Broadband analysis of microstrip patch antenna using 3D FDTD - UPML

Srikumar Sandeep
ECEN 5134- Term paper
University of Colorado at Boulder

Abstract – This paper presents the analysis of microstrip patch antenna using the three dimensional Finite Difference Time Domain (FDTD) method. The derivation and implementation of the three dimensional FDTD method is described. This is done along with the implementation of an artificial absorbing boundary medium known as the Uniaxial Perfectly Matched Layer (UPML). This medium enables to limit the computational domain required for simulation. This is followed by the broadband analysis of a microstrip line fed rectangular patch antenna using Gaussian pulse excitation. The broadband analysis is used to estimate the return loss of the line fed antenna. The results are compared with the solution obtained from Ansoft planar EM simulator. Furthermore, the input resistance and reactance of the antenna near resonance are estimated.

I .INTRODUCTION

Microstrip patch antennas are low – profile antennas that are simple and inexpensive to manufacture. They are mechanically robust and are conformable to planar and nonplanar surfaces. One of the main features of the microstrip antenna is the extremely narrow frequency bandwidth. Several methods exist for the analysis of microstrip antennas. They can be classified as the Transmission Line Model (TLM), cavity model and full wave models. Finite Element Method (FEM), FDTD, Integral Equation – Method of Moment (IE – MoM) are some of the prominent full wave models. The primary advantage of FDTD over other full-wave models is that, FDTD being a time domain method allows broadband analysis of the antenna by Gaussian pulse excitation. The antenna characteristics over a wide frequency range can be obtained by taking the Fourier transform of the FDTD simulation results obtained when a wideband Gaussian pulse is used as an excitation. Moreover, most of widely used electromagnetic simulation software's use frequency domain methods like FEM. The disadvantages of the FDTD are high memory and computational requirements. Since the computational power will continue to grow exponentially in the future, this drawback will become less significant.

FDTD has received tremendous attention in the literature recently. FDTD has been used to simulate and analyze a plethora of electromagnetic problems ranging from antennas, microwave wave circuits, electromagnetic compatibility (EMC) issues, bioelectromagnetics, electromagnetic scattering to novel materials and nanophotonics [2]. In this paper, we explore the analysis of microstrip patch antennas using FDTD. The technique described in this paper can be used to analyze other planar microwave circuits such as filters, hybrids, impedance transformers etc. FDTD analysis of microstrip antennas can be found in [3] – [5]. But most of these papers use either Berenger's PML or Mur Absorbing Boundary Conditions (ABC) for terminating the computational domain. In this paper, we use the UPML as the absorbing boundary to terminate the computational domain. UPML is superior in performance to either Berenger's PML or Mur ABC.

II. FINITE DIFFERENCE TIME DOMAIN (FDTD) METHOD

The Finite Difference Time Domain (FDTD) method is a computational electromagnetic method that can be used to simulate any electromagnetic problem. In the FDTD method, Maxwell's curl equations are converted to their corresponding scalar Partial Differential Equations (PDE). This is followed by the discretization of space and time domain. Central difference approximations are applied to the scalar Partial Differential Equations (PDE) with respect to the discretized time and space domain. This will result in discrete equations for each field component, which can be used to evaluate these field components. These equations are called update equations or time – stepping equations. The update equation for a particular field component can be defined as the discrete equation that expresses the future value of the same field component using previous value of the same field component and the spatial derivatives of other field components at the present time.

A. Three dimensional FDTD formulation

One of the most important considerations in FDTD simulation is that of Absorbing Boundary Conditions

(ABC). In solving electromagnetic wave problems, it is assumed that geometries of interest are defined in an open region, i.e. only the geometries of interest are the scattering objects present. But computational resources are limited and hence the spatial domain needs to be truncated in such way that there are no wave reflections from the boundaries. Such boundary conditions are called ABCs. The Uniaxial Perfectly Matched Layer (UPML) is the most efficient of the ABC's available in literature [2]. The computational domain will be surrounded by the UPML, which will absorb waves of any polarization, angle of incidence and frequency.

In FDTD – UPML formulation, the entire spatial domain is assumed to be an anisotropic medium [2]. Maxwell's curl equations in the anisotropic medium can be expressed as follows

$$\nabla \times \vec{\vec{H}} = j\omega\varepsilon\bar{\bar{s}}\vec{\vec{E}}$$
$$\nabla \times \vec{\vec{E}} = -j\omega\mu\bar{\bar{s}}\vec{\vec{H}} \quad (1)$$

where $\vec{\vec{E}}$, $\vec{\vec{H}}$ are the electric and magnetic field intensity vectors with components in phasor form and $\bar{\bar{s}}$ is the diagonal tensor given by (2).

$$\bar{\bar{s}} = \begin{bmatrix} \frac{s_y s_z}{s_x} & 0 & 0 \\ 0 & \frac{s_x s_z}{s_y} & 0 \\ 0 & 0 & \frac{s_x s_y}{s_z} \end{bmatrix} \quad (2)$$

In (2), s_x, s_y and s_z are the relative complex permittivities along x, y and z directions. They are given by (3).

$$s_x = \kappa_x + \frac{\sigma_x}{j\omega\varepsilon}$$
$$s_y = \kappa_y + \frac{\sigma_y}{j\omega\varepsilon}$$
$$s_z = \kappa_z + \frac{\sigma_z}{j\omega\varepsilon} \quad (3)$$

The first equation of (1) is a vector PDE and hence it can be expanded into three scalar PDEs as below.

$$\begin{bmatrix} \frac{\partial \overline{H_z}}{\partial y} - \frac{\partial \overline{H_y}}{\partial z} \\ \frac{\partial \overline{H_x}}{\partial z} - \frac{\partial \overline{H_z}}{\partial x} \\ \frac{\partial \overline{H_y}}{\partial x} - \frac{\partial \overline{H_x}}{\partial y} \end{bmatrix} = j\omega\varepsilon \begin{bmatrix} \frac{s_y s_z}{s_x} & 0 & 0 \\ 0 & \frac{s_x s_z}{s_y} & 0 \\ 0 & 0 & \frac{s_x s_y}{s_z} \end{bmatrix} \begin{bmatrix} \overline{E_x} \\ \overline{E_y} \\ \overline{E_z} \end{bmatrix} \quad (4)$$

The electric flux density components are related to the electric field intensity components as shown in (5).

$$\overline{D_x} = \varepsilon \frac{s_z}{s_x} \overline{E_x}$$
$$\overline{D_y} = \varepsilon \frac{s_x}{s_y} \overline{E_y}$$
$$\overline{D_z} = \varepsilon \frac{s_y}{s_z} \overline{E_z} \quad (5)$$

Substituting (5) in (4) results in (6)

$$\begin{bmatrix} \frac{\partial \overline{H_z}}{\partial y} - \frac{\partial \overline{H_y}}{\partial z} \\ \frac{\partial \overline{H_x}}{\partial z} - \frac{\partial \overline{H_z}}{\partial x} \\ \frac{\partial \overline{H_y}}{\partial x} - \frac{\partial \overline{H_x}}{\partial y} \end{bmatrix} = j\omega \begin{bmatrix} s_y & 0 & 0 \\ 0 & s_z & 0 \\ 0 & 0 & s_x \end{bmatrix} \begin{bmatrix} \overline{D_x} \\ \overline{D_y} \\ \overline{D_z} \end{bmatrix} \quad (6)$$

The PDEs in (6) are in frequency domain. They are converted to time domain, by using the transformation, $j\omega \rightarrow \frac{\partial}{\partial t}$. This is followed by substituting (3) in (6). The resultant time – domain PDEs relating magnetic field intensity components and electric flux density components are as follows.

$$\begin{bmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{bmatrix} = \frac{\partial}{\partial t} \begin{bmatrix} \kappa_y & 0 & 0 \\ 0 & \kappa_z & 0 \\ 0 & 0 & \kappa_x \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} + \frac{1}{\varepsilon} \begin{bmatrix} \sigma_y & 0 & 0 \\ 0 & \sigma_z & 0 \\ 0 & 0 & \sigma_x \end{bmatrix} \begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} \quad (7)$$

The second equation of (1) can be treated similarly leading to (8).

$$\begin{bmatrix} \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \\ \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} \\ \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \end{bmatrix} = -\frac{\partial}{\partial t} \begin{bmatrix} \kappa_y & 0 & 0 \\ 0 & \kappa_z & 0 \\ 0 & 0 & \kappa_x \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} - \frac{1}{\varepsilon} \begin{bmatrix} \sigma_y & 0 & 0 \\ 0 & \sigma_z & 0 \\ 0 & 0 & \sigma_x \end{bmatrix} \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \quad (8)$$

Thus (7) and (8) are time – domain PDEs relating magnetic field intensity to electric flux density and electric field intensity to magnetic flux density respectively. It should be noted that in (7) and (8), the quantities are in time – domain, and hence they are represented without bar.

Using (5), we can derive time – domain PDEs relating electric field intensity components to electric flux density components. The first equation of (5) is expanded as follows.

$$\left(\kappa_x + \frac{\sigma_x}{j\omega\varepsilon}\right)\overline{D_x} = \varepsilon\left(\kappa_z + \frac{\sigma_z}{j\omega\varepsilon}\right)\overline{E_x} \quad (9)$$

Multiplying both sides of (8) by $j\omega$ and transforming to time – domain leads to (10). A similar treatment of the remaining equations in (5) will result in (11) and (12).

$$\frac{\partial}{\partial t}(\kappa_x D_x) + \frac{\sigma_x}{\varepsilon}D_x = \varepsilon\left[\frac{\partial}{\partial t}(\kappa_z E_x) + \frac{\sigma_z}{\varepsilon}E_x\right] \quad (10)$$

$$\frac{\partial}{\partial t}(\kappa_y D_y) + \frac{\sigma_y}{\varepsilon}D_y = \varepsilon\left[\frac{\partial}{\partial t}(\kappa_x E_y) + \frac{\sigma_x}{\varepsilon}E_y\right] \quad (11)$$

$$\frac{\partial}{\partial t}(\kappa_z D_z) + \frac{\sigma_z}{\varepsilon}D_z = \varepsilon\left[\frac{\partial}{\partial t}(\kappa_y E_z) + \frac{\sigma_y}{\varepsilon}E_z\right] \quad (12)$$

Similarly time – domain PDEs relating magnetic field intensities and flux densities can be derived from their constitutive relations. They are given by (13), (14) and (15).

$$\frac{\partial}{\partial t}(\kappa_x B_x) + \frac{\sigma_x}{\varepsilon}B_x = \mu\left[\frac{\partial}{\partial t}(\kappa_z H_x) + \frac{\sigma_z}{\varepsilon}H_x\right] \quad (13)$$

$$\frac{\partial}{\partial t}(\kappa_y B_y) + \frac{\sigma_y}{\varepsilon}B_y = \mu\left[\frac{\partial}{\partial t}(\kappa_x H_y) + \frac{\sigma_x}{\varepsilon}H_y\right] \quad (14)$$

$$\frac{\partial}{\partial t}(\kappa_z B_z) + \frac{\sigma_z}{\varepsilon}B_z = \mu\left[\frac{\partial}{\partial t}(\kappa_y H_z) + \frac{\sigma_y}{\varepsilon}H_z\right] \quad (15)$$

B. Discretization of space and time domain

Spatial discretization is done by dividing the computational space into cuboidal cells called Yee – cells. The Yee – cell is shown in figure 1.

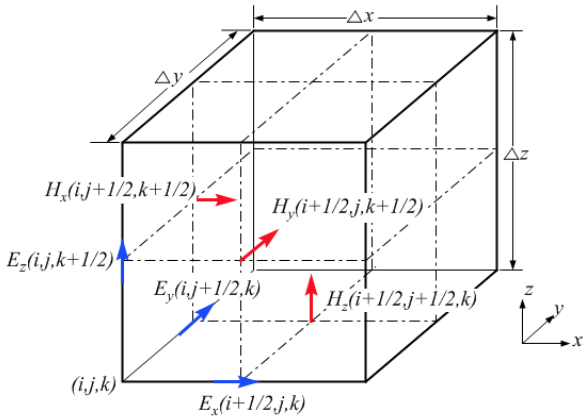


Figure 1. Electric and magnetic field vector components in a Yee – cell.

