3

Two-Dimensional Simulation

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

3.1 FDTD IN TWO DIMENSIONS

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

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

$$\tilde{\boldsymbol{D}}(\omega) = \varepsilon_r^*(\omega) \cdot \tilde{\boldsymbol{E}}(\omega) \tag{3.1b}$$

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

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

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

$$D_z(\omega) = \varepsilon_r^*(\omega) \cdot E_z(\omega) \tag{3.2b}$$

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

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

As in one-dimensional simulation, it is important that there is a systematic interleaving of the fields to be calculated. This is illustrated in Fig. 3.1. Putting Eqs. (3.2a), (3.2c), and

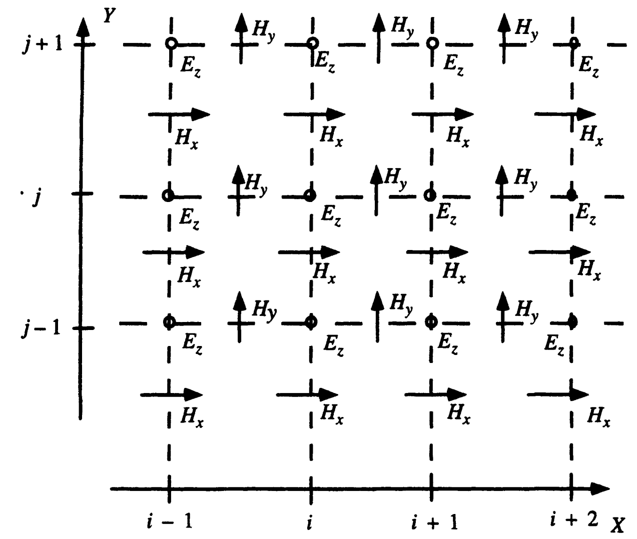

Figure 3.1 Interleaving of the E and H fields for the two-dimensional TM formulation.

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

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

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

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

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

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

we get the equations:

```
dz[i][j] = dz[i][j]
      +.5*(hy[i][j] - hy[i-1][j] - hx[i][j] + hx[i][j-1]);   (3.4a)
ez[i][j] = gaz[i][j]*(dz[i][j] - iz[i][j] );                  (3.4b)
iz[i][j] = iz[i][j] + gbz[i][j]*ez[i][j];                     (3.4c)
hx[i][j] = hx[i][j] + .5*( ez[i][j] - ez[i][j+1] );           (3.4d)
hy[i][j] = hy[i][j] + .5*( ez[i+1][j] - ez[i][j] );           (3.4e)
```

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

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

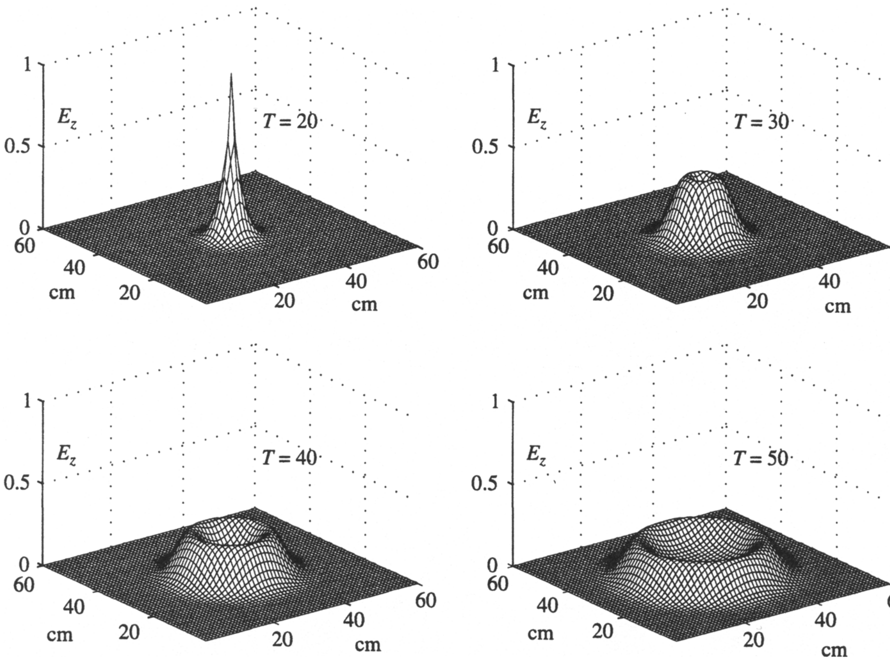

Figure 3.2 Results of a simulation using the program fd2d_3.1.c. A Gaussian pulse is initiated in the middle and travels outward.

