1

ONE-DIMENSIONAL SIMULATION WITH THE FDTD METHOD

This chapter provides a step-by-step introduction to the finite-difference time-domain (FDTD) method, beginning with the simplest possible problem, the simulation of a pulse propagating in free space in one dimension. This example is used to illustrate the FDTD formulation. Subsequent sections lead to formulations for more complicated media.

1.1 ONE-DIMENSIONAL FREE-SPACE SIMULATION

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

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_0} \nabla \times \boldsymbol{H}, \tag{1.1a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.1b}$$

Electromagnetic Simulation Using the FDTD Method with Python, Third Edition.
Jennifer E. Houle and Dennis M. Sullivan.

Published 2020 by John Wiley & Sons, Inc.

$\boldsymbol{E}$ and $\boldsymbol{H}$ are vectors in three dimensions, so, in general, Eq. (1.1a) and (1.1b) represent three equations each. We will start with a simple one-dimensional case using only E_x and H_y, so Eq. (1.1a) and (1.1b) become

$$\frac{\partial E_x}{\partial t} = -\frac{1}{\varepsilon_0}\frac{\partial H_y}{\partial z}, \tag{1.2a}$$

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

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

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

$$\frac{E_x^{n+1/2}(k) - E_x^{n-1/2}(k)}{\Delta t} = -\frac{1}{\varepsilon_0}\frac{H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)}{\Delta x}, \tag{1.3a}$$

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

In these two equations, time is specified by the superscripts, that is, n represents a time step, and the time t is $t = \Delta t \cdot n$. Remember, we have to discretize everything for formulation into the computer. The term $n + 1$ means one time step later. The terms in parentheses represent distance, that is, k is used to calculate the distance $z = \Delta x \cdot k$. (It might seem more sensible to use Δz as the incremental step because in this case we are going in the z direction. However, Δx is so commonly used for a spatial increment that we will use Δx.) The formulation of Eq. (1.3a) and (1.3b) assume that the E and H fields are interleaved in both space and time. H uses the arguments $k + 1/2$ and $k - 1/2$ to indicate that the H field values are assumed to be located between the E field values. This is illustrated in Fig. 1.1. Similarly, the $n + 1/2$ or $n - 1/2$ superscript indicates that it occurs slightly after or before n, respectively. Equations (1.3a) and (1.3b) can be rearranged in an iterative algorithm:

$$E_x^{n+1/2}(k) = E_x^{n-1/2}(k) - \frac{\Delta t}{\varepsilon_0 \cdot \Delta x}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right], \tag{1.4a}$$

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

Notice that the calculations are interleaved in space and time. In Eq. (1.4a), for example, the new value of E_x is calculated from the previous value of E_x and the most recent values of H_y.

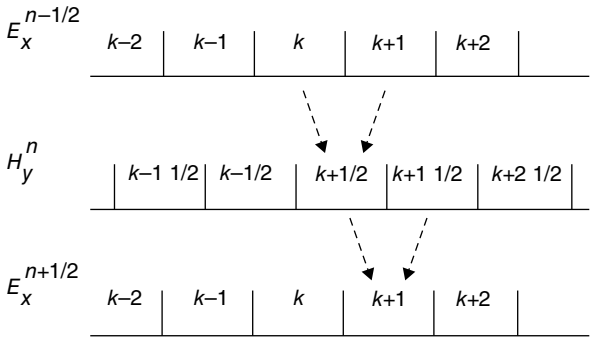

Figure 1.1 Interleaving of the E and H fields in space and time in the FDTD formulation. To calculate H_y, for instance, the neighboring values of E_x at k and $k+1$ are needed. Similarly, to calculate E_x, the values of H_y at $k+1/2$ and $k+1\frac{1}{2}$ are needed.

This is the fundamental paradigm of the FDTD method (1).

Equations (1.4a) and (1.4b) are very similar, but because ε_0 and μ_0 differ by several orders of magnitude, E_x and H_y will differ by several orders of magnitude. This is circumvented by making the following change of variables (2):

$$\widetilde{E} = \sqrt{\frac{\varepsilon_0}{\mu_0}} E. \tag{1.5}$$

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

$$\widetilde{E}_x^{n+1/2}(k) = \widetilde{E}_x^{n-1/2}(k) - \frac{\Delta t}{\sqrt{\varepsilon_0 \mu_0} \cdot \Delta x}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right], \tag{1.6a}$$

$$H_y^{n+1}\left(k+\frac{1}{2}\right) = H_y^n\left(k+\frac{1}{2}\right) - \frac{\Delta t}{\sqrt{\varepsilon_0 \mu_0} \cdot \Delta x}\left[\widetilde{E}_x^{n+1/2}(k+1) - \widetilde{E}_x^{n+1/2}(k)\right]. \tag{1.6b}$$

Once the cell size Δx is chosen, then the time step Δt is determined by

$$\Delta t = \frac{\Delta x}{2 \cdot c_0}, \tag{1.7}$$

where c_0 is the speed of light in free space. (The reason for this will be explained in Section 1.2.) Therefore, remembering that $\varepsilon_0 \mu_0 = 1/(c_0)^2$,

$$\frac{\Delta t}{\sqrt{\varepsilon_0 \mu_0} \cdot \Delta x} = \frac{\Delta x}{2 \cdot c_0} \cdot \frac{1}{\sqrt{\varepsilon_0 \mu_0} \cdot \Delta x} = \frac{1}{2}. \tag{1.8}$$