From figure 1, it can be seen that electric field components are located at Yee – cell edge centers and magnetic field components are located at face centers. The computation space is assumed to be filled with a number of Yee – cells. Each Yee – cell is identified by the spatial discretization indices (i, j, k). The indices (i, j, k) are mapped to the physical space as per the following relation.

$$(i, j, k) \rightarrow (i\Delta x, j\Delta y, k\Delta z) \quad (16)$$

where $\Delta x, \Delta y$ and Δz are the dimensions of the Yee – cell along the x, y and z axis respectively. Temporal discretization is done by dividing the time axis into segments of duration Δt each. The temporal discretization index n corresponds to the real time, $n\Delta t$.

In order to apply central difference approximations to the differential equations, the magnetic and electric field vector components are assigned to the discrertized space and time domain, in such a way that they are interleaved in both space and time. Spatial interleaving can be seen in figure. In order to have temporal interleaving, electric field / electric flux components are evaluated at $..n\Delta t, (n+1)\Delta t..$ and magnetic field / magnetic flux components are evaluated at $.(n-0.5)\Delta t, (n+0.5)\Delta t...$For convenience we will use a shorthand notation to represent the field components at a particular time. Two examples of this notation are given below.

$$E_z|^n_{i,j,k+0.5} \rightarrow E_z(i\Delta x, j\Delta y, k\Delta z + 0.5\Delta z\,; n\Delta t)$$
$$H_y|^{n+0.5}_{i+0.5,j,k+0.5} \rightarrow H_y(i\Delta x + 0.5\Delta x, j\Delta y, k\Delta z + 0.5\Delta z\,; n\Delta t + 0.5\Delta t) \quad (17)$$

C. Derivation of FDTD update equations

The three dimensional FDTD is implemented using 12 update equations. They are for $D_x, D_y, D_z, E_x, E_y, E_z, B_x, B_y, B_z, H_x, H_y, H_z$. In this section, the derivation of update equations for D_x and E_x are shown. The rest of the equations can be derived in the same way. The figure 1 and the shorthand notation in (17), should be used to understand the derivation of these update equations.

The starting point for the derivation of update equations are the continuous time PDEs given by (7), (8), (10) – (12), (13) – (15). The equations (7), (8) are used for the derivation of electric and magnetic flux density update equations respectively. (10) – (12), (13) – (15) are used for the derivation of electric and magnetic field intensity update equations. In order to derive D_x update equation, we use the first row of (7). The first row of (7) is given by (18).

$$\frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} = \kappa_y \frac{\partial}{\partial t}(D_x) + \frac{\sigma_y}{\varepsilon}D_x \quad (18)$$

From (18), it is obvious that future value of D_x, depends on the past value of D_x and the spatial derivatives of H_y, H_z at the current time. The D_x update equation will be used to evaluate, the future value of D_x, i.e. $D_x|_{i+0.5,j,k}^{n+1}$. This will depend on the previous value of D_x, i.e. $D_x|_{i+0.5,j,k}^{n}$ and the spatial derivatives of H_z,H_y at the current time, i.e. $n + 0.5$ .The spatial derivatives of H_z, H_y are obtained around the position of D_x, i.e. the same position of E_x in figure 1. The spatial derivatives approximated in such a way using central difference approximation are as follows.

$$\frac{\partial H_z}{\partial y} = \frac{H_z|_{i+0.5,j+0.5,k}^{n+0.5} - H_z|_{i+0.5,j-0.5,k}^{n+0.5}}{\Delta y}$$
$$\frac{\partial H_y}{\partial z} = \frac{H_y|_{i+0.5,j,k+0.5}^{n+0.5} - H_y|_{i+0.5,j,k-0.5}^{n+0.5}}{\Delta z} \quad (19)$$

In (18), the time derivative of D_x is replaced with the central difference approximation of the derivative around current time $n + 0.5$. This is given by

$$\frac{\partial D_x}{\partial t} = \frac{D_x|_{i+0.5,j,k}^{n+1} - D_x|_{i+0.5,j,k}^{n}}{\Delta t} \quad (20)$$

The last term on the right hand side of (18), denotes the value of D_x at the current time, i.e. $D_x|_{i+0.5,j,k}^{n+0.5}$. Since we evaluate the values of D_x at $..n\Delta t, (n+1)\Delta t..$, $D_x|_{i+0.5,j,k}^{n+0.5}$ is substituted by the following approximation.

$$D_x|_{i+0.5,j,k}^{n+0.5} = \frac{D_x|_{i+0.5,j,k}^{n} + D_x|_{i+0.5,j,k}^{n+1}}{2} \quad (21)$$

Substituting (19), (20) and (21) in (18) will result in the D_x update equation as shown below.

$$D_x|_{i+0.5,j,k}^{n+1} = C1Dx.D_x|_{i+0.5,j,k}^{n} + C2Dx.\left(\frac{H_z|_{i+0.5,j+0.5,k}^{n+0.5} - H_z|_{i+0.5,j-0.5,k}^{n+0.5}}{\Delta y} - \frac{H_y|_{i+0.5,j,k+0.5}^{n+0.5} - H_y|_{i+0.5,j,k-0.5}^{n+0.5}}{\Delta z}\right)$$
$$C1Dx = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right)$$
$$C2Dx = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right) \quad (22)$$

Similar to the derivation of D_x update equation, update equations for D_y, D_z, B_x, B_y and B_z can be derived.

In order to derive E_x update equation, (10) is used. The left hand side of (10) is substituted with (20) and (21). The right hand side of (10) is substituted with (23) and (24).

$$\frac{\partial E_x}{\partial t} = \frac{E_x|_{i+0.5,j,k}^{n+1} - E_x|_{i+0.5,j,k}^{n}}{\Delta t} \quad (23)$$
$$E_x|_{i+0.5,j,k}^{n+0.5} = \frac{E_x|_{i+0.5,j,k}^{n} + E_x|_{i+0.5,j,k}^{n+1}}{2} \quad (24)$$

This will result in E_x update equation as follows.

$$E_x|_{i+0.5,j,k}^{n+1} = C1Ex.D_x|_{i+0.5,j,k}^{n+1} - C2Ex.D_x|_{i+0.5,j,k}^{n} + C3Ex.E_x|_{i+0.5,j,k}^{n}$$
$$C1Ex = \left(\frac{2\varepsilon\kappa_x + \sigma_x\Delta t}{\varepsilon(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right)$$
$$C2Ex = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{\varepsilon(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right)$$
$$C2Ex = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right) \quad (25)$$

Similar to the derivation of E_x update equation, update equations for E_y, E_z, H_x, H_y and H_z can be derived. All the 12 update equations were carefully derived and are given in appendix I for reference.

D. Computer implementation of FDTD

FDTD is implemented in software by iterating the 12 update equations given in appendix I. In each iteration, the flux / field components in the entire computational space are updated. For instance, the updating of the components can be in this order, *D, E, B, H*. The computational space is divided into two regions, the inner problem space and the UPML region surrounding the problem space. The UPML is terminated by Perfect Electric Conductor (PEC) on all the six sides. Figure 2, shows the cross – sectional view of the computational domain in a constant y plane. The problem space, UPML and PEC boundary can be seen.

In figure 2, it should be noted that the UPML region is divided into different regions. These are the $x_{min}, x_{max}, z_{min}, z_{max}$ UPML slabs and the corner regions. A cross – sectional cut of the domain in a constant z plane will show y_{min}, y_{max} UPML regions. Corner UPML regions are the overlap of two or more UPML slabs. There will be 4 corner regions near the vertices of the computational domain, where x, y an z UPML slabs overlap (i.e. overlap of three UPML regions).

In FDTD, different materials are modeled by varying the values of $\kappa_x, \kappa_y, \kappa_z, \sigma_x, \sigma_y, \sigma_z$. The computational space will be divided into Yee – cells. Each Yee – cell will have its corresponding constitutive parameters. The variations of κ and σ in

different regions of the computational domain are given below.

Problem space : Isotropic materials in problem space are modeled by allowing $\kappa_x = \kappa_y = \kappa_z = 1$, $\sigma_x = \sigma_y = \sigma_x = \sigma_{mat}$, where σ_{mat} is the conductivity of the material. Furthermore, ε in all the update equations should be multiplied by the ε_r of the corresponding Yee – cell, i.e. the ε_r of the material.

x_{min}, x_{max} UPML slabs : $\sigma_y = \sigma_z = 0, \kappa_y = \kappa_z = 1$. σ_x, κ_x varies.

y_{min}, y_{max} UPML slabs : $\sigma_x = \sigma_z = 0, \kappa_x = \kappa_z = 1$. σ_y, κ_y varies.

z_{min}, z_{max} UPML slabs : $\sigma_x = \sigma_y = 0, \kappa_x = \kappa_y = 1$. σ_z, κ_z varies.

Corner UPML regions : Depending on which UPML slabs are overlapping, a combination of two or more of the UPML slab conditions should be used. For instance, when all the UPML slabs overlap in the corners near the computational space vertices, $\sigma_x, \sigma_y, \sigma_z, \kappa_x, \kappa_y, \kappa_z$ will vary. For the corners in figure 2, only $\sigma_y, \sigma_z, \kappa_y, \kappa_z$ varies.

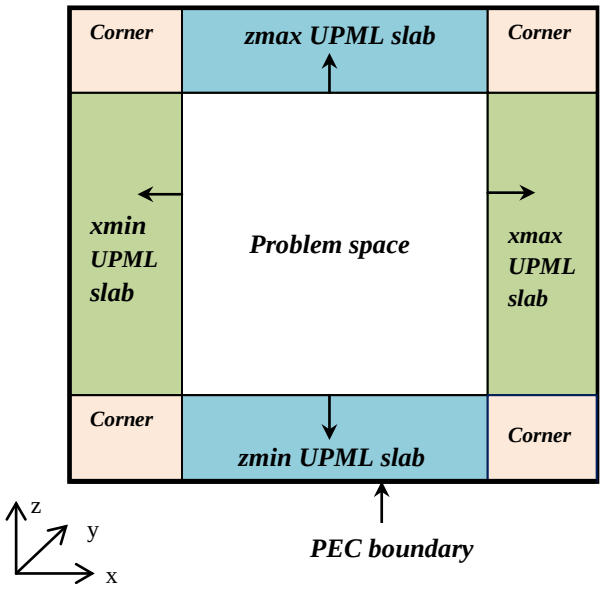


Figure 2. Cross sectional view of the computational space in a plane parallel to XZ plane.

In order to obtain gradual attenuation of any wave incident on the UPML, the parameters σ and κ are geometrically graded along the normal axes of the UPML regions. These normal axis directions of each UPML slab is shown as arrows in figure 2. The geometric grading equations are given by

$$\sigma_{x,y,z}(u) = \left(g^{\frac{1}{\Delta}}\right)^u \sigma_0 \tag{26}$$

$$\kappa_{x,y,z}(u) = \left(g^{\frac{1}{\Delta}}\right)^u \tag{27}$$

where u is the normal distance between the point where the parameter is calculated and the UPML - problem space boundary. Δ is the spatial discretization interval, i.e. $\Delta x = \Delta y = \Delta z = \Delta$. g, σ_o are constants. It should be noted that σ, κ are calculated at the location of the field component. For instance, σ, κ for the update equations of H_x, E_z of the same Yee – cell will be different. This is because the two field components are at different locations in the same Yee – cell. Since the exterior of the UPML is a PEC layer, the tangential electric field components on this surface will be zero. From figure 1, it can be seen that E_x, E_y and E_z on the PEC surface will be zero. This zero electric field condition acts as the stopping criterion for the FDTD update equation loops in software.

E. Precision and stability of FDTD method

The precision and algorithmic stability of the FDTD method is governed by the value of $\Delta x, \Delta y, \Delta z$ and Δt. In this paper, we will be using cubical Yee – cells, so $\Delta x = \Delta y = \Delta z = \Delta$. In order to minimize dispersive effects, (28) should be satisfied [2].

$$\Delta \leq \frac{\lambda_{min}}{10} \tag{28}$$

where λ_{min} is the minimum wavelength corresponding to the maximum frequency of simulation. The time discretization interval, Δt is determined by the Courant condition [2], given by (29).

$$\Delta t \leq \frac{\Delta}{c_o \sqrt{n}} \tag{29}$$

where n is the dimension of the FDTD formulation and c_o is the velocity of light in vacuum. For 3D simulation, it is usual practice to use, $\Delta t = \Delta / 2c_o$.