PROBLEM SET 3.1

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

3.2 THE PERFECTLY MATCHED LAYER (PML)

Up to now, we have only briefly mentioned the issue of absorbing boundary conditions (ABCs). The size of the area that can be simulated using FDTD is limited by computer resources. For instance, in the two-dimensional simulation of the previous section, the program contains two-dimensional matrices for the values of all the fields, `dz`, `ez`, `hx`, and `hy`, as well as matrices to hold the parameters `gaz`, `gbz`, and `iz`. Suppose we are simulating a wave generated from a point source propagating in free space as in Fig. 3.2. As the wave propagates outward, it will eventually come to the edge of the allowable space, which is dictated by how the matrices have been dimensioned in the program. If nothing were done to address this, unpredictable reflections would be generated that would go back inward. There would be no way to determine which is the real wave and which is the reflected junk. This is the reason that ABCs have been an issue for as long as FDTD has been used. There have been numerous approaches to this problem [2, 3].

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

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

which are determined by the dielectric constants ε and permeabilities μ of the two media

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

Up to now, we have assumed that μ was a constant, so when a propagating pulse went from $\varepsilon = 1$ to $\varepsilon = 4$, as in Fig. 2.1, it saw a change in impedance and reflected a portion of the pulse given by Eq. (3.5). However, if μ changed with ε so η remained a constant, Γ would be zero and no reflection would occur. This still doesn't solve our problem, because the pulse will continue propagating in the new medium. What we really want is a medium that is also lossy so the pulse will die out before it hits the boundary. This is accomplished by making both ε and μ of Eq. (3.6) complex, because the imaginary part represents the part that causes decay. (See Appendix 1A at the end of Chapter 1.)

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

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

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

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

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

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

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

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

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

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

A few things are worth noting: first, the value ε_F is associated with the flux density D, not the electric field E; second, we have added two values each of ε_F in Eqs. (3.8a), and μ_F in Eq. (3.8c) and (3.8d), one for the x direction and one for the y direction; and finally, nothing was added to Eq. (3.8b). These fictitious values to implement the PML have nothing to do with the *real* values of $\varepsilon_r^*(\omega)$ which specify the medium.

Sacks, et al. [6] shows that there are two conditions to form a PML:

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

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

   The impedance is 1 because of our normalized units.

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

the other directions; i.e.,

$$\varepsilon^*_{Fx} = \frac{1}{\varepsilon^*_{Fy}} \tag{3.10a}$$

$$\mu^*_{Fx} = \frac{1}{\mu^*_{Fy}} \tag{3.10b}$$

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

$$\varepsilon^*_{Fm} = \varepsilon_{Fm} + \frac{\sigma_{Dm}}{j\omega\varepsilon_0} \qquad \text{for } m = x \text{ or } y \tag{3.11a}$$

$$\mu^*_{Fm} = \mu_{Fm} + \frac{\sigma_{Hm}}{j\omega\mu_0} \qquad \text{for } m = x \text{ or } y. \tag{3.11b}$$

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

$$\varepsilon_{Fm} = \mu_{Fm} = 1 \tag{3.12a}$$

$$\frac{\sigma_{Dm}}{\varepsilon_0} = \frac{\sigma_{Hm}}{\mu_0} = \frac{\sigma_D}{\varepsilon_0}. \tag{3.12b}$$

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

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

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

We will start by implementing a PML only in the X direction. Therefore, we will retain only the x dependent values of ε^*_F and μ^*_F in Eq. (3.8)

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

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

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

and use the values of Eq. (3.12):

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

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

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

Note that the permeability of H_x in Eq. (3.13b) is the inverse of that of H_y in Eq. (3.13c) in keeping with Eq. (3.10b). Therefore, we have fulfilled the second requirement for the PML. (Eq. (3.10a) is irrelevant for this 2D case, because we only have an E field in the z direction, which is perpendicular to both x and y, the directions of propagation.)

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

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

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

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

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

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

where once again we have used the fact that

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

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

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

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

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

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

where

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

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

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

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

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

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

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

Implementing this into an FDTD formulation gives

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

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

$$\begin{aligned}H_x^{n+1}(i,j+1/2) &= H_x^n(i,j+1/2) + \frac{c_0\cdot\Delta t}{\Delta x} curl\_e \\ &+ \frac{\Delta t\cdot c_0}{\Delta x}\frac{\sigma_D(x)\cdot\Delta t}{\varepsilon_0} I_{Hx}^{n+1/2}(i,j+1/2) \\ &= H_x^n(i,j+1/2) + \frac{c_0\cdot\Delta t}{\Delta x} curl\_e \\ &+ \frac{\sigma_D(x)\cdot\Delta t}{2\varepsilon_0} I_{Hx}^{n+1/2}(i,j+1/2).\end{aligned}$$

