=Ist; i<=Ind; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x)*del.x;
```

```
                if(m==0) {z1=(Kst-cen_z)*del.z; Jy=-hxy[0][i][j];}
                else     {z1=(Knd-cen_z)*del.z; Jy= hxy[1][i][j];}
                if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
                Ny1+=Jy*exp_phase(x1,y1,z1,ct,fi_r)*coff;
               }
     //**************************
        for(k=Kst; k<=Knd; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z)*del.z;
               if(m==0) {x1=(Ist-cen_x)*del.x; Jy= hzy[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; Jy=-hzy[1][j][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Ny2+=Jy*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
       }
    Ny[fi]=del.y*(Ny1*del.x+Ny2*del.z);
     fout<<fi<<"," << real(Ny[fi])<<", " <<imag(Ny[fi])<<endl;
   }
fout.close();
}
//***********************over*****************
//**************************************************
void find_Nz1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex Jz, Nz1, Nz2;
writeFile("NZ.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Nz1=0; Nz2=0;
    for(k=Kst; k<=Knd-1; k++)
       {z1=(k-cen_z+0.5)*del.z;
     //**************************
        for(j=Jst; j<=Jnd; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y)*del.y;
               if(m==0) {x1=(Ist-cen_x)*del.x; Jz=-hyz[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; Jz= hyz[1][j][k];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Nz2+=Jz*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
     //**************************
```

```
        for(i=Ist; i<=Ind; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x)*del.x;
               if(m==0) {y1=(Jst-cen_y)*del.y; Jz= hxz[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Jz=-hxz[1][i][k];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Nz1+=Jz*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
       }
    Nz[fi]=del.z*(Nz1*del.x+Nz2*del.y);
     fout<<fi<<"," << real(Nz[fi])<<", " <<imag(Nz[fi])<<endl;
   }
fout.close();
}
//**********************over*****************
//***************************************************
void find_Lx1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex Mx, Lx1, Lx2;
writeFile("LX.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Lx1=0; Lx2=0;
    for(i=Ist; i<=Ind; i++)
       {x1=(i-cen_x)*del.x;
     //**************************
        for(j=Jst; j<=Jnd-1; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y+0.5)*del.y;
               if(m==0) {z1=(Kst-cen_z)*del.z; Mx= eyx[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; Mx=-eyx[1][i][j];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
               Lx1-=Mx*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
     //**************************
        for(k=Kst; k<=Knd-1; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z+0.5)*del.z;
               if(m==0) {y1=(Jst-cen_y)*del.y; Mx=-ezx[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Mx= ezx[1][i][k];}
               if(i==Ist || i==Ind) coff=0.5; else coff=1.0;
```

```
                Lx2-=Mx*exp_phase(x1,y1,z1,ct,fi_r)*coff;
               }
        }
     Lx[fi]=del.x*(Lx1*del.y+Lx2*del.z);
      fout<<fi<<"," << real(Lx[fi])<<", " <<imag(Lx[fi])<<endl;
   }
fout.close();
}
//**********************over*****************
//***************************************************
void find_Ly1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex My, Ly1, Ly2;
writeFile("LY.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Ly1=0; Ly2=0;
    for(j=Jst; j<=Jnd; j++)
       {y1=(j-cen_y)*del.y;
     //***************************
        for(i=Ist; i<=Ind-1; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x+0.5)*del.x;
               if(m==0) {z1=(Kst-cen_z)*del.z; My=-exy[0][i][j];}
               else     {z1=(Knd-cen_z)*del.z; My= exy[1][i][j];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Ly1-=My*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
     //***************************
        for(k=Kst; k<=Knd-1; k++)
           for(m=0; m<=1; m++)
              {z1=(k-cen_z+0.5)*del.z;
               if(m==0) {x1=(Ist-cen_x)*del.x; My= ezy[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; My=-ezy[1][j][k];}
               if(j==Jst || j==Jnd) coff=0.5; else coff=1.0;
               Ly2-=My*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
       }
    Ly[fi]=del.y*(Ly1*del.x+Ly2*del.z);
     fout<<fi<<"," << real(Ly[fi])<<", " <<imag(Ly[fi])<<endl;
   }
```