III. BROADBAND ANALYSIS USING FDTD

Three dimensional FDTD with UPML ABC was implemented in C++. The main objective was to simulate the rectangular patch antenna shown in figure 3. The code is given in appendix II. The code can be used with update equations in appendix I, to gain a clear understanding. The length of the antenna, $L = 16.2\,mm$ and width, $W = 12.45\,mm$. The height of the substrate $h = 0.795\ mm$ and substrate dielectric constant is 2.2. The simulation results can be used to obtain $S_{11}(dB)$ vs. frequency of this

antenna. Port 1 is the port represented by the terminal plane (TP) in figure 3.

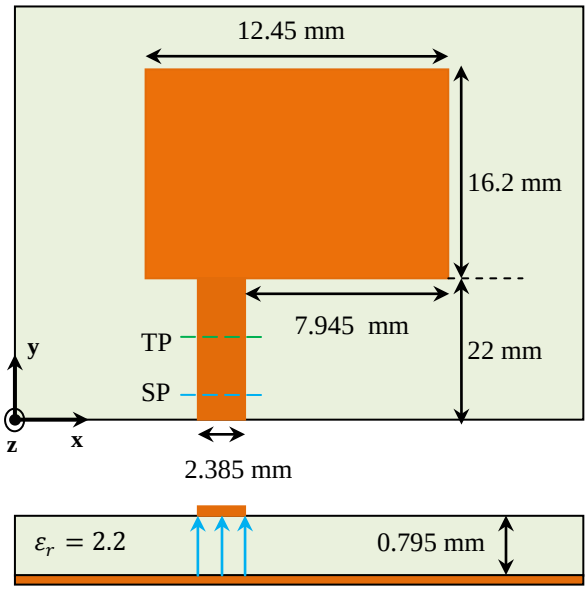


Figure 3. Microstrip line fed rectangular patch antenna (not to scale)

A. FDTD simulation details

The spatial discretization interval used was $\Delta = 0.265\,mm$. Using this interval all the relevant dimensions of the antenna in figure 3 can be represented by integer number of cells. By using (28), it can be seen that this value of Δ will allow the use of high excitation frequencies without causing numerical dispersion. The maximum excitation frequency can easily be more than 50 GHz. The temporal discretization interval is calculated using the Courant condition listed in section II.E. The Δt used for simulation is 0.441 ps.

The UPML parameters g, σ_o given in (26), (27) are found after performing several simulations. They are chosen so that reflection from the problem space – UPML boundary is very low. The following values were found suitable: $g = 1.4, \sigma_o = 0.5$. The number of UPML layers used was 19. Few results of the UPML experiments are given in Appendix III.

The size of the problem space was 70 x 150 x 16 cells. Since there are 19 layers of UPML adjoining each of the six faces of the problem space, the total size of the computational space is 106 x 186 x 52 cells. From appendix I, it can be seen that there are a total of 30 update equation coefficients. In addition to this, there are 12 field / flux components. Therefore the code needs 42 three dimensional matrices, where each matrix is of size 106 x 186 x 52. A floating point number requires 8 bytes of memory for storage. Hence the total amount of RAM required for the simulation was around 330 MB. The simulation environment was Win32 – x86. The microstrip antenna is modeled by modifying the constitutive parameters corresponding to the Yee – cells. The procedure was explained in section II . D. The conductor used as microstrip and ground plane is Copper ($\sigma = 5.8e7$). It should be noted that ground plane fills the entire computational XY plane and the substrate fills the entire problem – space XY plane. The ground plane is one cell thick, while the substrate is 3 cells thick, i.e 3 cells along the Z direction.

B. Estimation of S_{11}

In this section, the procedure used for estimating S_{11} is outlined. Port 1 is associated with the terminal plane (TP) in figure 3. S_{11} can be defined as follows.

$$S_{11}(f) = \frac{V_{ref}(f)}{V_{inc}(f)} = \frac{E_{ref}(f)}{E_{inc}(f)} \qquad (30)$$

The estimation of S_{11} consists of two parts. The objective of the first part is just to sample the incident wave at the terminal plane. In this part, only the microstrip feed line is used. The feed line will extend from $y = 0$ at one end to $y = y_{max}$ at the other end. At both ends the strip will extend into the UPML regions and touch the PEC boundary. If the strip is not extended into UPML, the microstrip discontinuity will result in reflection. The incident wave simulation setup is shown in figure 4. In figure 4, it should be noted that the ground plane extends throughout the XY plane, while the feed line is only 9 cells wide on the XY plane.

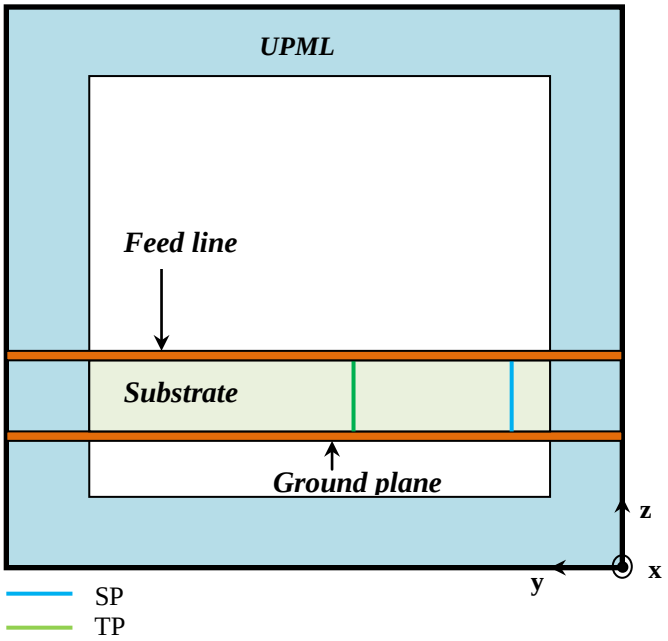


Figure 4. Incident wave simulation setup

Since we are interested in the broadband response of the patch antenna, the excitation source used should be a broadband pulse. A very narrow Gaussian pulse will approximate a time domain impulse function. Therefore the response we obtain from FDTD will approximate impulse response. The frequency response can be obtained by taking the Fourier transform of the response. The excitation pulse that is used for the simulation is a Gaussian pulse of the functional form given below.

$$E_z(t) = exp^{-\left[\left(\frac{t-t_o}{t_w}\right)^2\right]} \quad (31)$$

The Fourier transform of a time – domain Gaussian function is Gaussian in functional form. The lower the value of t_w, larger the frequency bandwidth of the pulse. The system is excited by adding (31) to all the E_z components under the feed line strip in the source plane. Source plane is denoted by SP in figures 3 and 4. The SP is placed 4 cells away from the UPML – problem space boundary. The values used for the simulation are: $t_o = 120\Delta t, t_w = 30\Delta t$. The idea is to generate a TEM wave under the strip which has a Gaussian time signature.

For estimating S_{11}, the incident wave at the terminal plane (TP) is sampled, after excitation at the the source plane. TP is located at about 10 cells away from SP. The transverse field components, i.e. E_z, H_xwere sampled. It was found that there is significant distortion in the E_z field. This is because of the simple source condition that is used. In FDTD, to generate a plane wave, the Total field / Scattered field (TF / SF) formulation is used. By using TF / SF a clean Gaussian plane wave can be generated. However, H_x sampled at the terminal plane showed considerably less distortion. The incident wave at TP is shown in figure 5.

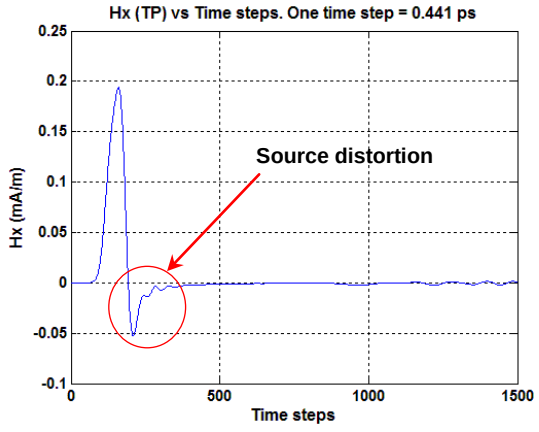


Figure 5. Incident wave at the terminal plane with only the microstrip feed line.

From figure 5, it can be seen that there is some distortion after the trailing edge of the Gaussian pulse. It can be noticed that once pulse has through the terminal plane, the field is nearly zero. This is because of the UPML, which will absorb the pulse. This shows the effectiveness of the UPML.

In the second part, the simulation model is the structure shown in figure 3. From figure 3, it can be seen that the length of the feed line from the UPML – problem space boundary to the edge of the patch antenna is 22 cm. The other end of the feed line is extended into the UPML till it touched PEC boundary at $y = 0$. This will make sure that the only reflection is from that of the patch antenna. The simulation result is shown in figure 6. By comparing figure 6 with 5, it can be seen that after 500 time steps, the incident wave simulation field is nearly zero. That means the pulse has passed the terminal plane. On the other hand, from figure 6, it can be seen that after 500 steps, there is a reflected wave due to microstrip discontinuities. The reflected wave will be completely absorbed by the UPML at the y = 0 end of the feed line. After about 3500 time steps, the terminal plane field settled down to zero. The reflected wave can be obtained by subtracting the incident wave from the total wave. This is depicted in figure 7.

Since we know the incident and reflected waves at the terminal plane, S_{11} can be found out as follows.

$$S_{11}(f) = \frac{H_{ref}(f)}{H_{inc}(f)} = \frac{FT\{H_{ref}(t)\}}{FT\{H_{inc}(t)\}} \quad (32)$$

$S_{11}(dB)$ can be calculated using (33).

$$S_{11}(dB) = 10\log(|S_{11}|) \quad (33)$$

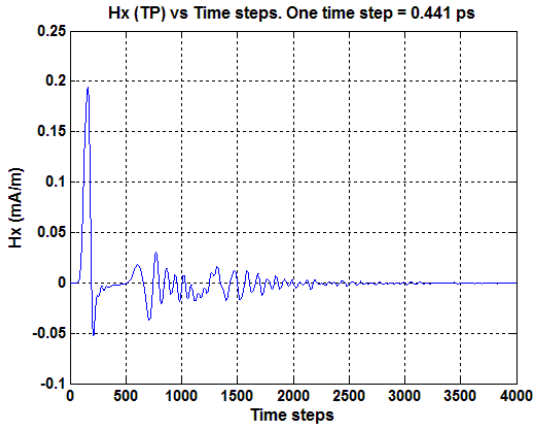


Figure 6. Total wave at the terminal plane with the microstrip patch antenna.

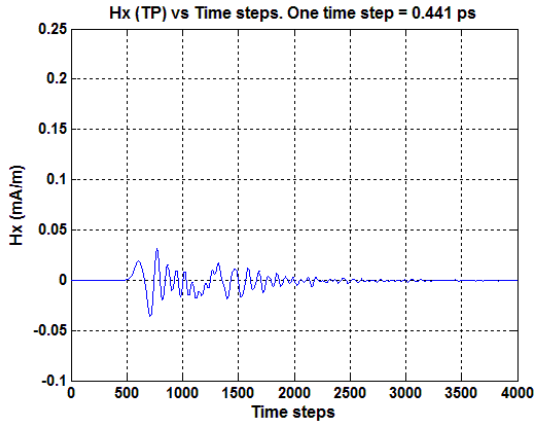


Figure 7. Reflected wave at the terminal plane with the microstrip patch antenna.

MATLAB Fast Fourier Transform (FFT) was used to evaluate (32). Zero padding was used to increase the frequency resolution. MATLAB code written for this purpose is given in appendix IV. The estimated $S_{11}(dB)$ is plotted against frequency in figure 8. In order to validate the result obtained by FDTD simulation, Ansoft planar EM simulator was used to simulate the patch antenna of figure 3. The $S_{11}(dB)$ obtained by this simulation is also shown in figure 7. It can be seen that there is a reasonable match between the results, especially the frequencies at which resonance occurs. A better result can be obtained if there was no source distortion that is seen in figure 5. In order to remove this distortion, more advanced incident source models should be used. In figure 7, only frequencies above 5 GHz are shown. This is because of the erroneous results obtained from FDTD simulation at frequencies below 5 GHz, which can be attributed to source distortion. Three dimensional plots of the patch antenna simulation are given in figure 9.