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

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

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

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

with

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

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

$$xn = \frac{\sigma\cdot\Delta t}{2\cdot\varepsilon_0}$$

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

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

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

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

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

Notice that the quantity in parentheses in Eq. (3.20) ranges between 0 and 1. The factor ".333" was found empirically to be the largest number that remained stable. Similarly, the cubic factor in Eq. (3.20) was found empirically to be the most effective variation—$fi2$ and $fi3$ are different only because they are computed at the half intervals, $i + 1/2$. The parameters vary in the following manner:

$fi1(i)$	from 0 to .333	(3.22a)
$gi2(i)$	from 1 to .75	(3.22b)
$gi3(i)$	from 1 to .5.	(3.22c)

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

So far, we have shown the implementation of the PML in the x direction. Obviously, it must also be done in the y direction. Therefore, we have to go back and add the y dependent

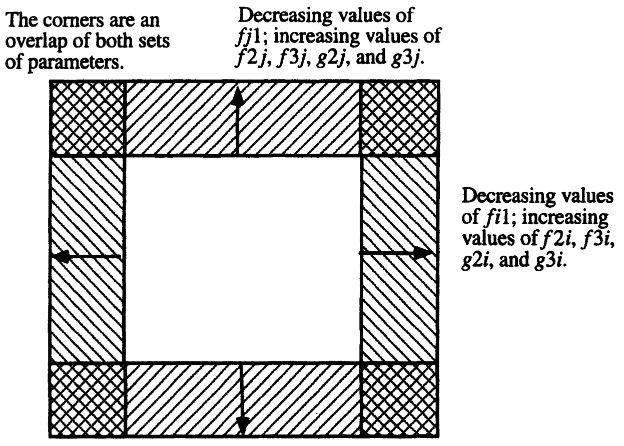
The corners are an overlap of both sets of parameters.
Decreasing values of *fj*1; increasing values of *f*2*j*, *f*3*j*, *g*2*j*, and *g*3*j*.
Decreasing values of *fi*1; increasing values of *f*2*i*, *f*3*i*, *g*2*i*, and *g*3*i*.

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

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

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

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

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

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

$$\begin{aligned}D_z^{n+1/2}(i,j)&=gi3(i)\cdot gj3(j)\cdot D_z^{n-1/2}(i)\\&+gi2(i)\cdot gj2(j)\cdot(0.5)\cdot\left[\begin{array}{c}H_y^n(i+1/2,j)-H_y^n(i-1/2,j)\\-H_x^n(i,j+1/2)+H_x^n(i,j-1/2)\end{array}\right].\end{aligned}$$

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

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

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

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

Finally, the H_x in the x direction becomes

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

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

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

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

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

Notice that we could simply "turn off" the PML in the main part of the problem space by setting $fi1$ and $fj1$ to 0, and the other parameters to 1. They are only one-dimensional parameters, so they add very little to the memory requirements. However, I_{Hx} and I_{Hy} are 2D parameters. Whereas memory requirements are not a main issue while we are in 2D, when we get to three dimensions, we will think twice before introducing two new parameters that are defined throughout the problem space, but are needed only in a small fraction of the space.

The PML is implemented in the program fd2d_3.2.c. Figure 3.4 illustrates the effectiveness of an 8-point PML with the source offset five cells from center in both the X and Y directions. Note that the outgoing contours remain concentric. Only when the wave gets within eight points of the edge, which is inside the PML, does distortion start to occur.