Rewriting Eq. (1.6a) and (1.6b) in Python gives the following:

$$\texttt{ex[k] = ex[k] + 0.5 * (hy[k - 1] - hy[k])}\,, \tag{1.9a}$$

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

Note that the n, $n + 1/2$, or $n - 1/2$ in the superscripts is gone. Time is implicit in the FDTD method. In Eq. (1.9a), the `ex` on the right side of the equal sign is the previous value at $n - 1/2$, and the `ex` on the left side is the new value $n + 1/2$, which is being calculated. Position, however, is explicit. The only difference is that $k + 1/2$ and $k - 1/2$ are rounded to k and $k - 1$ in order to specify a position in an array in the program.

The program fd1d_1_1.py at the end of this chapter is a simple one-dimensional FDTD program. It generates a Gaussian pulse in the center of the problem space, and the pulse propagates away in both directions as seen in Fig. 1.2. The E_x field is positive in both directions, but the H_y field is negative in the negative direction. The following points are worth noting about the program:

1. The E_x and H_y values are calculated by separate loops, and they employ the interleaving described above.
2. After the E_x values are calculated, the source is calculated. This is done by simply specifying a value of E_x at the point `k = kc` and overriding what was previously calculated. This is referred to as a *hard source* because a specific value is imposed on the FDTD grid.

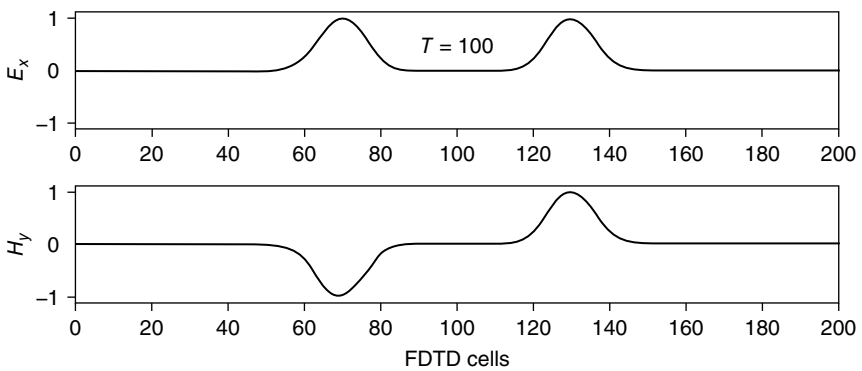

Figure 1.2 FDTD simulation of a pulse in free space after 100 time steps. The pulse originated in the center and travels outward.

PROBLEM SET 1.1

1. Get the program fd1d_1_1.py running. What happens when the pulse hits the end of the array? Why?
2. Modify the program so it has two sources, one at `kc - 20` and one at `kc + 20`. (Notice that `kc` is the center of the problem space.) What happens when the pulses meet? Explain this from basic electromagnetic (EM) theory.
3. Instead of E_x as the source, use H_y at `k = kc` as the source. What difference does it make? Try a two-point magnetic source at `kc - 1` and `kc` such that `hy[kc - 1] = - hy[kc]`. What does this look like? To what does it correspond physically?

1.2 STABILITY AND THE FDTD METHOD

Let us return to the discussion of how to determine the time step. An EM wave propagating in free space cannot go faster than the speed of light. To propagate a distance of one cell requires a minimum time of $\Delta t = \Delta x/c_0$. With a two-dimensional simulation, we must allow for the propagation in the diagonal direction, which brings the requirement to $\Delta t = \Delta x/\left(\sqrt{2}c_0\right)$. Obviously, a three-dimensional simulation requires $\Delta t = \Delta x/\left(\sqrt{3}c_0\right)$. This is summarized by the well-known *Courant Condition* (3, 4):

$$\Delta t = \frac{\Delta x}{\sqrt{n} \cdot c_0}, \tag{1.10}$$

where n is the dimension of the simulation. Unless otherwise specified, throughout this book we will determine Δt by

$$\Delta t = \frac{\Delta x}{2c_0}. \tag{1.11}$$

This is not necessarily the best formula; we will use it for simplicity to avoid using square roots.

PROBLEM SET 1.2

1. In fd1d_1_1.py, go to the governing equations, Eq. (1.9a) and (1.9b), and change the factor 0.5 to 1.0. What happens? Change it to 1.1. Now what happens? Change it to 0.25 and see what happens.

1.3 THE ABSORBING BOUNDARY CONDITION IN ONE DIMENSION

Absorbing boundary conditions are necessary to keep outgoing E and H fields from being reflected back into the problem space. Normally, in calculating the E field, we need to know the surrounding H values. This is a fundamental assumption of the FDTD method. At the edge of the problem space we will not have the value of one side. However, we have an advantage because we know that the fields at the edge must be propagating outward. We will use this fact to estimate the value at the end by using the value next to it (5).