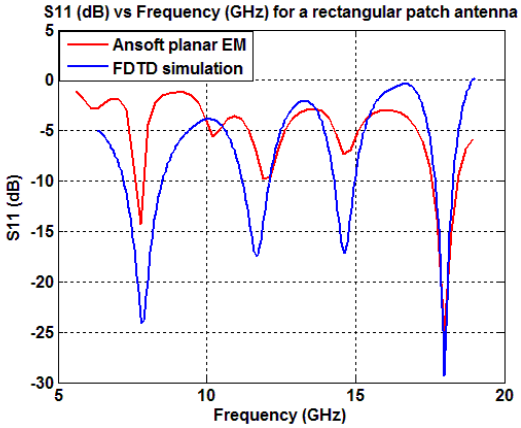


Figure 8. $S_{11}(dB)$ vs. frequency (GHz) of the rectangular patch antenna.

C. Microstrip antenna resonant frequencies

Cavity model of microstrip antennas can be used to obtain approximate values of resonant frequencies. For a microstrip antenna with $L > W > h$, the dominant mode is TM_{010} and its resonant frequency is given by (34).

$$(f_r)_{010} = \frac{v_o}{2L\sqrt{\varepsilon_r}} \tag{34}$$

where v_o is the velocity of light in free space. This value can be evaluated as $(f_r)_{010}$ = 6.25 GHz. This resonance frequency can be seen in the planar EM solution result of figure 8. However from figure 7, it can be noticed that $S_{11}(dB)$ is not low enough at this resonant frequency. Hence the operating resonance of this antenna is the next resonance.

From figure 8, it can be seen that the operating resonance of the antenna is about 7.8 GHz. Both the FDTD and Ansoft planar EM simulation confirms this. The higher order mode after the dominant mode is TM_{020}. The resonant frequency for this mode is given below.

$$(f_r)_{020} = \frac{v_o}{2W\sqrt{\varepsilon_r}} \tag{35}$$

This value is evaluated as $(f_r)_{020}$ = 8.1 GHz. This is slightly different from the estimated value of 7.8 GHz. This may be because the cavity model which is used to derive (35) is an approximate model. The Ansoft simulation and FDTD simulation as full wave models and gives much more accurate results. Microstrip antennas resemble dielectric filled rectangular cavities and hence they exhibit higher order resonances. These resonant frequencies can be seen in figure 8 at frequencies higher than 7.8 GHz.

D. Estimation of antenna input impedance around resonance

In this section, the input impedance of the patch antenna around resonance is estimated. In the last section, we estimated the S_{11} at the terminal plane. Input impedance at the terminal plane is given by (36).

$$Z_{in}^{TP} = Z_o\left(\frac{1+S_{11}}{1-S_{11}}\right) \tag{36}$$

where $Z_o = 50\ \Omega$ is the characteristic impedance of the feed line. The input impedance around the resonant frequency of 7.8 GHz was estimated. The input resistance and reactance versus frequency are shown in figure 10 and 11. From figure 10, it can be seen that at the resonant frequency of 7.8 GHz, the input resistance is about 46 Ω.

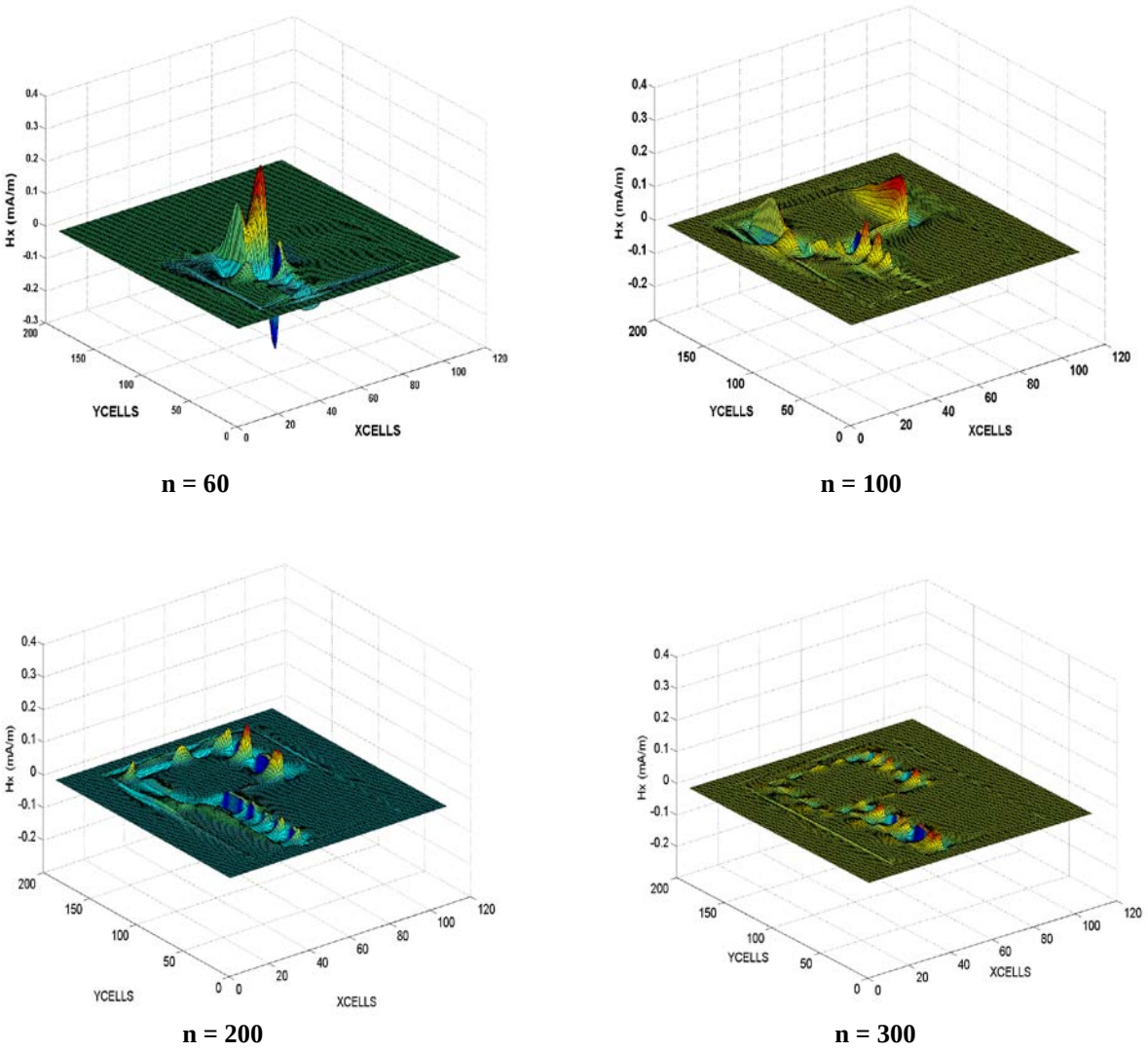


Figure 9 . Microstrip patch antenna simulation : Three dimensional plots of H_x in the computational space at different times

This is close to feed line characteristic impedance of 50 Ω resulting in low value of S_{11}. From figure 11, it can be noticed that input reactance at the resonant frequency is zero. This is typical for resonance.

Once we know the input impedance at the terminal plane, the input impedance of the antenna, i.e. the input impedance at the antenna – feed line junction can be found out using the transmission line impedance transformation equation. The antenna input impedance, Z_{in} can be evaluated as follows.

$$Z_{in} = Z_o \left[\frac{Z_{in}^{TP} - j Z_o tan\beta l}{Z_o - j Z_{in}^{TP} tan\beta l}\right] \quad (37)$$

where Z_o is the characteristic impedance of the microstrip feed line.

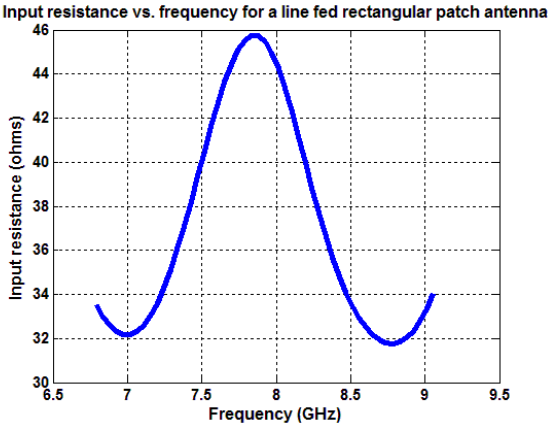


Figure 10. Input resistance vs. frequency (GHz) for the line fed rectangular patch antenna

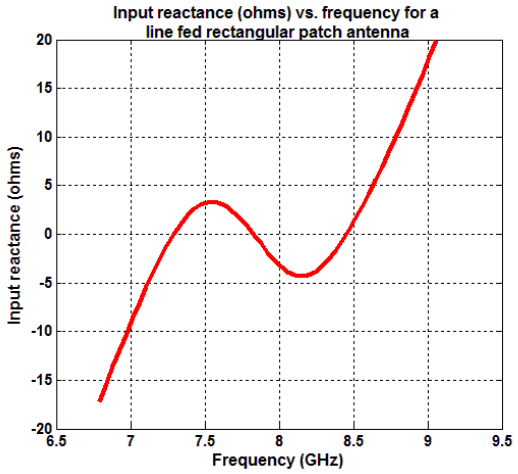


Figure 11. Input reactance vs. frequency (GHz) for the line fed rectangular patch antenna.

β ($\frac{rad}{m}$) is the phase constant of the feed line and l is the distance of the antenna – feed line junction from the terminal plane, i.e. l = 18.55 mm. For a microstrip line of width 2.385 mm on a substrate of thickness h = 0.795 mm and dielectric constant 2.2, the characteristic impedance Z_o = 50.5 Ω. This was verified using Ansoft designer SV. In order to find β, we need to know the phase velocity in the transmission feed line. To calculate the phase velocity, the effective dielectric constant of the feed line, ε_{reff} is found out as follows.

$$\varepsilon_{reff} = \left(\frac{\varepsilon_r+1}{2}\right) + \left(\frac{\varepsilon_r-1}{2}\right)\frac{1}{\sqrt{1+12h/W_f}} \quad (38)$$

where $\varepsilon_r = 2.2$, $h = 0.795$ mm and $W_f = 2.385$ mm. ε_{reff} was found to be 1.86.

The phase constant can be calculated as follows.

$$\beta = \frac{2\pi}{\lambda} = \frac{2\pi f}{v_p} \quad (39)$$

where v_p is the phase velocity calculated using (40).

$$v_p = \frac{c}{\sqrt{\varepsilon_{reff}}} \quad (40)$$

Once β for varying frequency is known, it can be substituted in (37) to estimate antenna input impedance.

IV. CONCLUSIONS

In this paper, we described the three dimensional FDTD – UPML and used it to perform broadband analysis of microstrip patch antenna. The results obtained showed reasonable match with the Ansoft designer simulated results. Better results can be easily obtained by using TF / SF for plane wave generation. The work can be extended to the analysis of fractal patch antennas such as Minkowski fractal square patch antennas. Furthermore, radiation patterns of antennas can be evaluated by using Near Field – Far Field (NF – FF) transformation.

REFERENCES

[1] C. A. Balanis, *Antenna Theory: Analysis and Design*, 3rd Edition, John Wiley & Sons 2005.

[2] A. Taflove and S. Hagness, *Computational Electrodynamics : The Finite – Difference Time – Domain method*, 3rd Edition, Artech House, Inc., 2005.

[3] C. Wu, K. Wu, Z. Bi and J. Litva, "Accurate Characterization of Planar Printed Antennas using Finite – Difference Time – Domain method, " *IEEE Transactions on Antennas and Propagation*, vol. 40, May 1992.

[4] M. Zweki, R. A. Abd – Alhameed, M. A. Mangoud, P. S. Excell, and J. A. Vaul, " Broadband Analysis of Finite Microstrip Patch Antenna Structure using FDTD ", *11th International Conference on Antennas and Propagation*, UMIST, Manchester, UK, April 2001.

[5] D. M. Sheen, S. M. Ali, M. D. Abouzahra and J. A. Kong, "Application of the Three – Dimensional Finite – Difference Time – Domain Method to the Analysis of Planar Microstrip Circuits", *IEEE Transactions on Microwave Theory and Techniques*, vol. 38, No. 7, July 1990.