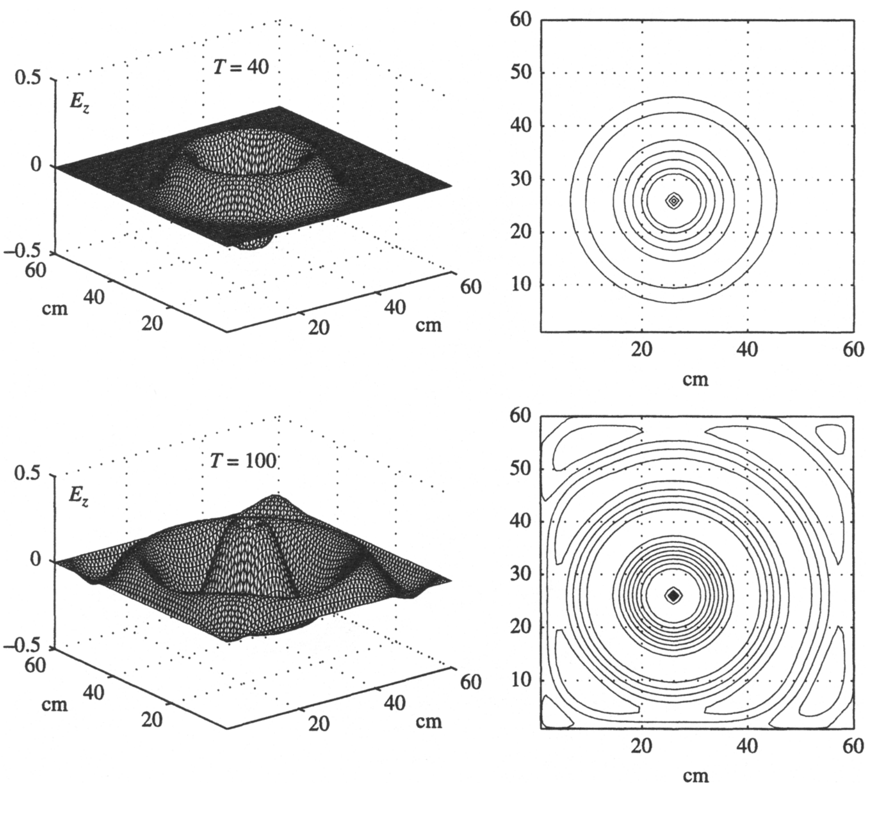

Figure 3.4 Results of a simulation using the program fd2d_3.2.c. A sinusoidal source is initiated at a point that is offset five cells from the center of the problems space in each direction. As the wave reaches the perfectly matched layer (PML), which is eight cells on every side, it is absorbed. The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

PROBLEM SET 3.2

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

3.3 TOTAL/SCATTERED FIELD FORMULATION

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

In order to simulate a plane wave in a 2D FDTD program, the problem space will be divided up into two regions, the *total field* and the *scattered field* (Fig. 3.5). There are two primary reasons for doing this: (1) The propagating plane wave should not interact with the absorbing boundary conditions; (2) the load on the absorbing boundary conditions should be minimized. These boundary conditions are not perfect, i.e., a certain portion of the impinging wave is reflected back into the problem space. By subtracting the incident field, the amount of the radiating field hitting the boundary is minimized, thereby reducing the amount of error.

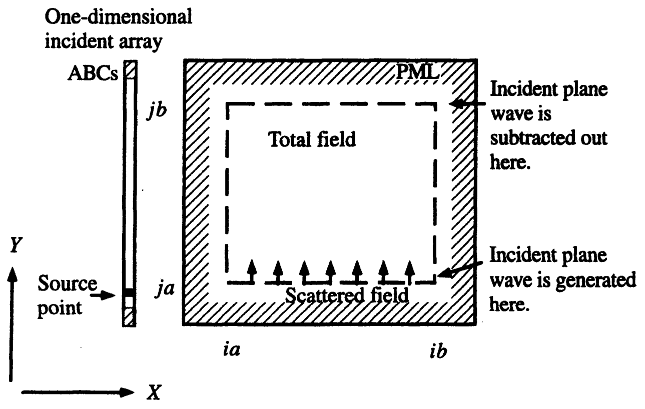
One-dimensional incident array
ABCs
jb
ja
Y
Source point
X
PML
Total field
Scattered field
ia
ib
Incident plane wave is subtracted out here.
Incident plane wave is generated here.

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

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

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

There are three places that must be modified:

1. The D_z value at $j = ja$ or $j = jb$:

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

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

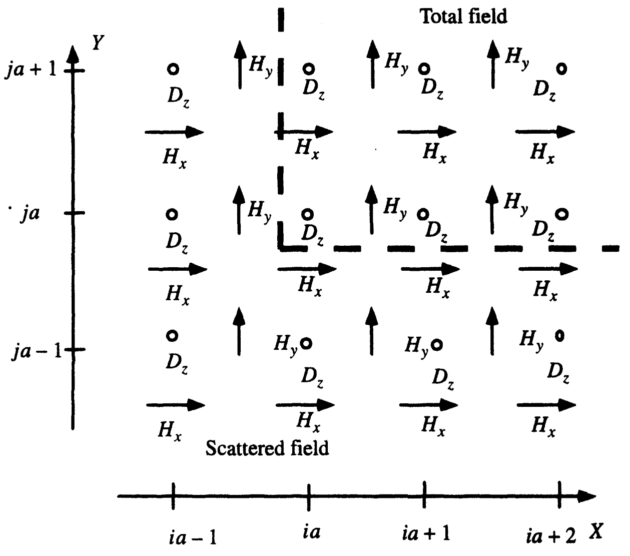

Figure 3.6 Every point is in either the total field or the scattered field.