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

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

This equation shows that it takes two time steps for the field to cross one cell. A commonsense approach tells us that an acceptable boundary condition might be

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

The implementation is relatively easy. Simply store a value of $E_x(1)$ for two time steps and then assign it to $E_x(0)$. Boundary conditions such as these have been implemented at both ends of the E_x array in the program fd1d_1_2.py. Figure 1.3 shows the results of a simulation using fd1d_1_2.py. A pulse that

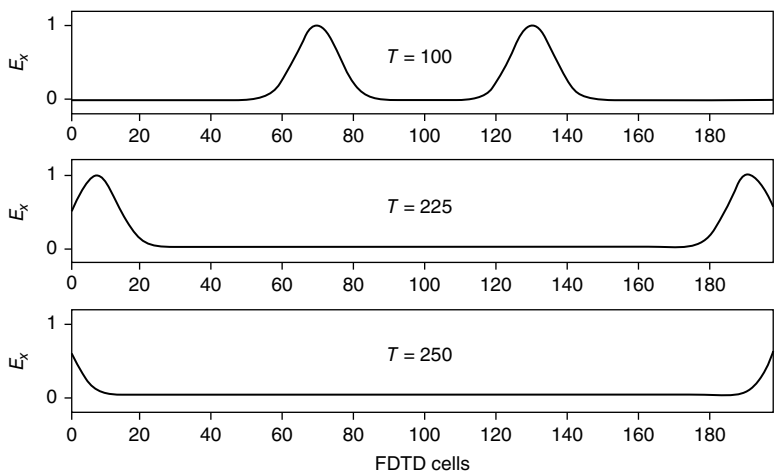

Figure 1.3 Simulation of an FDTD program with absorbing boundary conditions. Notice that the pulse is absorbed at the edges without reflecting anything back.

originates in the center propagates outward and is absorbed without reflecting anything back into the problem space.

PROBLEM SET 1.3

1. The program fd1d_1_2.py has absorbing boundary conditions at both ends. Get this program running and test it to ensure that the boundary conditions completely absorb the pulse.

1.4 PROPAGATION IN A DIELECTRIC MEDIUM

In order to simulate a medium with a dielectric constant other than 1, which corresponds to free space, we have to add the relative dielectric constant ε_r to Maxwell's equations:

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_r \varepsilon_0} \nabla \times \boldsymbol{H}, \tag{1.13a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.13b}$$

We will stay with our one-dimensional example,

$$\frac{\partial E_x}{\partial t} = -\frac{1}{\varepsilon_r \varepsilon_0} \frac{\partial H_y}{\partial z}, \tag{1.14a}$$

$$\frac{\partial H_y}{\partial t} = -\frac{1}{\mu_0} \frac{\partial E_x}{\partial z}, \tag{1.14b}$$

then go to the finite-difference approximations and make the change of variables in Eq. (1.5):

$$\widetilde{E}_x^{n+1/2}(k) = \widetilde{E}_x^{n-1/2}(k) - \frac{1}{2 \cdot \varepsilon_r} \left[H_y^n\left(k + \frac{1}{2}\right) - H_y^n\left(k - \frac{1}{2}\right) \right], \tag{1.15a}$$

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

From this we can get the computer equations

```
ex[k] = ex[k] + cb[k] * (hy[k - 1] - hy[k])        (1.16a)
hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1]),         (1.16b)
```

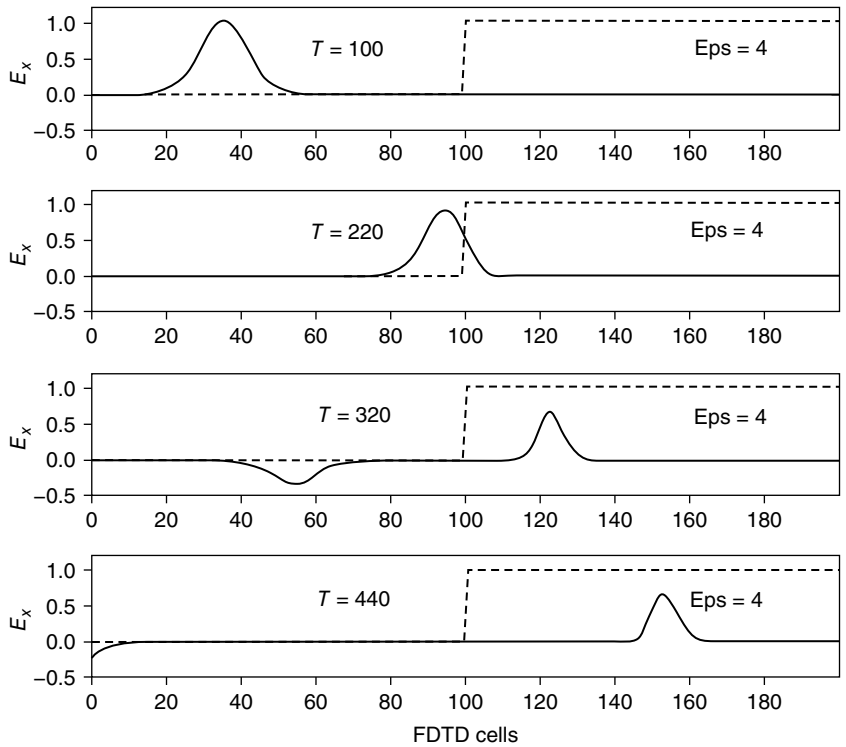

Figure 1.4 Simulation of a pulse striking dielectric material with a dielectric constant of 4. The source originates at cell number 5.

where

$$\texttt{cb[k] = 0.5/epsilon} \tag{1.17}$$

over those values of k that specify the dielectric material.