Appendix I : 3D FDTD – UPML update equations

1. **D_x update equation**

$$D_x|^{n+1}_{i+0.5,j,k} = C1Dx.D_x|^{n}_{i+0.5,j,k} + C2Dx.\left(\frac{H_z|^{n+0.5}_{i+0.5,j+0.5,k} - H_z|^{n+0.5}_{i+0.5,j-0.5,k}}{\Delta y} - \frac{H_y|^{n+0.5}_{i+0.5,j,k+0.5} - H_y|^{n+0.5}_{i+0.5,j,k-0.5}}{\Delta z}\right)$$

$$C1Dx = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right) \qquad C2Dx = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right)$$

2. **D_y update equation**

$$D_y|^{n+1}_{i,j+0.5,k} = C1Dy.D_y|^{n}_{i,j+0.5,k} + C2Dy.\left(\frac{H_x|^{n+0.5}_{i,j+0.5,k+0.5} - H_x|^{n+0.5}_{i,j+0.5,k-0.5}}{\Delta z} - \frac{H_z|^{n+0.5}_{i+0.5,j+0.5,k} - H_z|^{n+0.5}_{i-0.5,j+0.5,k}}{\Delta x}\right)$$

$$C1Dy = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right) \qquad C2Dy = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right)$$

3. **D_z update equation**

$$D_z|^{n+1}_{i,j,k+0.5} = C1Dz.D_z|^{n}_{i,j,k+0.5} + C2Dz.\left(\frac{H_y|^{n+0.5}_{i+0.5,j,k+0.5} - H_y|^{n+0.5}_{i-0.5,j,k+0.5}}{\Delta x} - \frac{H_x|^{n+0.5}_{i,j+0.5,k+0.5} - H_x|^{n+0.5}_{i,j-0.5,k+0.5}}{\Delta y}\right)$$

$$C1Dz = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right) \qquad C2Dz = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right)$$

4. **E_x update equation**

$$E_x|^{n+1}_{i+0.5,j,k} = C1Ex.E_x|^{n}_{i+0.5,j,k} + C2Ex.D_x|^{n+1}_{i+0.5,j,k} - C3Ex.D_x|^{n}_{i+0.5,j,k}$$

$$C1Ex = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right) \quad C2Ex = \left(\frac{2\varepsilon\kappa_x + \sigma_x\Delta t}{\varepsilon(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right) \quad C3Ex = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{\varepsilon(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right)$$

5. **E_y update equation**

$$E_y|^{n+1}_{i,j+0.5,k} = C1Ey.E_y|^{n}_{i,j+0.5,k} + C2Ey.D_y|^{n+1}_{i,j+0.5,k} - C3Ey.D_y|^{n}_{i,j+0.5,k}$$

$$C1Ey = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right) \quad C2Ey = \left(\frac{2\varepsilon\kappa_y + \sigma_y\Delta t}{\varepsilon(2\varepsilon\kappa_x + \sigma_x\Delta t)}\right) \quad C3Ey = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{\varepsilon(2\varepsilon\kappa_x + \sigma_x\Delta t)}\right)$$

6. **E_z update equation**

$$E_z|^{n+1}_{i,j,k+0.5} = C1Ez.E_z|^{n}_{i,j,k+0.5} + C2Ez.D_z|^{n+1}_{i,j,k+0.5} - C3Ez.D_z|^{n}_{i,j,k+0.5}$$

$$C1Ez = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right) \quad C2Ez = \left(\frac{2\varepsilon\kappa_z + \sigma_z\Delta t}{\varepsilon(2\varepsilon\kappa_y + \sigma_y\Delta t)}\right) \quad C3Ez = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{\varepsilon(2\varepsilon\kappa_y + \sigma_y\Delta t)}\right)$$

7. **B_x update equation**

$$B_x|^{n+1.5}_{i,j+0.5,k+0.5} = C1Bx.B_x|^{n+0.5}_{i,j+0.5,k+0.5} - C2Bx.\left(\frac{E_z|^{n+1}_{i,j+1,k+0.5} - E_z|^{n+1}_{i,j,k+0.5}}{\Delta y} - \frac{E_y|^{n+1}_{i,j+0.5,k+1} - E_y|^{n+1}_{i,j+0.5,k}}{\Delta z}\right)$$

$$C1Bx = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right) \qquad C2Bx = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right)$$

8. **B_y update equation**

$$B_y|^{n+1.5}_{i+0.5,j,k+0.5} = C1By.B_y|^{n+0.5}_{i+0.5,j,k+0.5} - C2By.\left(\frac{E_x|^{n+1}_{i+0.5,j,k+1} - E_x|^{n+1}_{i+0.5,j,k}}{\Delta z} - \frac{E_z|^{n+1}_{i+1,j,k+0.5} - E_z|^{n+1}_{i,j,k+0.5}}{\Delta x}\right)$$

$$C1By = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right) \qquad C2By = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right)$$

9. **B_z update equation**

$$B_z|^{n+1.5}_{i+0.5,j+0.5,k} = C1Bz.B_z|^{n+0.5}_{i+0.5,j+0.5,k} - C2Bz.\left(\frac{E_y|^{n+1}_{i+1,j+0.5,k} - E_y|^{n+1}_{i,j+0.5,k}}{\Delta x} - \frac{E_x|^{n+1}_{i+0.5,j+1,k} - E_x|^{n+1}_{i+0.5,j,k}}{\Delta y}\right)$$

$$C1Bz = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right) \qquad C2Bz = \left(\frac{2\varepsilon\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right)$$

10. **H_x update equation**

$$H_x|^{n+1.5}_{i,j+0.5,k+0.5} = C1Hx.H_x|^{n+0.5}_{i,j+0.5,k+0.5} + C2Hx.B_x|^{n+1.5}_{i,j+0.5,k+0.5} - C3Hx.B_x|^{n+0.5}_{i,j+0.5,k+0.5}$$

$$C1Hx = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{2\varepsilon\kappa_z + \sigma_z\Delta t}\right) \quad C2Hx = \left(\frac{2\varepsilon\kappa_x + \sigma_x\Delta t}{\mu(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right) \quad C3Hx = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{\mu(2\varepsilon\kappa_z + \sigma_z\Delta t)}\right)$$

11. H_y update equation

$$H_y|_{i+0.5,j,k+0.5}^{n+1.5} = C1Hy.H_y|_{i+0.5,j,k+0.5}^{n+0.5} + C2Hy.B_y|_{i+0.5,j,k+0.5}^{n+1.5} - C3Hy.B_y|_{i+0.5,j,k+0.5}^{n+0.5}$$

$$C1Hy = \left(\frac{2\varepsilon\kappa_x - \sigma_x\Delta t}{2\varepsilon\kappa_x + \sigma_x\Delta t}\right) \quad C2Hy = \left(\frac{2\varepsilon\kappa_y + \sigma_y\Delta t}{\mu(2\varepsilon\kappa_x + \sigma_x\Delta t)}\right) \quad C3Hy = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{\mu(2\varepsilon\kappa_x + \sigma_x\Delta t)}\right)$$

12. H_z update equation

$$H_z|_{i+0.5,j+0.5,k}^{n+1.5} = C1Hz.H_z|_{i+0.5,j+0.5,k}^{n+0.5} + C2Hz.B_z|_{i+0.5,j+0.5,k}^{n+1.5} - C3Hz.B_z|_{i+0.5,j+0.5,k}^{n+0.5}$$

$$C1Hz = \left(\frac{2\varepsilon\kappa_y - \sigma_y\Delta t}{2\varepsilon\kappa_y + \sigma_y\Delta t}\right) \quad C2Hz = \left(\frac{2\varepsilon\kappa_z + \sigma_z\Delta t}{\mu(2\varepsilon\kappa_y + \sigma_y\Delta t)}\right) \quad C3Hz = \left(\frac{2\varepsilon\kappa_z - \sigma_z\Delta t}{\mu(2\varepsilon\kappa_y + \sigma_y\Delta t)}\right)$$

