CHAPTER 4

Active and passive lumped elements

4.1 FDTD updating equations for lumped elements

Many practical electromagnetics applications require inclusion of lumped circuit elements. The lumped elements may be active sources in the form of voltage and current sources or passive in the form of resistors, inductors, and capacitors. Nonlinear circuit elements such as diodes and transistors are also required to be integrated in the numerical simulation of antennas and microwave devices. The electric current flowing through these circuit elements can be represented by the impressed current density term, $\vec{J}_i$, in Maxwell's curl equation

$$\nabla \times \vec{H} = \varepsilon \frac{\partial \vec{E}}{\partial t} + \sigma^e \vec{E} + \vec{J}_i. \tag{4.1}$$

Impressed currents are used to represent sources or known quantities. In this sense, they are the sources that cause the electric and magnetic fields in the computational domain [10]. A lumped element component placed between two nodes is characterized by the relationship between the voltage V across and the current I flowing between these two nodes. This relationship can be incorporated into Maxwell's curl equation (4.1) by expressing $\vec{E}$ in terms of V using

$$\vec{E} = -\nabla V \tag{4.2}$$

and by expressing $\vec{J}$ in terms of I using the relation

$$I = \int_S \vec{J} \cdot d\vec{s}, \tag{4.3}$$

where S is the cross-sectional area of a unit cell normal to the flow of the current I. These equations can be implemented in discrete time and space and can be incorporated into (4.1), which are expressed in terms of finite differences. Then, the finite-difference time-domain (FDTD) updating equations can be obtained that simulate the respective lumped element characteristics.

In this chapter we discuss the construction of the FDTD updating equations for lumped element components, and we demonstrate MATLAB® implementation of *definition*, *initialization*, and *simulation* of these elements.

4.1.1 Voltage source

In any electromagnetics simulation, one of the necessary components is the inclusion of sources. Types of sources vary depending on the problem type; scattering problems require incident fields from far zone sources, such as plane waves, to excite objects in a problem space, whereas many other problems require near zone sources, which are usually in the forms of voltage or current sources. In this section we derive updating equations that simulate the effects of a voltage source present in the problem space.

Consider the scalar curl equation (1.4c) repeated here for convenience:

$$\frac{\partial E_z}{\partial t} = \frac{1}{\varepsilon_z}\left(\frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} - \sigma_z^e E_z - J_{iz}\right). \tag{4.4}$$

This equation constitutes the relation between the current density J_{iz} flowing in the z direction and the electric and magnetic field vector components. Application of the central difference formula to the time and space derivatives based on the field positioning scheme illustrated in Figure 4.1 yields

$$\begin{aligned}\frac{E_z^{n+1}(i,\,j,\,k) - E_z^n(i,\,j,\,k)}{\Delta t} = {} & \frac{1}{\varepsilon_z(i,\,j,\,k)}\frac{H_y^{n+\frac{1}{2}}(i,\,j,\,k) - H_y^{n+\frac{1}{2}}(i-1,\,j,\,k)}{\Delta x} \\ & - \frac{1}{\varepsilon_z(i,\,j,\,k)}\frac{H_x^{n+\frac{1}{2}}(i,\,j,\,k) - H_x^{n+\frac{1}{2}}(i,\,j-1,\,k)}{\Delta y} \\ & - \frac{\sigma_z^e(i,\,j,\,k)}{2\varepsilon_z(i,\,j,\,k)}\left(E_z^{n+1}(i,\,j,\,k) + E_z^n(i,\,j,\,k)\right) \\ & - \frac{1}{\varepsilon_z(i,\,j,\,k)}J_{iz}^{n+\frac{1}{2}}(i,\,j,\,k)\end{aligned} \tag{4.5}$$