The program fd1d_1_3.py simulates the interaction of a pulse traveling in free space until it strikes a dielectric medium. The medium is specified by the parameter `cb` in Eq. (1.17). Figure 1.4 shows the result of a simulation with a dielectric medium having a relative dielectric constant of 4. Note that one portion of the pulse propagates into the medium and the other is reflected, in keeping with basic EM theory (6).

PROBLEM SET 1.4

1. The program fd1d_1_3.py simulates a problem containing partly free space and partly dielectric material. Run this program and duplicate the results of Fig. 1.4.

2. Look at the relative amplitudes of the reflected and transmitted pulses. Are they correct? Check them by calculating the reflection and transmission coefficients (Appendix 1.A).
3. Still using a dielectric constant of 4, let the transmitted pulse propagate until it hits the far right wall. What happens? What could you do to correct this?

1.5 SIMULATING DIFFERENT SOURCES

In the fd1d_1_1.py and fd1d_1_2.py, a source is assigned as values to E_x; this is referred to as a *hard source*. In fd1d_1_3.py, however, a value is added to E_x at a certain point; this is called a *soft source*. The reason is that with a hard source, a propagating pulse will see that added value and be reflected because a hard value of E_x looks like a metal wall to FDTD. With the soft source, a propagating pulse will just pass through.

Until now, we have been using a Gaussian pulse as the source. It is very easy to switch to a sinusoidal source. Just replace the parameter `pulse` with the following:

```
pulse = sin(2 * pi * freq_in * dt * time_step)
ex[5] = pulse + ex[5].
```

The parameter `freq_in` determines the frequency of the wave. This source is used in the program fd1d_1_4.py. Figure 1.5 shows the same dielectric medium problem with a sinusoidal source. A frequency of 700 MHz is used. Notice that the simulation was stopped before the wave reached the far right side. Remember that we have an absorbing boundary condition, but only for free space.

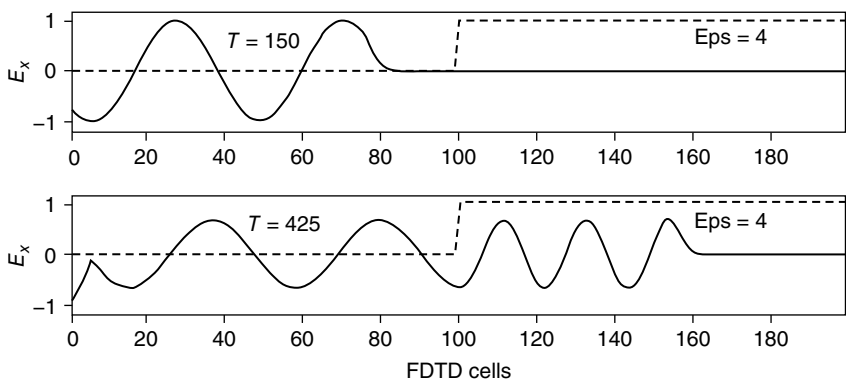

Figure 1.5 Simulation of a propagating sinusoidal wave of 700 MHz striking a medium with a relative dielectric constant of 4.

In fd1d_1_4.py, the cell size `ddx` and the time step `dt` are specified explicitly. We do this because we need `dt` in the calculation of `pulse`. The cell size `ddx` is only specified because it is needed to calculate `dt` from Eq. (1.7).

PROBLEM SET 1.5

1. Modify your program fd1d_1_3.py to simulate the sinusoidal source (see fd1d_1_4.py).
2. Keep increasing your incident frequency from 700 MHz upward at intervals of 300 MHz. What happens?
3. A *wave packet*, a sinusoidal function in a Gaussian envelope, is a type of propagating wave function that is of great interest in areas such as optics. Modify your program to simulate a wave packet.

1.6 DETERMINING CELL SIZE

Choosing the cell size to be used in an FDTD formulation is similar to any approximation procedure: Enough sampling points must be taken to ensure that an adequate representation is made. The number of points per wavelength is dependent on many factors (3, 4). However, a good rule of thumb is 10 points per wavelength. Experience has shown this to be adequate, with inaccuracies appearing as soon as the sampling drops below this rate.

Naturally, we must use a worst-case scenario. In general, this will involve looking at the highest frequencies we are simulating and determining the corresponding wavelength. For instance, suppose we are running simulations with 400 MHz. In free space, EM energy will propagate at the wavelength

$$\lambda_0 = \frac{c_0}{400\ \text{MHz}} = \frac{3 \times 10^8 \text{m/s}}{4 \times 10^8 \text{s}^{-1}} = 0.75\ \text{m}. \tag{1.18}$$

If we were only simulating free space, we would choose

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

However, if we are simulating EM propagation in biological tissues, for instance, we must look at the wavelength in the tissue with the highest dielectric constant, because this will have the corresponding shortest wavelength. For instance, muscle has a relative dielectric constant of about 50 at 400 MHz, so

$$\lambda_0 = \frac{\left(\frac{c_0}{\sqrt{50}}\right)}{400\,\text{MHz}} = \frac{0.424 \times 10^8 \text{m/s}}{4 \times 10^8 \text{s}^{-1}} = 10.6\,\text{cm}.$$

In this case, we would probably select a cell size of 1 cm.