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

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

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

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

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

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

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

3.3.1 A Plane Wave Impinging on a Dielectric Cylinder

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

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

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

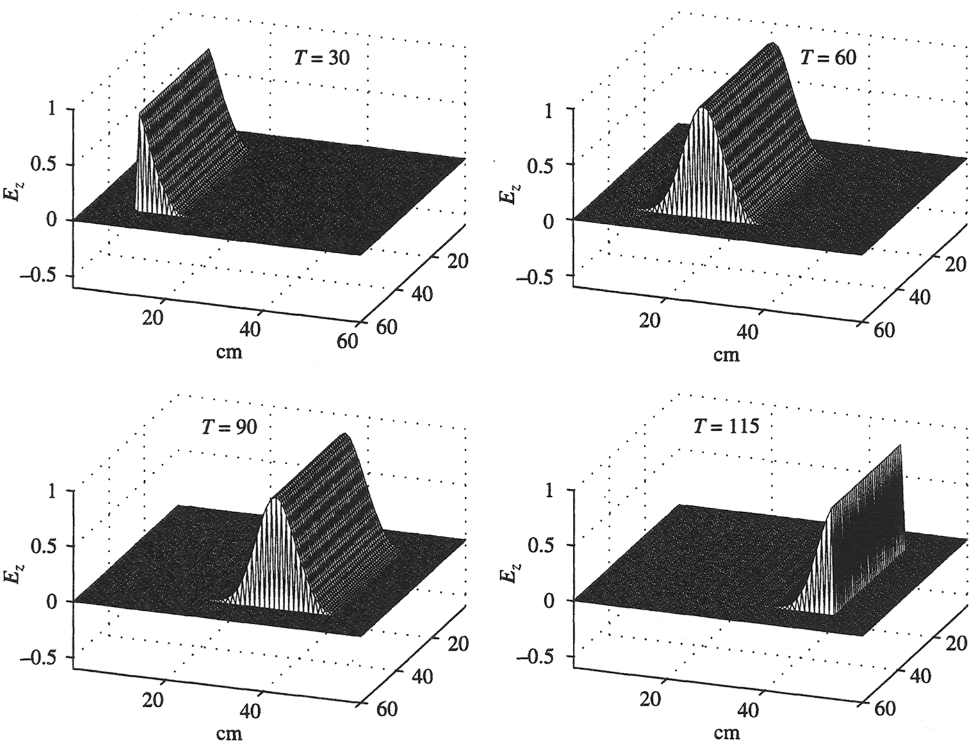

Figure 3.7 Simulation of a plane wave pulse propagating in free space. The incident pulse is generated at one end and subtracted out the other end.

the distance to the center of the problem space is calculated, and if it is less than the radius, the dielectric constant and conductivity are reset to `epsilon` and `sigma`, respectively. A diagram of the problem space to simulate a plane wave interacting with a dielectric cylinder is shown in Fig. 3.8.

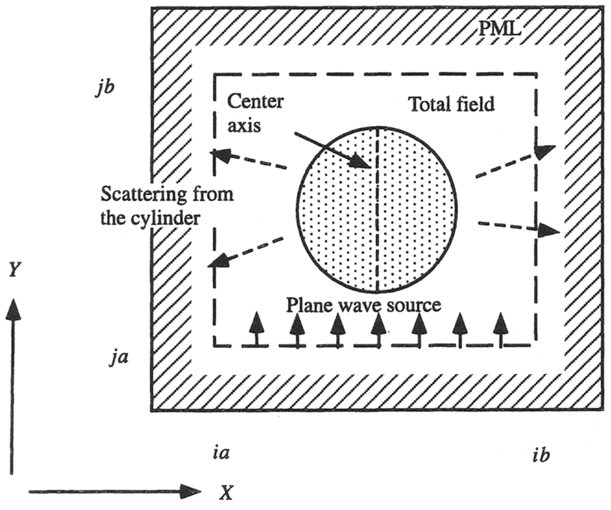

Figure 3.8 Diagram of the simulation of a plane wave striking a dielectric cylinder. The fields scattered from the cylinder are the only fields to leave the total field and strike the PML.

The weaknesses of this approach are obvious: since we determined the properties by a simple "in or out" approach, we are left with the "staircasing" at the edge of the cylinder.

Clearly it would be better if we had a way to make a smooth transition from one medium to another. One method is to divide every FDTD cell up into subcells; then determine the average dielectric properties according to how many subcells are in one medium and how many are in the other. The following code implements such a procedure. The ability to model multiple layers of different material has also been added.