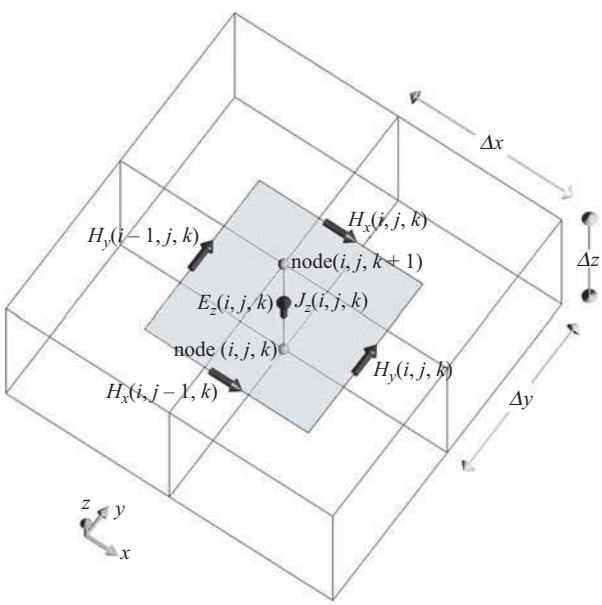

Figure 4.1 Field components around $E_z(i, j, k)$.

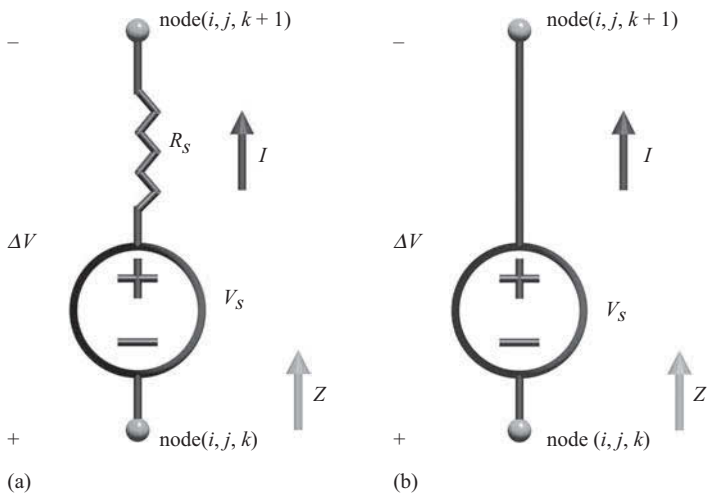

Figure 4.2 Voltage sources placed between nodes (i, j, k) and $(i, j, k + 1)$: (a) voltage source with internal resistance (soft source) and (b) voltage source without internal resistance (hard source).

We want to place a voltage source with V_s volts magnitude and R_s ohms internal resistance between nodes (i, j, k) and $(i, j, k+1)$ as illustrated in Figure 4.2(a), where V_s is a time-varying function with a predetermined waveform. The voltage–current relation for this circuit can be written as

$$I = \frac{\Delta V + V_s}{R_s}, \tag{4.6}$$

where ΔV is the potential difference between nodes (i, j, k) and $(i, j, k + 1)$. The term ΔV can be expressed in terms of E_z using (4.2), which translates to

$$\Delta V = \Delta z \times E_z^{n+\frac{1}{2}}(i, j, k) = \Delta z \times \frac{E_z^{n+1}(i, j, k) + E_z^n(i, j, k)}{2}, \tag{4.7}$$

in discrete form at time instant $(n + 0.5)\Delta t$. Current I is the current flowing through the surface enclosed by the magnetic field components in Figure 4.1, which can be expressed in terms of J_{iz} using (4.3) as

$$I^{n+\frac{1}{2}} = \Delta x \Delta y \, J_{iz}^{n+\frac{1}{2}}(i, j, k). \tag{4.8}$$

One should notice that ΔV in (4.7) is evaluated at time instant $(n + 0.5)\Delta t$, which corresponds to the required time instant of I and J dictated by (4.5). Inserting (4.7) and (4.8) in (4.6) one can obtain

$$J_{iz}^{n+\frac{1}{2}}(i, j, k) = \frac{\Delta z}{2\Delta x \Delta y R_s} \times \left(E_z^{n+1}(i, j, k) + E_z^n(i, j, k)\right) + \frac{1}{\Delta x \Delta y R_s} \times V_s^{n+\frac{1}{2}}. \tag{4.9}$$