PROBLEM SET 1.6

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

1.7 PROPAGATION IN A LOSSY DIELECTRIC MEDIUM

So far, we have simulated EM propagation in free space or in simple media that are specified by the relative dielectric constant ε_r. However, there are many media that also have a loss term specified by the conductivity. This loss term results in the attenuation of the propagating energy.

Once more we will start with the time-dependent Maxwell's curl equations, but we will write them in a more general form, which allows us to simulate propagation in media that have conductivity:

$$\varepsilon_r \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} = \nabla \times \boldsymbol{H} - \boldsymbol{J}, \tag{1.19a}$$

$$\frac{\partial \boldsymbol{H}}{\partial t} = -\frac{1}{\mu_0} \nabla \times \boldsymbol{E}. \tag{1.19b}$$

$\boldsymbol{J}$, the current density, can also be written as

$$\boldsymbol{J} = \sigma \boldsymbol{E},$$

where σ is the conductivity. Putting this into Eq. (1.19a) and dividing through by the dielectric constant we get

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{1}{\varepsilon_r \varepsilon_0} \nabla \times \boldsymbol{H} - \frac{\sigma}{\varepsilon_r \varepsilon_0} \boldsymbol{E}.$$

We now revert to our simple one-dimensional equation:

$$\frac{\partial E_x(t)}{\partial t} = -\frac{1}{\varepsilon_r \varepsilon_0} \frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\varepsilon_r \varepsilon_0} E_x(t),$$

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

$$\frac{\partial \widetilde{E}_x(t)}{\partial t} = -\frac{1}{\varepsilon_r\sqrt{\mu_0\varepsilon_0}}\frac{\partial H_y(t)}{\partial z} - \frac{\sigma}{\varepsilon_r\varepsilon_0}\widetilde{E}_x(t), \tag{1.20a}$$

$$\frac{\partial H_y(t)}{\partial t} = -\frac{1}{\sqrt{\mu_0\varepsilon_0}}\frac{\partial \widetilde{E}_x(t)}{\partial z}. \tag{1.20b}$$

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

$$\begin{aligned}\frac{E_x^{n+1/2}(k) - E_x^{n-1/2}(k)}{\Delta t} = &-\frac{1}{\varepsilon_r\sqrt{\varepsilon_0\mu_0}}\frac{H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)}{\Delta x} \\ &-\frac{\sigma}{\varepsilon_r\varepsilon_0}\frac{E_x^{n+1/2}(k) + E_x^{n-1/2}(k)}{2}.\end{aligned} \tag{1.21}$$

Notice that the last term in Eq. (1.20a) is approximated as the average across two time steps in Eq. (1.21). The tildes were dropped from Eq. (1.21) for simplicity. From Eq. (1.8),

$$\frac{1}{\sqrt{\varepsilon_0\mu_0}}\frac{\Delta t}{\Delta x} = \frac{1}{2},$$

so Eq. (1.21) becomes

$$\begin{aligned}E_x^{n+1/2}(k)\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right) = &\left(1-\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)E_x^{n-1/2}(k) \\ &-\frac{\left(\frac{1}{2}\right)}{\varepsilon_r}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right]\end{aligned}$$

or

$$\begin{aligned}E_x^{n+1/2}(k) = &\frac{\left(1-\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}{\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}E_x^{n-1/2}(k) \\ &-\frac{\left(\frac{1}{2}\right)}{\varepsilon_r\left(1+\frac{\Delta t\cdot\sigma}{2\varepsilon_r\varepsilon_0}\right)}\left[H_y^n\left(k+\frac{1}{2}\right) - H_y^n\left(k-\frac{1}{2}\right)\right].\end{aligned}$$

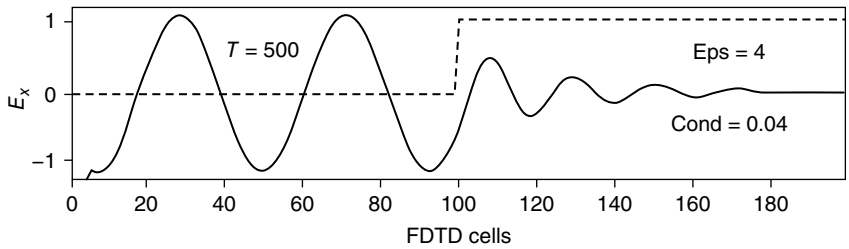

Figure 1.6 Simulation of a propagating sinusoidal wave striking a lossy dielectric material with a dielectric constant of 4 and a conductivity of 0.04 (S/m). The source is 700 MHz and originates at cell number 5.