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

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

The simulation of a plane wave pulse hitting a dielectric cylinder with $\varepsilon_r = 30$ and $\sigma = .3$ is shown in Fig. 3.9. After 25 time steps, the plane wave has started from the side; after 50 time steps, the pulse is interacting with the cylinder. Some of it passes through the cylinder, and some of it goes around it. After 100 steps, the main part of the propagating pulse is being subtracted out the end of the total field.

3.3.2 Fourier Analysis

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

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

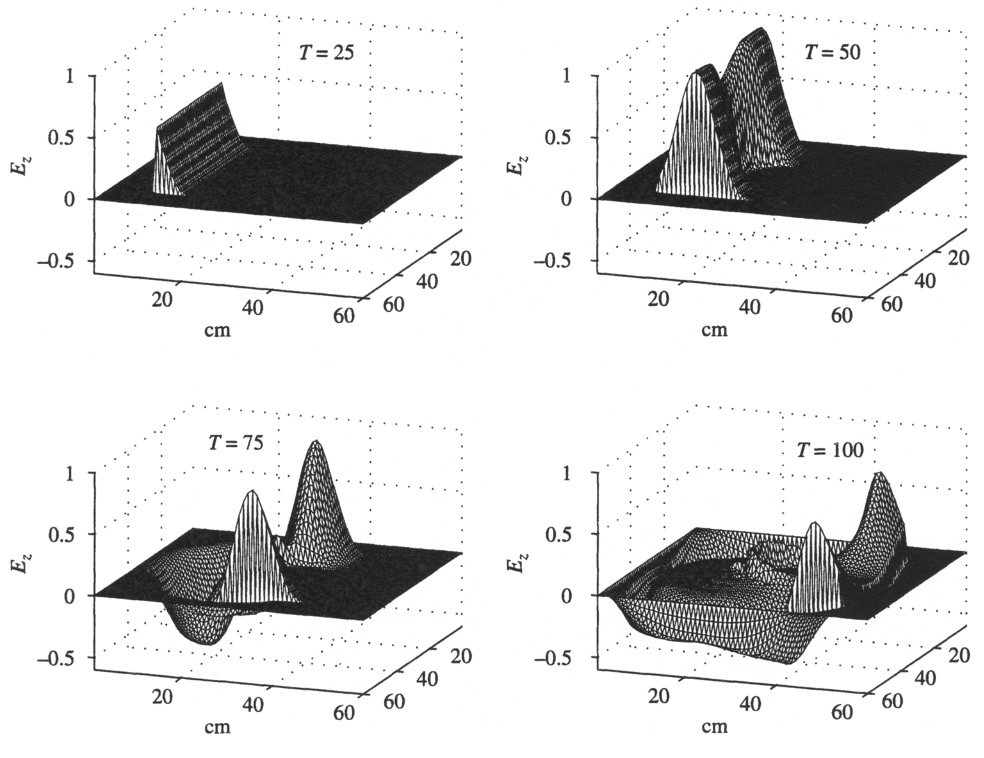

Figure 3.9 Simulation of a plane wave pulse impinging on a dielectric cylinder. The cylinder is 20 cm in diameter and has a dielectric constant of 30 and a conductivity of 0.3 S/m.

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

PROBLEM SET 3.3

1. The program fd2d_3.3.c implements the 2D TM FDTD algorithm with an incident plane wave. Note that Eq. (3.26) is implemented to give the correct E_z field at the total/scattered field boundary and Eq. (3.27) is implemented to give the correct H_x field. Why is there no modification to the H_y field?
2. Get the program fd2d_3.3.c running. You should be able to observe the pulse that is generated at `j=ja`, propagates through the problem space, and is subtracted out at `j=jb`.
3. The program fd2d_3.4.c differs from fd2d_3.3.c in that it simulates a plane wave hitting a dielectric cylinder. It also calculates the frequency response at three frequencies within the cylinder. Get this running and verify the results in Fig. 3.10.
4. The cylinder generated in fd2d_3.4.c generates the cylinder by the "in-or-out" method. Change this to the averaged values and add the ability to generate layered cylinders, as described in the last section of this chapter.