Appendix II : C++ code for 3D FDTD - UPML simulation of microstrip patch antenna

```
/*  Fdtd3D.cpp : 3D FDTD + UPML
    Author : Srikumar Sandeep

1)     E,D vectors on yee cell edge centres and H,B vectors
       on Yee cell face centres.

       Ex[i + 0.5][j][k], Ey[i][j + 0.5][k], Ez[i][j][k + 0.5]
       Dx[i + 0.5][j][k], Dy[i][j + 0.5][k], Dz[i][j][k + 0.5]

       Hx[i][j + 0.5][k + 0.5], Hy[i + 0.5][j][k + 0.5], Hz[i + 0.5][j + 0.5][k]
       Bx[i][j + 0.5][k + 0.5], By[i + 0.5][j][k + 0.5], Bz[i + 0.5][j + 0.5][k]

2)     Constitutive parameter matrices
       epsr [XCELLS] [YCELLS] [ZCELLS]
       sigma[XCELLS] [YCELLS] [ZCELLS]
*/

#include "iostream.h"
#include "math.h"
#include "stdlib.h"
#include "fstream.h"

#define PL   19  //Number of PML layers.
#define PSX  70  //Number of yee cells in problem space along x-direction
#define PSY  150 //Number of yee cells in problem space along y-direction
#define PSZ  16  //Number of yee cells in problem space along z-direction

#define XCELLS  PSX + (2 * PL)  //Total number of yee cells along x-direction
#define YCELLS  PSY + (2 * PL)  //Total number of yee cells along y-direction
#define ZCELLS  PSZ + (2 * PL)  //Total number of yee cells along z-direction

typedef double*** MatPtr;       //3D matrix

const double muo     = 12.5664e-7;
const double epso    = 8.85e-12;
const double CuSigma = 5.8e78;

//Matrices
MatPtr epsr, sigma;
MatPtr Dx,Dy,Dz,Bx,By,Bz,Ex,Ey,Ez,Hx,Hy,Hz;
MatPtr C1Dx,C2Dx,C1Dy,C2Dy,C1Dz,C2Dz;
MatPtr C1Bx,C2Bx,C1By,C2By,C1Bz,C2Bz;
MatPtr C1Ex,C2Ex,C3Ex,C1Ey,C2Ey,C3Ey,C1Ez,C2Ez,C3Ez;
MatPtr C1Hx,C2Hx,C3Hx,C1Hy,C2Hy,C3Hy,C1Hz,C2Hz,C3Hz;

//Function prototypes
void  CreateAllMatrices();
void  DeleteAllMatrices();
void  CreateMatrix(MatPtr&,int dimx,int dimy,int dimz);
void  DeleteMatrix(MatPtr, int dimx,int dimy,int dimz);
void  FillUpdateEqnConstantMatrices(double g, double sigmao,double dt);

void main()
{
       cout<<"--------------FDTD 3D simulator--------------------\n";

       //T     - Number of time steps
       //i,j,k - x,y,z direction yee cell index
       //n     - Temporal index
       int T = 5000;
```

```
int i,j,k,n;

//Text files to write the simulation data
ofstream Ezfile,Hxfile;
Ezfile.open("Ezfile.txt");
Hxfile.open("Hxfile.txt");

 //dx, dy, dz ,dt – Spatial and temporal discretization interval
double  dx,dy,dz,dt;

//UPML parameters
double  g      = 1.4;
double  sigmao = 0.5;

// Source plane at j = SP
// Terminal plane at j = TP
// GP – Ground plane location at k = PL + 1
int SP = 22;
int TP = 32;
int GP = PL + 1;

//Initiliazation of discretization interval
dx = 0.000265;
dy = 0.000265;
dz = 0.000265;
dt = dz/(6e8);

// Create all the 3D matrices and initialize the elements to zero
 // matrices include field / flux components and precomputed update equation
// coefficients
CreateAllMatrices();

//Fill the constitutive parameter matrices
for(i = 0;i < XCELLS;i++)
{
       for(j = 0;j < YCELLS;j++)
       {
              for(k = 0;k < ZCELLS;k++)
              {
                     epsr [i][j][k]  = 1.0;
                     sigma[i][j][k]  = 0.0;

                     //Ground plane at k = 17
                     //Ground plane extends throughout the entire XY plane
                     if(k == GP && i >= 0 && i < XCELLS && j >= 0 && j < YCELLS)
                     {
                            sigma[i][j][k] = Cusigma;
                     }

                     //3 layers of substrate corresponding to substrate
                     //thickness = 0.795 mm. The substrate extends in the whole
                     //XY plane of the problem space. But does not extend into
                     //UPML
                     if((k == GP + 1 || k == GP + 2 || k == GP + 3)
                            && i >= PL && i < XCELLS - PL && j >= PL && j <
                            YCELLS - PL)
                     {
                            epsr[i][j][k] = 2.2;
                     }


                     //Feed line width is 9dx
                     //Feed line is 8dx offset from the edge of the patch
```

```
                        //i.e it starts from i = 35
                        if(k == GP + 4)
                        {
                                if(i >= 35 && i <= 43)
                                {
                                        //For incident wave simulation : j >=0 && j <=
                                        //YCELLS - 1
                                        //For patch antenna simulation : j >= 0 && j
                                        //<= 102
                                        if(j >= 0 && j <= 102)
                                        {
                                                sigma[i][j][k] = CuSigma;
                                        }
                                }
                        }

                        //Patch antenna
                        if(k == GP + 4)
                        {
                                //The width of patch = 12.45 mm = 47dx
                                if(i >= 27 && i <= 73)
                                {
                                        //The length of patch = 16 mm = 60 dx
                                        if(j >= 102 && j <= 162)
                                        {
                                                sigma[i][j][k] = CuSigma;
                                        }
                                }
                        }
                }
        }
}

//Precomputer update equation coefficient matrices
FillUpdateEqnConstantMatrices(g,sigmao,dt);

//Temp variables to hold previous field values and field differential values.
double Dxo,Exo,Dyo,Eyo,Dzo,Ezo;
double dHx,dHy,dHz;
double Hxo,Hyo,Hzo,Bxo,Byo,Bzo;
double dEx,dEy,dEz;

//----------------------FDTD update loop starts here--------------------------
for(n = 0;n < T;n++)
{
        //to = 4T
        double temp      = -1*(((n*dt - 120*dt)*(n*dt - 120*dt))/(30*30*dt*dt));
        double source    = exp(temp);
        double Ezinc  = source;

        /*Dx,Ex update
          Dx,Ex[XCELLS][YCELLS + 1][ZCELLS + 1]
          Dx,Ex at j = 0 or j = YCELLS or k = 0 or k = ZCELLS are on the
          PEC surface. So they are zero and need not be calculated.
        */
        for(i = 0;i < XCELLS;i++)
        {
                for(j = 1;j < YCELLS;j++)
                {
                        for(k = 1;k < ZCELLS;k++)
                        {
                                Dxo = Dx[i][j][k];
                                Exo = Ex[i][j][k];
```

```
                        //Dx update equation
                        dHz = (Hz[i][j][k] - Hz[i][j - 1][k]) / dy;
                        dHy = (Hy[i][j][k] - Hy[i][j][k - 1]) / dz;

                        Dx[i][j][k] = C1Dx[i][j][k] * Dxo
                               +  C2Dx[i][j][k] * (dHz - dHy);

                        //Ex update equation
                        Ex[i][j][k] = C2Ex[i][j][k] * Dx[i][j][k]
                                              - C3Ex[i][j][k] * Dxo
                                              + C1Ex[i][j][k] * Exo;
                }
        }
}

/*Dy,Ey update
  Dy,Ey[XCELLS + 1][YCELLS][ZCELLS + 1]
  Dy,Ey at i = 0 or i = XCELLS or k = 0 or k = ZCELLS are on the
  PEC surface. So they are zero and need not be calculated.
*/
for(i = 1;i < XCELLS;i++)
{
        for(j =  0;j < YCELLS;j++)
        {
                for(k = 1;k < ZCELLS;k++)
                {
                        Dyo = Dy[i][j][k];
                        Eyo = Ey[i][j][k];

                        //Dy update equation
                        dHx = (Hx[i][j][k] - Hx[i][j][k - 1]) / dz;
                        dHz = (Hz[i][j][k] - Hz[i - 1][j][k]) / dx;
                        Dy[i][j][k] = C1Dy[i][j][k] * Dyo
                               +  C2Dy[i][j][k] * (dHx - dHz);

                        //Ey update equation
                        Ey[i][j][k] = C2Ey[i][j][k] * Dy[i][j][k]
                                              - C3Ey[i][j][k] * Dyo
                                              + C1Ey[i][j][k] * Eyo;
                }
        }
}

//Dz,Ez update
for(i = 1;i < XCELLS;i++)
{
        for(j = 1;j < YCELLS;j++)
        {
                for(k = 0;k < ZCELLS;k++)
                {
                        Dzo = Dz[i][j][k];
                        Ezo = Ez[i][j][k];

                        //Dy update equation
                        dHx = (Hx[i][j][k] - Hx[i][j - 1][k]) / dy;
                        dHy = (Hy[i][j][k] - Hy[i - 1][j][k]) / dx;

                        Dz[i][j][k] = C1Dz[i][j][k] * Dzo
                               +  C2Dz[i][j][k] * (dHy - dHx);

                        //Ez update equation
                        Ez[i][j][k] = C2Ez[i][j][k] * Dz[i][j][k]
```

```
                                        - C3Ez[i][j][k] * Dzo
                                        + C1Ez[i][j][k] * Ezo;


                        //Excitation of the all Ez components under the feed
                        //line microstrip at j = Source plane
                        if(j == SP && i >= 35 && i <= 43 && k >= GP + 1 && k
                            <= GP + 3)
                        {
                              Ez[i][j][k] += Ezinc;
                        }

                }
        }
}

//Write simulation data to text file
Hxfile<<Hx[39][TP][GP + 2]<<"\n";
Ezfile<<Ez[35][TP][GP + 2]<<"\n";

//Hx,Bx update
for(i = 0;i < XCELLS;i++)
{
        for(int j = 0;j < YCELLS;j++)
        {
                for(int k = 0;k < ZCELLS;k++)
                {
                        Hxo = Hx[i][j][k];
                        Bxo = Bx[i][j][k];

                        dEz = (Ez[i][j + 1][k] - Ez[i][j][k]) / dy;
                        dEy = (Ey[i][j][k + 1] - Ey[i][j][k]) / dz;


                        Bx[i][j][k] = C1Bx[i][j][k] * Bxo - C2Bx[i][j][k] *
                        (dEz - dEy);
                        Hx[i][j][k] = C1Hx[i][j][k] * Hxo + C2Hx[i][j][k] *
                        Bx[i][j][k] - C3Hx[i][j][k] * Bxo;
                }
        }
}


//Hy,By update
for(i = 0;i < XCELLS;i++)
{
        for(int j = 0;j < YCELLS;j++)
        {
                for(int k = 0;k < ZCELLS;k++)
                {
                        Hyo = Hy[i][j][k];
                        Byo = By[i][j][k];

                        dEx = (Ex[i][j][k + 1] - Ex[i][j][k]) / dz;
                        dEz = (Ez[i + 1][j][k] - Ez[i][j][k]) / dx;

                        By[i][j][k] = C1By[i][j][k] * Byo - C2By[i][j][k] *
                        (dEx - dEz);
                        Hy[i][j][k] = C1Hy[i][j][k] * Hyo + C2Hy[i][j][k] *
                        By[i][j][k] - C3Hy[i][j][k] * Byo;

                }
        }
```

```
		}

		//Hz,Bz update
		for(i = 0;i < XCELLS;i++)
		{
			for(int j = 0;j < YCELLS;j++)
			{
				for(int k = 0;k < ZCELLS;k++)
				{
					Hzo = Hz[i][j][k];
					Bzo = Bz[i][j][k];

					dEy = (Ey[i + 1][j][k] - Ey[i][j][k]) / dx;
					dEx = (Ex[i][j + 1][k] - Ex[i][j][k]) / dy;

					Bz[i][j][k] = C1Bz[i][j][k] * Bzo - C2Bz[i][j][k] *
					(dEy - dEx);
					Hz[i][j][k] = C1Hz[i][j][k] * Hzo + C2Hz[i][j][k] *
					Bz[i][j][k] - C3Hz[i][j][k] * Bzo;
				}
			}
		}
	}

	DeleteAllMatrices();

	Hxfile.close();
	Ezfile.close();
}

//-------------------------------------------------------------------------------------
void FillUpdateEqnConstantMatrices(double g, double sigmao, double dt)
{
	double Dx_ky,Dx_sigmay,Dy_kz,Dy_sigmaz,Dz_kx,Dz_sigmax;
	double Bx_ky,Bx_sigmay,By_kz,By_sigmaz,Bz_kx,Bz_sigmax;
	double Ex_kx,Ex_sigmax,Ex_kz,Ex_sigmaz,Ey_ky,Ey_sigmay,
		Ey_kx,Ey_sigmax,Ez_kz,Ez_sigmaz,Ez_ky,Ez_sigmay;
	double Hx_kx,Hx_sigmax,Hx_kz,Hx_sigmaz,Hy_ky,Hy_sigmay,
		Hy_kx,Hy_sigmax,Hz_kz,Hz_sigmaz,Hz_ky,Hz_sigmay;


	//C1Dx,C2Dx,C1Ex,C2Ex,C3Ex
	for(int i = 0;i < XCELLS; i++)
	{
		for(int j = 1;j < YCELLS; j++)
		{
			for(int k = 1;k < ZCELLS; k++)
			{
				Dx_ky     =  1;
				Dx_sigmay =  sigma[i][j][k];
				Ex_kx     =  1;
				Ex_sigmax =  sigma[i][j][k];
				Ex_kz     =  1;
				Ex_sigmaz =  sigma[i][j][k];

				//Dx_ky,Dx_sigmay
				if(j < PL)
				{
					Dx_ky     = pow(g,PL - j);
					Dx_sigmay = sigmao * Dx_ky;
				}
				else if (j >= PL + PSY)
				{
```

```
                        Dx_ky     = pow(g,j - PL - PSY);
                        Dx_sigmay = sigmao * Dx_ky;
                }

                //Ex_kx, Ex_sigmax
                if(i < PL)
                {
                        Ex_kx     = pow(g,PL - i - 0.5);
                        Ex_sigmax = sigmao * Ex_kx;
                }
                else if(i >= PL + PSX)
                {
                        Ex_kx     = pow(g,i - PL - PSX  + 0.5);
                        Ex_sigmax = sigmao * Ex_kx;
                }

                //Ex_kz,Ex_sigmaz
                if(k < PL)
                {
                        Ex_kz     = pow(g,PL - k);
                        Ex_sigmaz = sigmao * Ex_kz;
                }
                else if(k >= PL + PSZ)
                {
                        Ex_kz     = pow(g,k - PL - PSZ);
                        Ex_sigmaz = sigmao * Ex_kz;
                }

                //C1Dx,C2Dx
                C1Dx[i][j][k] = (2  * epso * epsr[i][j][k] * Dx_ky -
                Dx_sigmay * dt)/(2 * epso * epsr[i][j][k] * Dx_ky +
                Dx_sigmay * dt);

                C2Dx[i][j][k] = (2  * epso * epsr[i][j][k] * dt) /(2 * epso
                * epsr[i][j][k] * Dx_ky + Dx_sigmay * dt);

                //C1Ex,C2Ex,C3Ex,C1Ey,C2Ey,C3Ey,C1Ez,C2Ez,C3Ez
                C1Ex[i][j][k] = (2 * epso * epsr[i][j][k]    * Ex_kz -
                Ex_sigmaz * dt) /(2* epso * epsr[i][j][k]    * Ex_kz +
                Ex_sigmaz * dt);

                C2Ex[i][j][k] = (2   * epso * epsr[i][j][k]  * Ex_kx +
                Ex_sigmax * dt)/((2 * epso * epsr[i][j][k]  * Ex_kz +
                Ex_sigmaz * dt)* epso * epsr[i][j][k]);

                C3Ex[i][j][k] = (2   * epso * epsr[i][j][k]  * Ex_kx -
                Ex_sigmax * dt)/((2 * epso * epsr[i][j][k]  * Ex_kz +
                Ex_sigmaz * dt)* epso * epsr[i][j][k]);
        }
    }
}

//C1Dy,C2Dy,C1Ey,C2Ey,C3Ey
for(i = 1;i < XCELLS; i++)
{
    for(int j = 0;j < YCELLS; j++)
    {
        for(int k = 1;k < ZCELLS; k++)
        {

                Dy_kz     =  1;
                Dy_sigmaz =  sigma[i][j][k];
                Ey_ky     =  1;
```

```
                Ey_sigmay =  sigma[i][j][k];
                Ey_kx     =  1;
                Ey_sigmax =  sigma[i][j][k];

                //Dy_kz,Dy_sigmaz
                if(k < PL)
                {
                      Dy_kz     = pow(g,PL - k);
                      Dy_sigmaz = sigmao * Dy_kz;
                }
                else if (k >= PL + PSZ)
                {
                      Dy_kz     = pow(g,k - PL - PSZ);
                      Dy_sigmaz = sigmao * Dy_kz;
                }

                //Ey_ky, Ey_sigmay
                if(j < PL)
                {
                      Ey_ky     = pow(g,PL - j - 0.5);
                      Ey_sigmay = sigmao * Ey_ky;
                }
                else if(j >= PL + PSY)
                {
                      Ey_ky     = pow(g,j - PL - PSY  + 0.5);
                      Ey_sigmay = sigmao * Ey_ky;
                }


                //Ey_kx,Ey_sigmax
                if(i < PL)
                {
                      Ey_kx     = pow(g,PL - i);
                      Ey_sigmax = sigmao * Ey_kx;
                }
                else if(i >= PL + PSX)
                {
                      Ey_kx     = pow(g,i - PL - PSX);
                      Ey_sigmax = sigmao * Ey_kx;
                }

                //C1Dy,C2Dy
                C1Dy[i][j][k] = (2  * epso * epsr[i][j][k] * Dy_kz -
                Dy_sigmaz * dt)/(2 * epso * epsr[i][j][k] * Dy_kz +
                Dy_sigmaz * dt);

                C2Dy[i][j][k] = (2  * epso * epsr[i][j][k] * dt) /(2 * epso
                * epsr[i][j][k] * Dy_kz + Dy_sigmaz * dt);

                //C1Ey,C2Ey,C3Ey
                C1Ey[i][j][k] = (2 * epso * epsr[i][j][k]    * Ey_kx -
                Ey_sigmax * dt)/(2* epso * epsr[i][j][k]    * Ey_kx +
                Ey_sigmax * dt);

                C2Ey[i][j][k] = (2   * epso * epsr[i][j][k]  * Ey_ky +
                Ey_sigmay * dt)/((2 * epso * epsr[i][j][k]  * Ey_kx +
                Ey_sigmax * dt)* epso * epsr[i][j][k]);

                C3Ey[i][j][k] = (2   * epso * epsr[i][j][k]  * Ey_ky -
                Ey_sigmay * dt)/((2 * epso * epsr[i][j][k]  * Ey_kx +
                Ey_sigmax * dt)* epso * epsr[i][j][k]);
        }
}
```

```
}

//C1Dz,C2Dz,C1Ez,C2Ez,C3Ez
for(i = 1;i < XCELLS; i++)
{
	for(int j = 1;j < YCELLS; j++)
	{
		for(int k = 0;k < ZCELLS; k++)
		{
			Dz_kx     =  1;
			Dz_sigmax =  sigma[i][j][k];
			Ez_kz     =  1;
			Ez_sigmaz =  sigma[i][j][k];
			Ez_ky     =  1;
			Ez_sigmay =  sigma[i][j][k];

			//Dz_kx,Dz_sigmax
			if(i < PL)
			{
				Dz_kx  = pow(g,PL - i);
				Dz_sigmax = sigmao * Dz_kx;
			}
			else if (i >= PL + PSX)
			{
				Dz_kx     = pow(g,i - PL - PSX);
				Dz_sigmax = sigmao * Dz_kx;
			}

			//Ez_kz, Ez_sigmaz
			if(k < PL)
			{
				Ez_kz     = pow(g,PL - k - 0.5);
				Ez_sigmaz = sigmao * Ez_kz;
			}
			else if(k >= PL + PSZ)
			{
				Ez_kz     = pow(g,k - PL - PSZ + 0.5);
				Ez_sigmaz = sigmao * Ez_kz;
			}


			//Ez_ky,Ez_sigmay
			if(j < PL)
			{
				Ez_ky     = pow(g,PL - j);
				Ez_sigmay = sigmao * Ez_ky;
			}
			else if(j >= PL + PSY)
			{
				Ez_ky     = pow(g,j - PL - PSY);
				Ez_sigmay = sigmao * Ez_ky;
			}


			//C1Dz,C2Dz
			C1Dz[i][j][k] = (2  * epso * epsr[i][j][k] * Dz_kx -
			Dz_sigmax * dt) /(2 * epso * epsr[i][j][k] * Dz_kx +
			Dz_sigmax * dt);

			C2Dz[i][j][k] = (2  * epso * epsr[i][j][k] * dt) /(2 * epso
			* epsr[i][j][k] * Dz_kx + Dz_sigmax * dt);

			//C1Ez,C2Ez,C3Ez
```

```
                        C1Ez[i][j][k] = (2 * epso * epsr[i][j][k]    * Ez_ky -
                        Ez_sigmay * dt)/(2* epso * epsr[i][j][k]    * Ez_ky +
                        Ez_sigmay * dt);

                        C2Ez[i][j][k] = (2   * epso * epsr[i][j][k]  * Ez_kz +
                        Ez_sigmaz * dt)/((2 * epso * epsr[i][j][k]  * Ez_ky +
                        Ez_sigmay * dt) * epso * epsr[i][j][k]);

                        C3Ez[i][j][k] = (2   * epso * epsr[i][j][k]  * Ez_kz -
                        Ez_sigmaz * dt)/((2 * epso * epsr[i][j][k]  * Ez_ky +
                        Ez_sigmay * dt)* epso * epsr[i][j][k]);
                }
        }
}


//C1Bx,C2Bx,C1Hx,C2Hx,C3Hx
for(i = 0;i < XCELLS;i++)
{
        for(int j = 0;j < YCELLS;j++)
        {
                for(int k = 0;k < ZCELLS;k++)
                {
                        Bx_ky     = 1;
                        Bx_sigmay = sigma[i][j][k];
                        Hx_kx     = 1;
                        Hx_sigmax = sigma[i][j][k];
                        Hx_kz     = 1;
                        Hx_sigmaz = sigma[i][j][k];

                        //Bx_ky,Bx_sigmay
                        if(j < PL)
                        {
                                Bx_ky     = pow(g,PL - j - 0.5);
                                Bx_sigmay = sigmao * Bx_ky;
                        }
                        else if(j >= PL + PSY)
                        {
                                Bx_ky     = pow(g,j - PL - PSY + 0.5);
                                Bx_sigmay = sigmao * Bx_ky;
                        }
                        //}

                        //Hx_kx,Hx_sigmax
                        if(i < PL)
                        {
                                Hx_kx     = pow(g,PL - i);
                                Hx_sigmax = sigmao * Hx_kx;
                        }
                        else if(i >= PL + PSX)
                        {
                                Hx_kx     = pow(g,i - PL - PSX);
                                Hx_sigmax = sigmao * Hx_kx;
                        }

                        //Hx_kz,Hx_sigmaz
                        if(k < PL)
                        {
                                Hx_kz     = pow(g,PL - k - 0.5);
                                Hx_sigmaz = sigmao * Hx_kz;
                        }
                        else if(k >= PL + PSZ)
                        {
```

```
                        Hx_kz     = pow(g,k - PL - PSZ + 0.5);
                        Hx_sigmaz = sigmao * Hx_kz;
                    }


                    //C1Bx,C2Bx,C1By,C2By,C1Bz,C2Bz
                    C1Bx[i][j][k] = (2  * epso * epsr[i][j][k] * Bx_ky -
                    Bx_sigmay * dt)/(2 * epso * epsr[i][j][k] * Bx_ky +
                    Bx_sigmay * dt);

                    C2Bx[i][j][k] = (2  * epso * epsr[i][j][k] * dt)
                    /(2 * epso * epsr[i][j][k] * Bx_ky + Bx_sigmay * dt);

                    //C1Hx,C2Hx,C3Hx,C1Hy,C2Hy,C3Hy,C1Hz,C2Hz,C3Hz
                    C2Hx[i][j][k] = (2   * epso * epsr[i][j][k]  * Hx_kx +
                    Hx_sigmax * dt)/((2 * epso * epsr[i][j][k]  * Hx_kz +
                    Hx_sigmaz * dt)* muo );

                    C3Hx[i][j][k] = (2   * epso * epsr[i][j][k]  * Hx_kx -
                    Hx_sigmax * dt)/((2 * epso * epsr[i][j][k]  * Hx_kz +
                    Hx_sigmaz * dt)* muo );

                    C1Hx[i][j][k] = (2 * epso * epsr[i][j][k]    * Hx_kz -
                    Hx_sigmaz * dt) /(2* epso * epsr[i][j][k]    * Hx_kz +
                    Hx_sigmaz * dt);
                }
        }
}


//C1By,C2By,C1Hy,C2Hy,C3Hy
for(i = 0;i < XCELLS;i++)
{
        for(int j = 0;j < YCELLS;j++)
        {
                for(int k = 0;k < ZCELLS;k++)
                {
                    By_kz     = 1;
                    By_sigmaz = sigma[i][j][k];
                    Hy_ky     = 1;
                    Hy_sigmay = sigma[i][j][k];
                    Hy_kx     = 1;
                    Hy_sigmax = sigma[i][j][k];

                    //By_kz,By_sigmaz
                    if(k < PL)
                    {
                        By_kz     = pow(g,PL - k - 0.5);
                        By_sigmaz = sigmao * By_kz;
                    }
                    else if(k >= PL + PSZ)
                    {
                        By_kz     = pow(g,k - PL - PSZ + 0.5);
                        By_sigmaz = sigmao * By_kz;
                    }

                    //Hy_ky,Hy_sigmay
                    if(j < PL)
                    {
                        Hy_ky     = pow(g,PL - j);
                        Hy_sigmay = sigmao * Hy_ky;
                    }
                    else if(j >= PL + PSY)
```

```
                        {
                                Hy_ky     = pow(g,j - PL - PSY);
                                Hy_sigmay = sigmao * Hy_ky;
                        }
                        //}

                        //Hy_kx,Hy_sigmax
                        if(i < PL)
                        {
                                Hy_kx     = pow(g,PL - i - 0.5);
                                Hy_sigmax = sigmao * Hy_kx;
                        }
                        else if(i >= PL + PSX)
                        {
                                Hy_kx     = pow(g,i - PL - PSX + 0.5);
                                Hy_sigmax = sigmao * Hy_kx;
                        }

                        //C1By,C2By
                        C1By[i][j][k] = (2  * epso * epsr[i][j][k] * By_kz -
                        By_sigmaz * dt)/(2 * epso * epsr[i][j][k] * By_kz +
                        By_sigmaz * dt);

                        C2By[i][j][k] = (2  * epso * epsr[i][j][k] * dt) /(2 * epso
                        * epsr[i][j][k] * By_kz + By_sigmaz * dt);

                        //C1Hy,C2Hy,C3Hy
                        C2Hy[i][j][k] = (2   * epso * epsr[i][j][k]  * Hy_ky +
                        Hy_sigmay * dt)/((2 * epso * epsr[i][j][k]  * Hy_kx +
                        Hy_sigmax * dt)* muo );

                        C3Hy[i][j][k] = (2   * epso * epsr[i][j][k]  * Hy_ky -
                        Hy_sigmay * dt)/((2 * epso * epsr[i][j][k]  * Hy_kx +
                        Hy_sigmax * dt)* muo );

                        C1Hy[i][j][k] = (2 * epso * epsr[i][j][k]    * Hy_kx -
                        Hy_sigmax * dt)/(2* epso * epsr[i][j][k]    * Hy_kx +
                        Hy_sigmax * dt);
                }
        }
}

//C1Bz,C2Bz,C1Hz,C2Hz,C3Hz
for(i = 0;i < XCELLS;i++)
{
        for(int j = 0;j < YCELLS;j++)
        {
                for(int k = 0;k < ZCELLS;k++)
                {
                        Bz_kx     = 1;
                        Bz_sigmax = sigma[i][j][k];
                        Hz_kz     = 1;
                        Hz_sigmaz = sigma[i][j][k];
                        Hz_ky     = 1;
                        Hz_sigmay = sigma[i][j][k];

                        //Bz_kx,Bz_sigmax
                        if(i < PL)
                        {
                                Bz_kx     = pow(g,PL - i - 0.5);
                                Bz_sigmax = sigmao * Bz_kx;
                        }
                        else if(i >= PL + PSX)
```

```
                        {
                                Bz_kx     = pow(g,i - PL - PSX + 0.5);
                                Bz_sigmax = sigmao * Bz_kx;
                        }

                        //Hz_kz,Hz_sigmaz
                        if(k < PL)
                        {
                                Hz_kz     = pow(g,PL - k);
                                Hz_sigmaz = sigmao * Hz_kz;
                        }
                        else if(k >= PL + PSZ)
                        {
                                Hz_kz     = pow(g,k - PL - PSZ);
                                Hz_sigmaz = sigmao * Hz_kz;
                        }


                        //Hz_ky,Hz_sigmay
                        if(j < PL)
                        {
                                Hz_ky     = pow(g,PL - j - 0.5);
                                Hz_sigmay = sigmao * Hz_ky;
                        }
                        else if(j >= PL + PSY)
                        {
                                Hz_ky     = pow(g,j - PL - PSY + 0.5);
                                Hz_sigmay = sigmao * Hz_ky;
                        }
                        //}

                        //C1Bz,C2Bz
                        C1Bz[i][j][k] = (2  * epso * epsr[i][j][k] * Bz_kx -
                        Bz_sigmax * dt)/(2 * epso * epsr[i][j][k] * Bz_kx +
                        Bz_sigmax * dt);

                        C2Bz[i][j][k] = (2  * epso * epsr[i][j][k] * dt) /(2 * epso
                        * epsr[i][j][k] * Bz_kx + Bz_sigmax * dt);

                        //C1Hz,C2Hz,C3Hz
                        C2Hz[i][j][k] = (2   * epso * epsr[i][j][k]  * Hz_kz +
                        Hz_sigmaz * dt)/((2 * epso * epsr[i][j][k]  * Hz_ky +
                        Hz_sigmay * dt)* muo );

                        C3Hz[i][j][k] = (2   * epso * epsr[i][j][k]  * Hz_kz -
                        Hz_sigmaz * dt)/((2 * epso * epsr[i][j][k]  * Hz_ky +
                        Hz_sigmay * dt)* muo );

                        C1Hz[i][j][k] = (2 * epso * epsr[i][j][k]    * Hz_ky -
                        Hz_sigmay * dt)/(2* epso * epsr[i][j][k]    * Hz_ky +
                        Hz_sigmay * dt);
                }
        }
    }
}


void CreateAllMatrices()
{
    //Create the parameter matrices
    CreateMatrix(epsr, XCELLS,YCELLS,ZCELLS);
    CreateMatrix(sigma,XCELLS,YCELLS,ZCELLS);
```