Equation (4.9) includes the voltage–current relation for a voltage source tying V_s and R_s to electric field components in the discrete space and time. One can use (4.9) in (4.5) and

rearrange the terms such that the future value of the electric field component E_z^{n+1} can be calculated using other terms, which yields our standard form updating equations such that

$$\begin{aligned} E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k) \times E_z^n(i,j,k) \\ & + C_{ezhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k) \right) \\ & + C_{ezhx}(i,j,k) \times \left(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j-1,k) \right) \\ & + C_{ezs}(i,j,k) \times V_s^{n+\frac{1}{2}}(i,j,k), \end{aligned} \tag{4.10}$$

where

$$C_{eze}(i,j,k) = \frac{2\varepsilon_z(i,j,k) - \Delta t \sigma_z^e(i,j,k) - \dfrac{\Delta t \Delta z}{R_s \Delta x \Delta y}}{2\varepsilon_z(i,j,k) + \Delta t \sigma_z^e(i,j,k) + \dfrac{\Delta t \Delta z}{R_s \Delta x \Delta y}},$$

$$C_{ezhy}(i,j,k) = \frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t \sigma_z^e(i,j,k) + \dfrac{\Delta t \Delta z}{R_s \Delta x \Delta y}\right)\Delta x},$$

$$C_{ezhx}(i,j,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t \sigma_z^e(i,j,k) + \dfrac{\Delta t \Delta z}{R_s \Delta x \Delta y}\right)\Delta y},$$

$$C_{ezs}(i,j,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t \sigma_z^e(i,j,k) + \dfrac{\Delta t \Delta z}{R_s \Delta x \Delta y}\right)(R_s \Delta x \Delta y)},$$

Equation (4.10) is the FDTD updating equation modeling a voltage source placed between the nodes (i, j, k) and $(i, j, k + 1)$, which is oriented in the z direction. The FDTD updating equations for voltage sources oriented in other directions can easily be obtained following the same steps as just illustrated.

The FDTD updating equation (4.10) is given for a voltage source with a polarity in the positive z direction as indicated in Figure 4.2(a). To model a voltage source with the opposite polarity, one needs only to use the inverse of the voltage magnitude waveform by changing V_s to $-V_s$.

4.1.2 Hard voltage source

In some applications it may be desired that a voltage difference is enforced between two points in the problem space. This can be achieved by using a voltage source without any internal resistances, which is called a *hard voltage source*. For instance, Figure 4.2(b) illustrates such a scenario, where a voltage source with V_s volts magnitude is placed between the nodes (i, j, k) and $(i, j, k + 1)$. The FDTD updating equation for this voltage source can simply be obtained by letting $R_s \to 0$ in (4.10), which can be written as

$$E_z^{n+1}(i,j,k) = -E_z^n(i,j,k) - \frac{2}{\Delta z} V_s^{n+\frac{1}{2}}(i,j,k). \tag{4.11}$$

To conform with the general form of the FDTD updating equations, (4.11) can be expressed as

$$\begin{aligned} E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k) \times E_z^n(i,j,k) \\ & + C_{ezhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)\right) \\ & + C_{ezhx}(i,j,k) \times \left(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j-1,k)\right) \\ & + C_{ezs}(i,j,k) \times V_s^{n+\frac{1}{2}}(i,j,k), \end{aligned} \tag{4.12}$$

where

$$C_{eze}(i,j,k) = -1, \quad C_{ezhy}(i,j,k) = 0, \quad C_{ezhx}(i,j,k) = 0, \quad C_{ezs}(i,j,k) = -\frac{2}{\Delta z}.$$

4.1.3 Current source

Figure 4.3(a) illustrates a current source with I_s amperes magnitude and R_s internal resistance, where I_s is a function of time with a time-varying waveform. The voltage–current relation for the current source can be written as

$$I = I_s + \frac{\Delta V}{R_s}. \tag{4.13}$$

Expressions of ΔV in (4.7) and $I^{n+\frac{1}{2}}$ in (4.8) can be used in (4.13), which yields in discrete time and space

$$J_{iz}^{n+\frac{1}{2}}(i,j,k) = \frac{\Delta z}{2\Delta x \Delta y R_s} \times \left(E_z^{n+1}(i,j,k) + E_z^n(i,j,k)\right) + \frac{1}{\Delta x \Delta y} \times I_s^{n+\frac{1}{2}}. \tag{4.14}$$