From these we can get the computer equations:

$$\texttt{ex[k] = ca[k] * ex[k] + cb[k] * (hy[k - 1] - hy[k])} \tag{1.22a}$$

$$\texttt{hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])}\,, \tag{1.22b}$$

where

$$\texttt{eaf = dt * sigma/(2 * epsz * epsilon)}\,, \tag{1.23a}$$

$$\texttt{ca[k] = (1 - eaf)/(1 + eaf)}\,, \tag{1.23b}$$

$$\texttt{cb[k] = 0.5/(epsilon * (1 + eaf))}\,. \tag{1.23c}$$

The program fd1d_1_5.py simulates a sinusoidal wave hitting a lossy medium that has a dielectric constant of 4 and a conductivity of 0.04. The pulse is generated at the left side and propagates to the right (Fig. 1.6). Notice that the waveform in the medium is absorbed before it hits the boundary, so we do not have to worry about absorbing boundary conditions.

PROBLEM SET 1.7

1. Run program fd1d_1_5.py to simulate a complex dielectric material. Duplicate the results of Fig. 1.6.
2. Verify that your calculation of the sine wave in the lossy dielectric is correct: That is, it is the correct amplitude going into the slab, and then it attenuates at the proper rate (Appendix 1.A).
3. How would you write an absorbing boundary condition for a lossy material?
4. Simulate a pulse hitting a metal wall. This is very easy to do, if you remember that metal has a very high conductivity. For the complex dielectric, just

use σ = 1e6 or any large number. (It does not have to be the correct conductivity of the metal, just very large.) What does this do to the FDTD parameters `ca` and `cb`? What result does this have for the field parameters E_x and H_y? If you did not want to specify dielectric parameters, how else would you simulate metal in an FDTD program?

1.A APPENDIX

When a plane wave traveling in medium 1 strikes medium 2, the fraction that is reflected is given by the reflection coefficient Γ, and the fraction that is transmitted into medium 2 is given by the transmission coefficient τ. These are determined by the intrinsic impedances η_1 and η_2 of the respective media (6):

$$\Gamma = \frac{E_{\text{ref}}}{E_{\text{inc}}} = \frac{\eta_2 - \eta_1}{\eta_2 + \eta_1} \tag{1.A.1}$$

$$\tau = \frac{E_{\text{trans}}}{E_{\text{inc}}} = \frac{2\eta_2}{\eta_2 + \eta_1}. \tag{1.A.2}$$

The impedances are given by

$$\eta = \sqrt{\frac{\mu}{\varepsilon_0 \varepsilon_r}}. \tag{1.A.3}$$

The complex relative dielectric constant ε_r^* is given by

$$\varepsilon_r^* = \varepsilon_r + \frac{\sigma}{j\omega\varepsilon_0}.$$

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

$$\Gamma = \frac{\frac{1}{\sqrt{\varepsilon_2^*}} - \frac{1}{\sqrt{\varepsilon_1^*}}}{\frac{1}{\sqrt{\varepsilon_2^*}} + \frac{1}{\sqrt{\varepsilon_1^*}}} = \frac{\sqrt{\varepsilon_1^*} - \sqrt{\varepsilon_2^*}}{\sqrt{\varepsilon_1^*} + \sqrt{\varepsilon_2^*}} \tag{1.A.4}$$

$$\tau = \frac{\frac{2}{\sqrt{\varepsilon_1^*}}}{\frac{1}{\sqrt{\varepsilon_2^*}} + \frac{1}{\sqrt{\varepsilon_1^*}}} = \frac{2\sqrt{\varepsilon_1^*}}{\sqrt{\varepsilon_1^*} + \sqrt{\varepsilon_2^*}}. \tag{1.A.5}$$

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

$$E_x(z) = E_0 \cdot \mathrm{e}^{-k \cdot z} = E_0 \cdot \mathrm{e}^{-\mathrm{Re}\{k\} \cdot z} \mathrm{e}^{-j\mathrm{Im}\{k\} \cdot z},$$

where E_0 is the amplitude at $z = 0$. The wave number k is determined by

$$k = \frac{\omega}{c_0}\sqrt{\varepsilon_r}. \tag{1.A.6}$$

REFERENCES

1. K. S. Yee, Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media, *IEEE Trans. Antennas Propag.*, vol. 17, 1966, pp. 585–589.
2. A. Taflove and M. Brodwin, Numerical solution of steady state electromagnetic scattering problems using the time-dependent Maxwell's equations, *IEEE Trans. Microwave Theory Tech.*, vol. 23, 1975, pp. 623–730.
3. A. Taflove, *Computational Electrodynamics: The Finite-Difference Time-Domain Method*, 3rd Edition, Boston, MA: Artech House, 1995.
4. K. S. Kunz and R. J. Luebbers, *The Finite Difference Time Domain Method for Electromagnetics*, Boca Raton, FL: CRC Press, 1993.
5. G. Mur, Absorbing boundary conditions for the finite-difference approximation of the time domain electromagnetic field equations, *IEEE Trans. Electromagn. Compat.*, vol. 23, 1981, pp. 377–384.
6. D. K. Cheng, *Field and Wave Electromagnetics, Menlo Park, CA: Addison-Wesley, 1992.*