```
fout.close();
}
//**********************over*****************
//***************************************************
void find_Lz1(double ct)
{int fi, i,j,k,m;
double x1, y1, z1, fi_r, coff;
complex Mz, Lz1, Lz2;
writeFile("LZ.dat", fout);
for(fi=1;fi<=360;fi++)
   {fi_r=fi*dtor;
    Lz1=0; Lz2=0;
    for(k=Kst; k<=Knd; k++)
       {z1=(k-cen_z)*del.z;
     //**************************
        for(j=Jst; j<=Jnd-1; j++)
           for(m=0; m<=1; m++)
              {y1=(j-cen_y+0.5)*del.y;
               if(m==0) {x1=(Ist-cen_x)*del.x; Mz=-eyz[0][j][k];}
               else     {x1=(Ind-cen_x)*del.x; Mz= eyz[1][j][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Lz2-=Mz*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
     //**************************
        for(i=Ist; i<=Ind-1; i++)
           for(m=0; m<=1; m++)
              {x1=(i-cen_x+0.5)*del.x;
               if(m==0) {y1=(Jst-cen_y)*del.y; Mz= exz[0][i][k];}
               else     {y1=(Jnd-cen_y)*del.y; Mz=-exz[1][i][k];}
               if(k==Kst || k==Knd) coff=0.5; else coff=1.0;
               Lz1-=Mz*exp_phase(x1,y1,z1,ct,fi_r)*coff;
              }
       }
    Lz[fi]=del.z*(Lz1*del.x+Lz2*del.y);
     fout<<fi<<"," << real(Lz[fi])<<", " <<imag(Lz[fi])<<endl;
   }
fout.close();
}
//**********************over*****************

************************
```

```
********mis.cpp**********

//**********************
//*** input filename***
//**********************
#include "defin.h"
extern const unsigned NAME_SIZE=64;
extern char * inFile;
extern fstream f;

void getInputFilename( char * inFile, fstream & f)
{Mybool ok;
do{
   ok=true1;
   cout<<"Enter input file  name:";
   cin.getline(inFile, NAME_SIZE);
   f.open(inFile, ios::in);
   if(!f){cout<<"Cannot open the file" <<inFile<< "\n\n";
          ok=false0;}
  }while(!ok);
}

//***************
//***read file***
//***************
void readFile(char * inFile, fstream & f)
{f.open(inFile, ios::in);
 if(!f) {cout<<"Cannot open the material " << inFile<< "\n\n"; exit(0);}
}

//*****************
//***write file***
//*****************
void writeFile(char * inFile, fstream & f)
{f.open(inFile, ios::out);
 if(!f) {cout<<"Cannot open the file " << inFile<< "\n\n"; exit(0);}
}
```

2.6.2 Far Fields

There are two different ways to obtain the far fields from the time-domain fields in the defined space. One is sine steady-state response method [25]. Another is the so-called FFT method [27, 28]. Here the first method is used because it is more efficient when the radiation patterns of antenna are not very sensitive to frequency change where only two or three frequencies are required.

Since the source is the form of sine wave, one can find the amplitude and phase of the fields when steady-state condition is reached. After obtaining the tangential electric and magnetic currents in terms of surface fields on a closed surface surrounding the antenna, one can use the equivalent electric and magnetic currents to compute the corresponding radiated fields in the far zone.

It is assumed that $\vec{J_s} = \hat{n} \times \vec{H}$ and $\vec{M_s} = -\hat{n} \times \vec{E}$, where $\hat{n}$ is the unit normal vector of the surface surrounding the antenna while $\vec{E}$ and $\vec{H}$ are the fields on the surface. The retarded potentials $\vec{F}$ and $\vec{A}$ can be defined in terms of the magnetic source and electric source respectively [7]. For a homogeneous isotropic medium, the relations are

$$\vec{A} = \mu \int_{s'} \frac{\vec{J_s} e^{-ikr}}{4\pi r} ds' \tag{2.38}$$

$$\vec{F} = \epsilon \int_{s'} \frac{\vec{M_s} e^{-ikr}}{4\pi r} ds' \tag{2.39}$$