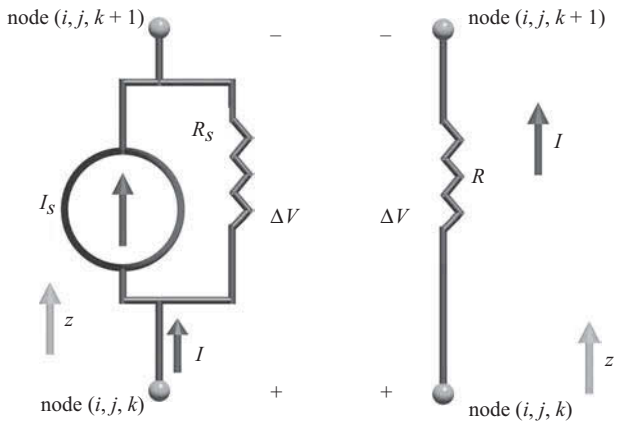

Figure 4.3 Lumped elements placed between nodes (i, j, k) and $(i, j, k + 1)$: (a) current source with internal resistance and (b) resistor.

One can use (4.14) in (4.5) and rearrange the terms such that E_z^{n+1} can be calculated using other terms, which yields

$$\begin{aligned} E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k) \times E_z^n(i,j,k) \\ & + C_{ezhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)\right) \\ & + C_{ezhx}(i,j,k) \times \left(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j-1,k)\right) \\ & + C_{ezs}(i,j,k) \times I_s^{n+\frac{1}{2}}(i,j,k), \end{aligned} \tag{4.15}$$

where

$$C_{eze}(i,j,k) = \frac{2\varepsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k) - \dfrac{\Delta t\Delta z}{R_s\Delta x\Delta y}}{2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{\Delta t\Delta z}{R_s\Delta x\Delta y}},$$

$$C_{ezhy}(i,j,k) = \frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{\Delta t\Delta z}{R_s\Delta x\Delta y}\right)\Delta x},$$

$$C_{ezhx}(i,j,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{\Delta t\Delta z}{R_s\Delta x\Delta y}\right)\Delta y},$$

$$C_{ezs}(i,j,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{\Delta t\Delta z}{R_s\Delta x\Delta y}\right)\Delta x\Delta y},$$

Equation (4.15) is the updating equation modeling a current source. One should notice that (4.15) is the same as (4.10) except for the source term; that is, replacing the $V_s^{n+\frac{1}{2}}(i,j,k)$ term in (4.10) by $R_s \times I_s^{n+\frac{1}{2}}(i,j,k)$ yields (4.15).

The FDTD updating equation (4.15) is given for a current source with a current flowing in the positive z direction as indicated in Figure 4.3(a). To model a current source with the opposite flow direction, one needs only to use the inverse of the current magnitude waveform by changing I_s to $-I_s$.

4.1.4 Resistor

Having derived the updating equations for a voltage source or a current source with internal resistance, it is straightforward to derive updating equations for a resistor. For instance, eliminating the current source in Figure 4.3(a) yields the resistor in Figure 4.3(b). Hence, setting the source term $I_s^{n+\frac{1}{2}}(i,j,k)$ in (4.15) to zero results the updating equation for a resistor as

$$\begin{aligned} E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k) \times E_z^n(i,j,k) \\ & + C_{ezhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)\right) \\ & + C_{ezhx}(i,j,k) \times \left(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j-1,k)\right), \end{aligned} \tag{4.16}$$

where

$$C_{eze}(i,\,j,\,k) = \frac{2\varepsilon_z(i,\,j,\,k) - \Delta t\sigma_z^e(i,\,j,\,k) - \dfrac{\Delta t\Delta z}{R\Delta x\Delta y}}{2\varepsilon_z(i,\,j,\,k) + \Delta t\sigma_z^e(i,\,j,\,k) + \dfrac{\Delta t\Delta z}{R\Delta x\Delta y}},$$

$$C_{ezhy}(i,\,j,\,k) = \frac{2\Delta t}{\left(2\varepsilon_z(i,\,j,\,k) + \Delta t\sigma_z^e(i,\,j,\,k) + \dfrac{\Delta t\Delta z}{R\Delta x\Delta y}\right)\Delta x},$$

$$C_{ezhx}(i,\,j,\,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,\,j,\,k) + \Delta t\sigma_z^e(i,\,j,\,k) + \dfrac{\Delta t\Delta z}{R\Delta x\Delta y}\right)\Delta y}.$$