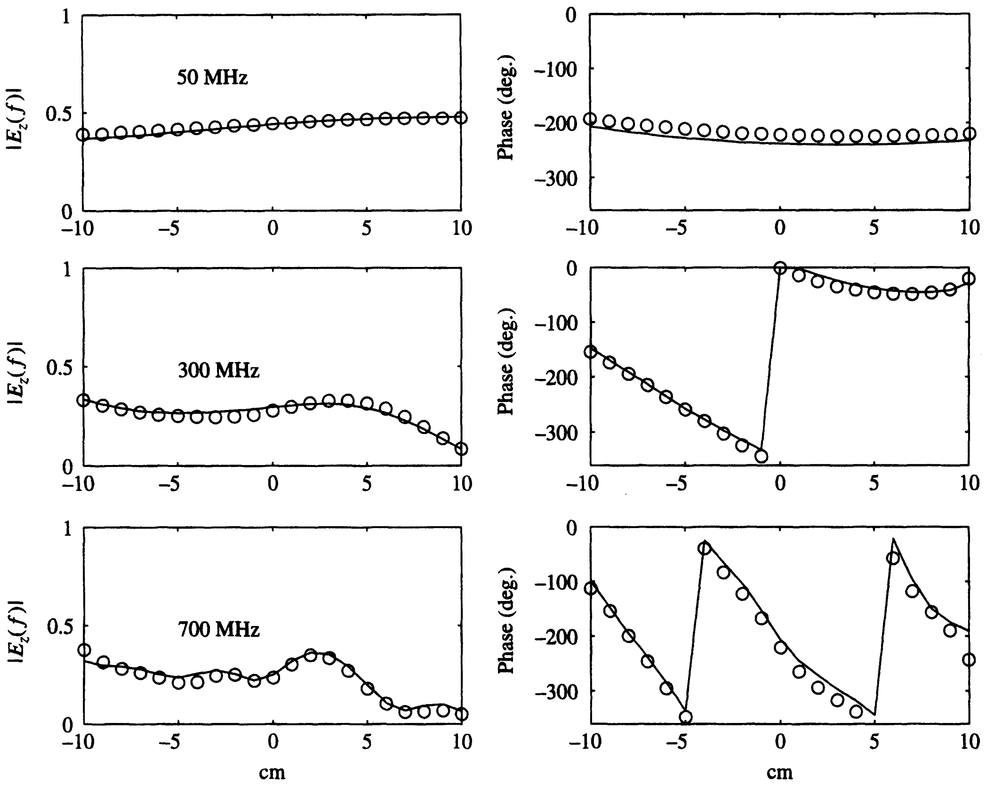

Figure 3.10 Comparison of the FDTD results vs. Bessel function expansion results along the propagation center axis of a cylinder at three frequencies. The cylinder is 20 cm in diameter and has a dielectric constant of 30 and a conductivity of 0.3 S/m.

REFERENCES

[1] K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. on Antennas and Propagation*, vol. AP-17, 1996, pp. 585–589.

[2] A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain*. Boston: Artech House, 1995.

[3] K. S. Kunz, and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*. Boca Raton, FL: CRC Press, 1993.

[4] J. P. Berenger, A perfectly matched layer for the absorption of electromagnetic waves, *J. Comput. Phys.*, vol. 114, 1994, pp. 185–200.

[5] D. M. Sullivan, A simplified PML for use with the FDTD method, *IEEE Microwave and Guided Wave Letters*, vol. 6, Feb. 1996, pp. 97–99.

[6] Z. S. Sacks, D. M. Kingsland, R. Lee, and J. F. Lee, A perfectly matched anisotropic absorber for use as an absorbing boundary condition, *IEEE Trans. Anten. and Prop.*, vol. 43, Dec. 1995, pp. 1460–1463.

[7] D. M. Sullivan, An unsplit step 3-D PML for use with the FDTD method, *IEEE Microwave and Guided Wave Letters*, vol. 7, July 1997, pp. 184–186.

[8] R. Harrington, *Time-Harmonic Electromagnetic Fields*, New York: McGraw-Hill, 1961.

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

#define IE  60
#define JE  60

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

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

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

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

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

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

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

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

        /* Put a Gaussian pulse in the middle */

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

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

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

        for ( j=1; j < jc; j++ ) {
           printf( "%2d  ",j);
           for ( i=1; i < ic; i++ ) {
              printf( "%5.2f ",ez[2*i][2*j]);
         }
              printf( " \n");
        }
              printf( "T = %5.0f \n",T);

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

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

  }
}
```

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

#define IE  60
#define JE  60

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

    ic = IE/2-5;
    jc = JE/2-5;
    ddx = .01;                    /* Cell size */
    dt =ddx/6e8;                  /* Time steps */
    epsz = 8.8e-12;
    pi=3.14159;

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

    /* Calculate the PML parameters */

       for (i=0;i< IE; i++) {
        gi2[i] = 1.0;
        gi3[i] = 1.0;
        fi1[i] = 0.0;
```