PYTHON PROGRAMS USED TO GENERATE FIGURES IN THIS CHAPTER

```
""" fd3d_1_1.py: 1D FDTD

Simulation in free space
"""

import numpy as np
from math import exp
from matplotlib import pyplot as plt

ke = 200
ex = np.zeros(ke)
hy = np.zeros(ke)

# Pulse parameters
kc = int(ke / 2)
t0 = 40
spread = 12
```

```
nsteps = 100

# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate the Ex field
    for k in range(1, ke):
        ex[k] = ex[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a Gaussian pulse in the middle
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    ex[kc] = pulse

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Plot the outputs as shown in Fig. 1.2
plt.rcParams['font.size'] = 12
plt.figure(figsize=(8, 3.5))

plt.subplot(211)
plt.plot(ex, color='k', linewidth=1)
plt.ylabel('E$_x$', fontsize='14')
plt.xticks(np.arange(0, 201, step=20))
plt.xlim(0, 200)
plt.yticks(np.arange(-1, 1.2, step=1))
plt.ylim(-1.2, 1.2)
plt.text(100, 0.5, 'T = {}'.format(time_step),
horizontalalignment='center')

plt.subplot(212)
plt.plot(hy, color='k', linewidth=1)
plt.ylabel('H$_y$', fontsize='14')
plt.xlabel('FDTD cells')
plt.xticks(np.arange(0, 201, step=20))
plt.xlim(0, 200)
plt.yticks(np.arange(-1, 1.2, step=1))
plt.ylim(-1.2, 1.2)

plt.subplots_adjust(bottom=0.2, hspace=0.45)
plt.show()


""" fd3d_1_2.py: 1D FDTD

Simulation in free space
Absorbing Boundary Condition added
"""
```

```
import numpy as np
from math import exp
from matplotlib import pyplot as plt

ke = 200
ex = np.zeros(ke)
hy = np.zeros(ke)

# Pulse parameters
kc = int(ke / 2)
t0 = 40
spread = 12

boundary_low = [0, 0]
boundary_high = [0, 0]

nsteps = 250

# Dictionary to keep track of desired points for plotting
plotting_points = [
    {'num_steps': 100, 'data_to_plot': None, 'label': ''},
    {'num_steps': 225, 'data_to_plot': None, 'label': ''},
    {'num_steps': 250, 'data_to_plot': None, 'label': 'FDTD cells'}
]

# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate the Ex field
    for k in range(1, ke):
        ex[k] = ex[k] + 0.5 * (hy[k - 1] - hy[k])

    # Put a Gaussian pulse in the middle
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    ex[kc] = pulse

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])

    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])
```

```
    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
           plotting_point['data_to_plot'] = np.copy(ex)

# Plot the outputs as shown in Fig. 1.3
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 5.25))


def plot_e_field(data, timestep, label):
    """Plot of E field at a single time step"""

    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('E$_x$', fontsize='14')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(np.arange(0, 1.2, step=1))
    plt.ylim(-0.2, 1.2)
    plt.text(100, 0.5, 'T = {}'.format(timestep),
             horizontalalignment='center')
    plt.xlabel('{}'.format(label))


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(3, 1, subplot_num + 1)
    plot_e_field(plotting_point['data_to_plot'],
                 plotting_point['num_steps'],
                 plotting_point['label'])

plt.tight_layout()
plt.show()

""" fd3d_1_3.py: 1D FDTD

Simulation of a pulse hitting a dielectric medium
"""

import numpy as np
from math import exp
from matplotlib import pyplot as plt

ke = 200
ex = np.zeros(ke)
hy = np.zeros(ke)

t0 = 40
spread = 12
```

```
boundary_low = [0, 0]
boundary_high = [0, 0]

# Create Dielectric Profile
cb = np.ones(ke)
cb = 0.5 * cb
cb_start = 100
epsilon = 4
cb[cb_start:] = 0.5 / epsilon

nsteps = 440

# Dictionary to keep track of desired points for plotting
plotting_points = [
    {'num_steps': 100, 'data_to_plot': None, 'label': ''},
    {'num_steps': 220, 'data_to_plot': None, 'label': ''},
    {'num_steps': 320, 'data_to_plot': None, 'label': ''},
    {'num_steps': 440, 'data_to_plot': None, 'label': 'FDTD cells'}
]

# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate the Ex field
    for k in range(1, ke):
        ex[k] = ex[k] + cb[k] * (hy[k - 1] - hy[k])

    # Put a Gaussian pulse at the low end
    pulse = exp(-0.5 * ((t0 - time_step) / spread) ** 2)
    ex[5] = pulse + ex[5]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])

    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
           plotting_point['data_to_plot'] = np.copy(ex)
```

```
# Plot the outputs as shown in Fig. 1.4
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 7))


def plot_e_field(data, timestep, epsilon, cb, label):
    """Plot of E field at a single time step"""

    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('E$_x$', fontsize='14')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(np.arange(-0.5, 1.2, step=0.5))
    plt.ylim(-0.5, 1.2)