4.1.5 Capacitor

Figure 4.4(a) illustrates a capacitor with C farads capacitance. The voltage–current relation for the capacitor can be written as

$$I = C\frac{d\Delta V}{dt}. \tag{4.17}$$

This relation can be expressed in discrete time and space as

$$I^{n+\frac{1}{2}} = C\frac{\Delta V^{n+1} - \Delta V^n}{\Delta t}. \tag{4.18}$$

Using ΔV in (4.7) and $I^{n+\frac{1}{2}}$ in (4.8) together with (4.18), one can obtain

$$J_{iz}^{n+\frac{1}{2}}(i,\,j,\,k) = \frac{C\Delta z}{\Delta t\Delta x\Delta y} \times \left(E_z^{n+1}(i,\,j,\,k) - E_z^n(i,\,j,\,k)\right). \tag{4.19}$$

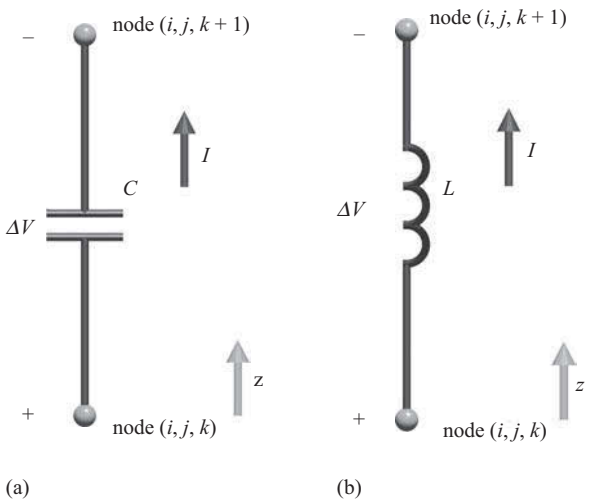

Figure 4.4 Lumped elements placed between nodes (i, j, k) and $(i, j, k + 1)$: (a) capacitor and (b) inductor.

One can use (4.19) in (4.5) and rearrange the terms such that E_z^{n+1} can be calculated using other terms, which yields the updating equation for a capacitor as

$$\begin{aligned} E_z^{n+1}(i,j,k) &= C_{eze}(i,j,k) \times E_z^n(i,j,k) \\ &+ C_{ezhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i-1,j,k)\right) \\ &+ C_{ezhx}(i,j,k) \times \left(H_x^{n+\frac{1}{2}}(i,j,k) - H_x^{n+\frac{1}{2}}(i,j-1,k)\right), \end{aligned} \tag{4.20}$$

where

$$C_{eze}(i,j,k) = \frac{2\varepsilon_z(i,j,k) - \Delta t\sigma_z^e(i,j,k) + \dfrac{2C\Delta z}{\Delta x\Delta y}}{2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{2C\Delta z}{\Delta x\Delta y}},$$

$$C_{ezhy}(i,j,k) = \frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{2C\Delta z}{\Delta x\Delta y}\right)\Delta x},$$

$$C_{ezhx}(i,j,k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k) + \Delta t\sigma_z^e(i,j,k) + \dfrac{2C\Delta z}{\Delta x\Delta y}\right)\Delta y}.$$

4.1.6 Inductor

Figure 4.4(b) illustrates an inductor with L henrys inductance. The inductor is characterized by the voltage–current relation

$$V = L\frac{dI}{dt}. \tag{4.21}$$

This relation can be expressed in discrete time and space using the central difference formula at time instant $n\Delta t$ as

$$\Delta V^n = \frac{L}{\Delta t}\left(I^{n+\frac{1}{2}} - I^{n-\frac{1}{2}}\right). \tag{4.22}$$

Using the discrete domain relation (4.7) and (4.8) one can obtain

$$J_{iz}^{n+\frac{1}{2}}(i,j,k) = J_{iz}^{n-\frac{1}{2}}(i,j,k) + \frac{\Delta t\Delta z}{L\Delta x\Delta y}E_z^n(i,j,k). \tag{4.23}$$