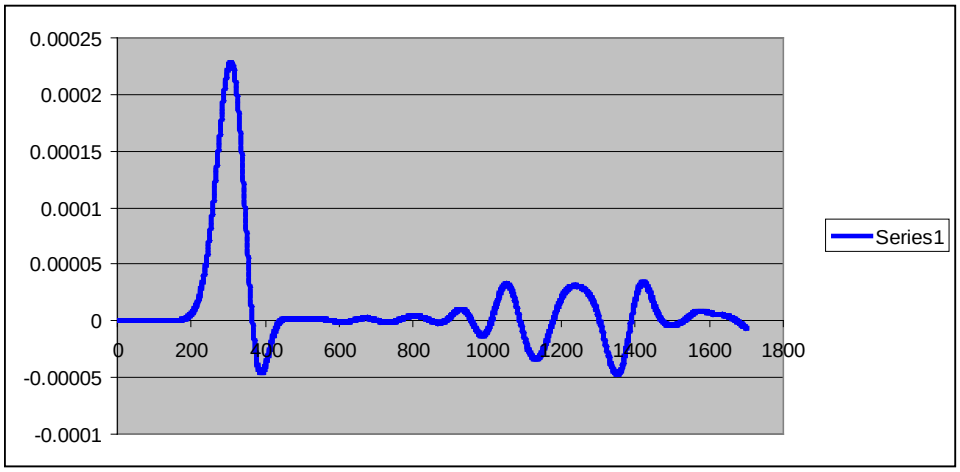
```
	//Field matrices
	CreateMatrix(Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(Dz,XCELLS + 1,YCELLS + 1,ZCELLS);

	CreateMatrix(Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(Ez,XCELLS + 1,YCELLS + 1,ZCELLS);

	CreateMatrix(Bx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(By,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(Bz,XCELLS,YCELLS,ZCELLS);

	CreateMatrix(Hx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(Hy,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(Hz,XCELLS,YCELLS,ZCELLS);


	//Create the update eqn constant matrices
	CreateMatrix(C1Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(C2Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(C1Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(C2Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(C1Dz,XCELLS + 1,YCELLS + 1,ZCELLS);
	CreateMatrix(C2Dz,XCELLS + 1,YCELLS + 1,ZCELLS);

	CreateMatrix(C1Bx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2Bx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C1By,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2By,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C1Bz,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2Bz,XCELLS,YCELLS,ZCELLS);

	CreateMatrix(C1Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(C2Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(C3Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	CreateMatrix(C1Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(C2Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(C3Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	CreateMatrix(C1Ez,XCELLS + 1,YCELLS + 1,ZCELLS);
	CreateMatrix(C2Ez,XCELLS + 1,YCELLS + 1,ZCELLS);
	CreateMatrix(C3Ez,XCELLS + 1,YCELLS + 1,ZCELLS);

	CreateMatrix(C1Hx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2Hx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C3Hx,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C1Hy,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2Hy,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C3Hy,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C1Hz,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C2Hz,XCELLS,YCELLS,ZCELLS);
	CreateMatrix(C3Hz,XCELLS,YCELLS,ZCELLS);
}

void DeleteAllMatrices()
{
	//Delete the constitute parameter matrices
	DeleteMatrix(epsr ,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(sigma,XCELLS,YCELLS,ZCELLS);

	//Delete the field component matrices
	DeleteMatrix(Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
```

```
	DeleteMatrix(Dz,XCELLS + 1,YCELLS + 1,ZCELLS);

	DeleteMatrix(Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(Ez,XCELLS + 1,YCELLS + 1,ZCELLS);

	DeleteMatrix(Bx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(By,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(Bz,XCELLS,YCELLS,ZCELLS);

	DeleteMatrix(Hx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(Hy,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(Hz,XCELLS,YCELLS,ZCELLS);

	//Delete the update eqn constant matrices
	DeleteMatrix(C1Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(C2Dx,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(C1Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(C2Dy,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(C1Dz,XCELLS + 1,YCELLS + 1,ZCELLS);
	DeleteMatrix(C2Dz,XCELLS + 1,YCELLS + 1,ZCELLS);

	DeleteMatrix(C1Bx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2Bx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C1By,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2By,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C1Bz,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2Bz,XCELLS,YCELLS,ZCELLS);

	DeleteMatrix(C1Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(C2Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(C3Ex,XCELLS,YCELLS + 1,ZCELLS + 1);
	DeleteMatrix(C1Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(C2Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(C3Ey,XCELLS + 1,YCELLS,ZCELLS + 1);
	DeleteMatrix(C1Ez,XCELLS + 1,YCELLS + 1,ZCELLS);
	DeleteMatrix(C2Ez,XCELLS + 1,YCELLS + 1,ZCELLS);
	DeleteMatrix(C3Ez,XCELLS + 1,YCELLS + 1,ZCELLS);


	DeleteMatrix(C1Hx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2Hx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C3Hx,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C1Hy,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2Hy,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C3Hy,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C1Hz,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C2Hz,XCELLS,YCELLS,ZCELLS);
	DeleteMatrix(C3Hz,XCELLS,YCELLS,ZCELLS);
}

//Create 3 dimensional matrix of dimx X dimy X dimz
//All elements are set to 0.0 as default
void CreateMatrix(MatPtr& mat,int dimx,int dimy,int dimz)
{
	mat = new double**[dimx];
	for(int i = 0; i < dimx; i++)
	{
		mat[i] = new double*[dimy];
		for(int j = 0;j < dimy; j++)
		{
			mat[i][j] = new double[dimz];
			for(int k = 0;k < dimz;  k++)
```

```
                        {
                                mat[i][j][k] = 0.0;
                        }
                }
        }
}

//Delete the  3 D matrix. Free the memory
void DeleteMatrix(MatPtr mat,int dimx,int dimy,int dimz)
{
        for(int i = 0;i < dimx;i++)
        {
                for(int j = 0;j<dimy;j++)
                {
                        free(mat[i][j]);
                }
                free(mat[i]);
        }
        free(mat);
}
```