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

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

for (i=0;i<= npml; i++) {
xnum  = npml - i;
xd  = npml;
xxn = xnum/xd;
xn  = 0.33*pow(xxn,3.0);
printf(" %d %7.4f  %7.4f \n",i,xxn,xn);
   gi2[i] = 1.0/(1.0+xn);
   gi2[IE-1-i] = 1.0/(1.0+xn);
   gi3[i] = (1.0 - xn)/(1.0 + xn);
   gi3[IE-i-1] = (1.0 - xn)/(1.0 + xn);
xxn = (xnum-.5)/xd;
xn  = 0.25*pow(xxn,3.0);
   fi1[i] = xn;
   fi1[IE-2-i] = xn;
   fi2[i] = 1.0/(1.0+xn);
   fi2[IE-2-i] = 1.0/(1.0+xn);
   fi3[i] = (1.0 - xn)/(1.0 + xn);
   fi3[IE-2-i] = (1.0 - xn)/(1.0 + xn);
}

for (j=0;j<= npml; j++) {
xnum  = npml - j;
xd  = npml;
xxn = xnum/xd;
xn  = 0.33*pow(xxn,3.0);
printf(" %d %7.4f  %7.4f \n",i,xxn,xn);
   gj2[j] = 1.0/(1.0+xn);
   gj2[JE-1-j] = 1.0/(1.0+xn);
   gj3[j] = (1.0 - xn)/(1.0 + xn);
   gj3[JE-j-1] = (1.0 - xn)/(1.0 + xn);
xxn = (xnum-.5)/xd;
xn  = 0.25*pow(xxn,3.0);
   fj1[j] = xn;
   fj1[JE-2-j] = xn;
   fj2[j] = 1.0/(1.0+xn);
   fj2[JE-2-j] = 1.0/(1.0+xn);
```

```
            fj3[j] = (1.0 - xn)/(1.0 + xn);
            fj3[JE-2-j] = (1.0 - xn)/(1.0 + xn);
        }

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

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

    t0 = 40.0;
    spread = 15.0;
    T = 0;
    nsteps = 1;

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

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

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

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

        /* Sinusoidal Source */

        pulse =  sin(2*pi*1500*1e6*dt*T);;
        dz[ic][jc] = pulse;

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

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

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

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

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

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

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

        for ( j=1; j < JE; j++ ) {
           printf( "%2d  ",j);
           for ( i=1; i <= IE; i++ ) {
              printf( "%4.1f",ez[i][j]);
         }
              printf( " \n");
        }
        /* Write the E field out to a file "Ez" */
       fp = fopen( "Ez","w");
       for ( j=0; j < JE; j++ ) {
         for ( i=0; i < IE; i++ ) {
             fprintf( fp,"%6.3f ",ez[i][j]);
        }
             fprintf( fp," \n");
        }

        fclose(fp);
       printf("T =  %6.0f \n ",T);
   }
}
```

```
/* Fd2d_3.3.c.  2D TM program with plane wave source */

#define IE 60
#define JE 60

main ()
{

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

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

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

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

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

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

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

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

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

```
ez_inc_high_m1  = ez_inc[JE-2];

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

 /* Source */

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

    /* Incident Dz values */

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

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

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

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

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

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

 /* Calculate the Hx field */
 for ( j=0; j < JE-1; j++ ) {
```

```
          for ( i=0; i < IE; i++ ) {
           curl_e =   ez[i][j] - ez[i][j+1]   ;
           ihx[i][j] = ihx[i][j]   + fi1[i]*curl_e ;
           hx[i][j] = fj3[j]*hx[i][j]
                  + fj2[j]*.5*(curl_e + ihx[i][j]);
          }
       }

          /* Incident Hx values */

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

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

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

    }
/*  ----  End of the main FDTD loop ---- */
/*  Fd2d_3.4.c. 2D TM simulation of a plane
 wave source impinging on a dielectric cylinder.
 Analysis using Fourier Transforms */
# include <math.h>
# include <stdlib.h>
# include <stdio.h>

#define IE 60
#define JE 60
#define NFREQS 3

main ()
{
float    real_pt[NFREQS][IE][JE],imag_pt[NFREQS][IE][JE];
float    real  amp[IE][JE],phase[IE][JE];
float    real_in[5],imag_in[5],amp_in[5],phase_in[5];


     /* Parameters for the Fourier Transforms */

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

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

    /* Specify the dielectric cylinder */

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

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

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

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

         t0 = 25.0;
    spread = 8.0;
    T = 0;
    nsteps = 1;

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

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


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

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

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


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

       /*  Source */
```

```
        pulse =  exp(-.5*(pow((t0-T)/spread,2.0) ));
        ez_inc[3] = pulse;


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

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

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

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

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

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

        /* Incident Hx values */

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

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

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

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

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

  }
  }
```