Since we were able to obtain an expression for $J_{iz}^{n+\frac{1}{2}}(i,j,k)$ in terms of the previous value of the electric field component $E_z^n(i,j,k)$ and the previous value of the impressed current

density component $J_{iz}^{n-\frac{1}{2}}(i, j, k)$, we can use the general form of the FDTD updating equation (1.28) without any modification, which is rewritten here for convenience as

$$\begin{aligned} E_z^{n+1}(i, j, k) = & C_{eze}(i, j, k) \times E_z^n(i, j, k) \\ & + C_{ezhy}(i, j, k) \times \left(H_y^{n+\frac{1}{2}}(i, j, k) - H_y^{n+\frac{1}{2}}(i-1, j, k)\right) \\ & + C_{ezhx}(i, j, k) \times \left(H_x^{n+\frac{1}{2}}(i, j, k) - H_x^{n+\frac{1}{2}}(i, j-1, k)\right) \\ & + C_{ezj}(i, j, k) \times J_{iz}^{n+\frac{1}{2}}(i, j, k), \end{aligned} \tag{4.24}$$

where

$$\begin{aligned} C_{eze}(i, j, k) &= \frac{2\varepsilon_z(i, j, k) - \Delta t\sigma_z^e(i, j, k)}{2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)}, \\ C_{ezhy}(i, j, k) &= \frac{2\Delta t}{\left(2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)\right)\Delta x}, \\ C_{ezhx}(i, j, k) &= -\frac{2\Delta t}{\left(2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)\right)\Delta y}, \\ C_{ezj}(i, j, k) &= -\frac{2\Delta t}{2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)}. \end{aligned}$$

One should notice that during the FDTD time-marching iteration, at every time step the new value of $J_{iz}^{n+\frac{1}{2}}(i, j, k)$ should be calculated by (4.23) using $E_z^n(i, j, k)$ and $J_{iz}^{n-\frac{1}{2}}(i, j, k)$ before updating $E_z^{n+1}(i, j, k)$ by (4.24).

4.1.7 Lumped elements distributed over a surface or within a volume

In previous sections, we have demonstrated the derivation of FDTD updating equations for common lumped element circuit components. These components were assumed to be placed between two neighboring nodes along the edge of a cell oriented in a certain direction (x, y, or z). However, in some applications it may be desired to simulate a lumped element component behavior over a surface or within a volume, which extends to a number of cells in the discrete space.

For instance, consider the voltage source in Figure 4.5. Such a source can be used to feed a strip with a uniform potential V_s across its cross-section at its edge. Similarly, a resistor is illustrated in Figure 4.5, which is distributed over a volume and maintains a resistance R between the top and bottom strips. Similarly, the other types of lumped elements as well can be distributed over a surface or a volume. In such cases, the lumped element behavior extends over a number of cells, and the surfaces or volumes of the elements can be indicated by the indices of the nodes at the lower points and upper points of the cell groups. For instance, the voltage source is indicated by the nodes (*vis, vjs, vks*) and (*vie, vje, vke*), whereas the resistor is indicated by the nodes (*ris, rjs, rks*) and (*rie, rje, rke*), as illustrated in Figure 4.6.

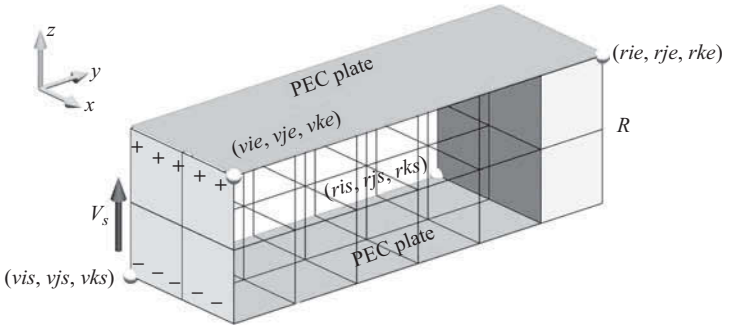

Figure 4.5 Parallel PEC plates excited by a voltage source at one end and terminated by a resistor at the other end.