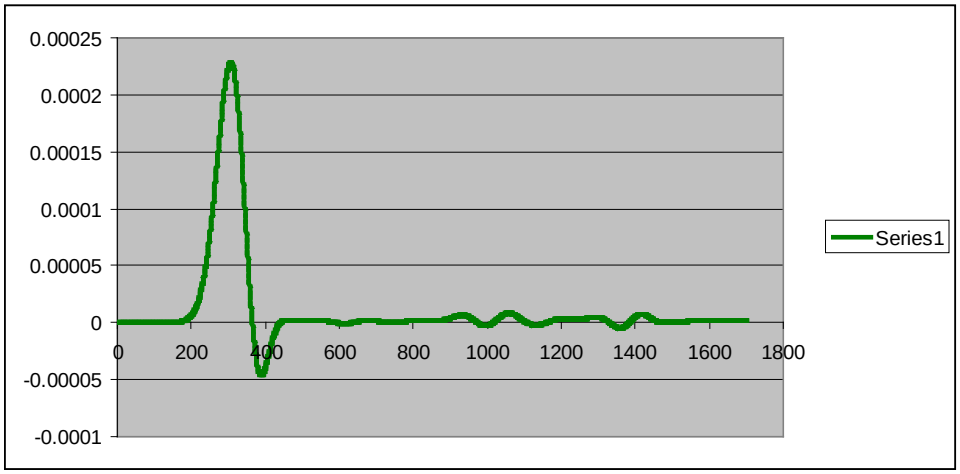
Appendix III : UPML experiments

Gaussian source at one end and Hx is sampled near the other end (near the UPML at the other end). The figures show the incident Gaussian and the reflection from the UPML – problem space boundary for varying values of UPML parameters.



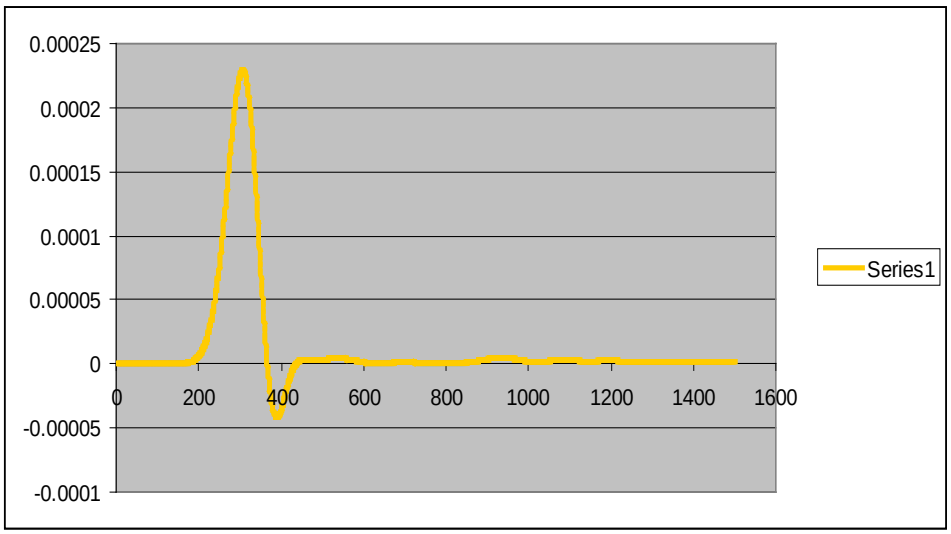
$PML\ layers = 19$

$g = 1.4, \sigma_o = 0.01$



$PML\ layers = 19$

$g = 1.4, \sigma_o = 0.1$



$PML\ layers = 19$

$g = 1.4, \sigma_o = 0.5$

Appendix IV : MATLAB code to evaluate S_{11} from simulation results

```
%Total  -> Simulation data from the microstrip patch simulation
%Incident -> Simulation data from the incident wave simulation
%Size of Total and Incident is 4000
dt = 0.265 / 6e8;
Ref = Total - Incident;

%Zero padding to increase frequency resolution
Incident = [Incident zeros(1,46000)];
Ref      = [Ref zeros(1,46000)];

reffft = fft(ref);
incfft = fft(Incident);

S11 = reffft ./ incfft;

k = 0 : 450;
plot(k/(50000*dt*1e9),10*log10(abs(S11(k + 1))) );
xlabel('Frequency (GHz)');
ylabel('S11 (dB)');
```