    plt.text(70, 0.5, 'T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot((0.5 / cb - 1) / 3, 'k--', linewidth=0.75)
    # The math on cb above is just for scaling
    plt.text(170, 0.5, 'Eps = {}'.format(epsilon),
             horizontalalignment='center')
    plt.xlabel('{}'.format(label))


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(4, 1, subplot_num + 1)
    plot_e_field(plotting_point['data_to_plot'],
                 plotting_point['num_steps'], epsilon, cb,
                 plotting_point['label'])

plt.subplots_adjust(bottom=0.1, hspace=0.45)
plt.show()


""" fd3d_1_4.py: 1D FDTD

Simulation of a sinusoidal wave hitting a dielectric medium
"""

import numpy as np
from math import pi, sin
from matplotlib import pyplot as plt

ke = 200
ex = np.zeros(ke)
hy = np.zeros(ke)
```

```
ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
freq_in = 700e6

boundary_low = [0, 0]
boundary_high = [0, 0]

# Create Dielectric Profile
cb = np.ones(ke)
cb = 0.5 * cb
cb_start = 100
epsilon = 4
cb[cb_start:] = 0.5 / epsilon

nsteps = 425

# Dictionary to keep track of desired points for plotting
plotting_points = [
    {'num_steps': 150, 'data_to_plot': None, 'label': ''},
    {'num_steps': 425, 'data_to_plot': None, 'label': 'FDTD cells'}
]

# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate the Ex field
    for k in range(1, ke):
        ex[k] = ex[k] + cb[k] * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = sin(2 * pi * freq_in * dt * time_step)
    ex[5] = pulse + ex[5]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])

    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

    # Save data at certain points for later plotting
    for plotting_point in plotting_points:
        if time_step == plotting_point['num_steps']:
           plotting_point['data_to_plot'] = np.copy(ex)
```

```
# Plot the outputs in Fig. 1.5
plt.rcParams['font.size'] = 12
fig = plt.figure(figsize=(8, 3.5))


def plot_e_field(data, timestep, epsilon, cb, label):
    """Plot of E field at a single time step"""

    plt.plot(data, color='k', linewidth=1)
    plt.ylabel('E$_x$', fontsize='14')
    plt.xticks(np.arange(0, 199, step=20))
    plt.xlim(0, 199)
    plt.yticks(np.arange(-1, 1.2, step=1))
    plt.ylim(-1.2, 1.2)
    plt.text(50, 0.5, 'T = {}'.format(timestep),
             horizontalalignment='center')
    plt.plot((0.5 / cb - 1) / 3, 'k--', linewidth=0.75)
    # The math on cb above is just for scaling
    plt.text(170, 0.5, 'Eps = {}'.format(epsilon),
             horizontalalignment='center')
    plt.xlabel('{}'.format(label))


# Plot the E field at each of the time steps saved earlier
for subplot_num, plotting_point in enumerate(plotting_points):
    ax = fig.add_subplot(2, 1, subplot_num + 1)
    plot_e_field(plotting_point['data_to_plot'],
                 plotting_point['num_steps'], epsilon, cb,
                 plotting_point['label'])

plt.subplots_adjust(bottom=0.2, hspace=0.45)
plt.show()


""" fd3d_1_5.py: 1D FDTD

Simulation of a sinusoid wave hitting a lossy dielectric
"""

import numpy as np
from math import pi, sin
from matplotlib import pyplot as plt

ke = 200
ex = np.zeros(ke)
hy = np.zeros(ke)
```

```
ddx = 0.01  # Cell size
dt = ddx / 6e8  # Time step size
freq_in = 700e6

boundary_low = [0, 0]
boundary_high = [0, 0]

# Create Dielectric Profile
epsz = 8.854e-12
epsilon = 4
sigma = 0.04

ca = np.ones(ke)
cb = np.ones(ke) * 0.5
cb_start = 100

eaf = dt * sigma / (2 * epsz * epsilon)
ca[cb_start:] = (1 - eaf) / (1 + eaf)
cb[cb_start:] = 0.5 / (epsilon * (1 + eaf))

nsteps = 500

# Main FDTD Loop
for time_step in range(1, nsteps + 1):

    # Calculate the Ex field
    for k in range(1, ke):
        ex[k] = ca[k] * ex[k] + cb[k] * (hy[k - 1] - hy[k])

    # Put a sinusoidal at the low end
    pulse = sin(2 * pi * freq_in * dt * time_step)
    ex[5] = pulse + ex[5]

    # Absorbing Boundary Conditions
    ex[0] = boundary_low.pop(0)
    boundary_low.append(ex[1])

    ex[ke - 1] = boundary_high.pop(0)
    boundary_high.append(ex[ke - 2])

    # Calculate the Hy field
    for k in range(ke - 1):
        hy[k] = hy[k] + 0.5 * (ex[k] - ex[k + 1])

# Plot the outputs in Fig. 1.6
plt.rcParams['font.size'] = 12
plt.figure(figsize=(8, 2.25))
```

```
plt.plot(ex, color='k', linewidth=1)
plt.ylabel('E$_x$', fontsize='14')
plt.xticks(np.arange(0, 199, step=20))
plt.xlim(0, 199)
plt.yticks(np.arange(-1, 1.2, step=1))
plt.ylim(-1.2, 1.2)
plt.text(50, 0.5, 'T = {}'.format(time_step),
         horizontalalignment='center')
plt.plot((0.5 / cb - 1) / 3, 'k--',
         linewidth=0.75)  # The math on cb is just for scaling
plt.text(170, 0.5, 'Eps = {}'.format(epsilon),
         horizontalalignment='center')
plt.text(170, -0.5, 'Cond = {}'.format(sigma),
         horizontalalignment='center')
plt.xlabel('FDTD cells')

plt.subplots_adjust(bottom=0.25, hspace=0.45)
plt.show()
```