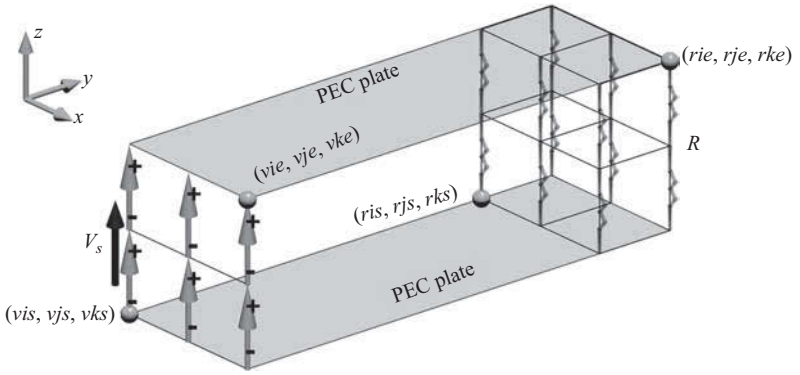

Figure 4.6 A voltage source distributed over a surface and a resistor distributed over a volume.

The lumped element updating equations derived in previous sections still can be used to model the elements distributed over a number of cells, but a modification of parameter values is required to maintain the respective values of the lumped elements. For instance, consider the voltage source in Figure 4.5. To simulate this source, the electric field components $E_z(vis : vie, vjs : vje, vks : vke - 1)$ need to be updated as shown in Figure 4.6. Hence, the total number of field components needing to be updated is $(vie - vis + 1) \times (vje - vjs + 1) \times (vke - vks)$. Therefore, the main voltage source can be replaced by multiple voltage sources, each of which is associated with a respective field component. Each individual voltage source in Figure 4.6 then should be assigned a voltage value V_s',

$$V_s' = \frac{V_s}{(vke - vks)}, \tag{4.25}$$

since the potential difference between the ends of the main voltage source in the z direction shall be kept the same. If the main voltage source has an internal resistance R_s, it should be maintained as well. Then the main resistance R_s will be represented by a network of $(vie - vis + 1) \times (vje - vjs + 1)$ resistors connected in parallel where each element in this

parallel combination is made of $(vke - vks)$ series resistors. Therefore, the resistance for each source component R_s' shall be set as

$$R_s' = R_s \times \frac{(vie - vis + 1) \times (vje - vjs + 1)}{(vke - vks)}. \tag{4.26}$$

Having applied these modifications to V_s and R_s, the electric field components $E_z(vis : vie, vjs : vje, vks : vke - 1)$ can be updated using (4.10).

The same procedure holds for each individual resistor R' at the other end of the parallel PEC plates. Thus, the resistance R' can be given as

$$R' = R \times \frac{(rie - ris + 1) \times (rje - rjs + 1)}{(rke - rks)}, \tag{4.27}$$

and (4.16) can be used to update the electric field components $E_z(ris : rie, rjs : rje, rks : rke - 1)$.

Similarly, for an inductor with inductance L extending between the nodes (is, js, ks) and (ie, je, ke), the individual inductance L' can be defined as

$$L' = L \times \frac{(ie - is + 1) \times (je - js + 1)}{(ke - ks)}, \tag{4.28}$$

and (4.24) can be used to update the electric field components $E_z(is : ie, js : je, ks : ke - 1)$.

For a capacitor with capacitance C extending between the nodes (is, js, ks) and (ie, je, ke), each individual capacitance C' can be defined as

$$C' = C \times \frac{(ke - ks)}{(ie - is + 1) \times (je - js + 1)}, \tag{4.29}$$

and (4.19) can be used to update the electric field components $E_z(is : ie, js : je, ks : ke - 1)$.

A current source with magnitude I_s and resistance R_s extending between the nodes (is, js, ks) and (ie, je, ke) can be modeled by updating $E_z(is : ie, js : je, ks : ke - 1)$ using (4.15) after modifying I_s and resistance R_s such that

$$I_s' = \frac{I_s}{(ie - is + 1) \times (je - js + 1)}, \tag{4.30}$$

and

$$R_s' = R_s \times \frac{(vie - vis + 1) \times (vje - vjs + 1)}{(vke - vks)}. \tag{4.31}$$

One should notice that the equations given for modifying the lumped values are all for the elements oriented in the z direction. The indexing scheme should be reflected appropriately to obtain equations for lumped elements oriented in other directions.

4.1.8 Diode

Derivation of the FDTD updating equations of some lumped element circuit components have been presented in previous sections. These circuit components are characterized by

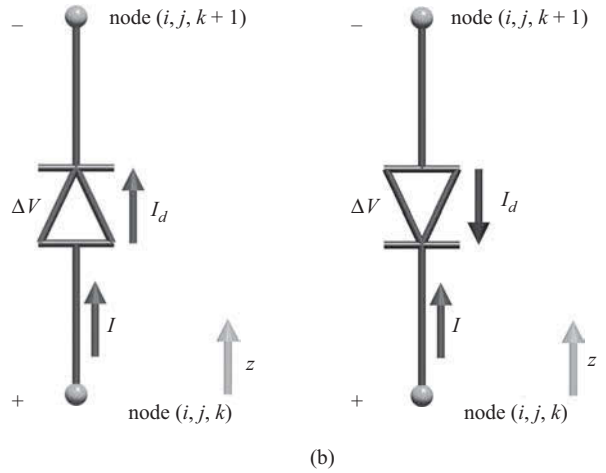

Figure 4.7 Diodes placed between nodes (i, j, k) and $(i, j, k + 1)$: (a) diode oriented in the positive z direction and (b) diode oriented in the negative z direction.

linear voltage–current relations, and it is straightforward to model their behavior in FDTD by properly expressing their voltage–current relations in discrete time and space. However, one of the strengths of the FDTD method is that it is possible to model nonlinear components as well. In this section we present the derivation of the updating equations for modeling a diode.

Figure 4.7(a) illustrates a diode placed between nodes (i, j, k) and $(i, j, k + 1)$ in an FDTD problem space. This diode allows currents flowing only in the positive z direction, as indicated by direction of the current I_d and characterized by the voltage–current relation

$$I = I_d = I_0\left[e^{(qV_d/kT)} - 1\right], \tag{4.32}$$

where q is the absolute value of electron charge in coulombs, k is Boltzmann's constant, and T is the absolute temperature in kelvins. This equation can be expressed in discrete form as

$$J_{iz}^{n+\frac{1}{2}}(i,\, j,\, k) = \frac{I_0}{\Delta x \Delta y}\left[e^{(q\,\Delta z/2kT)(E_z^{n+1}(i,j,k)+E_z^n(i,j,k))} - 1\right], \tag{4.33}$$

where the relation $V_d = \Delta V = \Delta z E_z$ is used. Equation 4.33 can be used in (4.5), which yields

$$\begin{aligned}\frac{E_z^{n+1}(i,\, j,\, k) - E_z^n(i,\, j,\, k)}{\Delta t} = {} & \frac{1}{\varepsilon_z(i,\, j,\, k)}\frac{H_y^{n+\frac{1}{2}}(i,\, j,\, k) - H_y^{n+\frac{1}{2}}(i-1,\, j,\, k)}{\Delta x} \\ & - \frac{1}{\varepsilon_z(i,\, j,\, k)}\frac{H_x^{n+\frac{1}{2}}(i,\, j,\, k) - H_x^{n+\frac{1}{2}}(i,\, j-1,\, k)}{\Delta y} \\ & - \frac{\sigma_z^e(i,\, j,\, k)}{2\varepsilon_z(i,\, j,\, k)}\left(E_z^{n+1}(i,\, j,\, k) + E_z^n(i,\, j,\, k)\right) \\ & - \frac{I_0}{\varepsilon_z(i,\, j,\, k)\Delta x \Delta y}\left[e^{(q\Delta z/2kT)\left(E_z^{n+1}(i,\, j,\, k)+E_z^n(i,\, j,\, k)\right)} - 1\right].\end{aligned} \tag{4.34}$$

This equation can be expanded as

$$\begin{aligned}E_z^{n+1}(i,j,k)+\frac{\sigma_z^e(i,j,k)\Delta t}{2\varepsilon_z(i,j,k)}E_z^{n+1}(i,j,k) = E_z^n(i,j,k)-\frac{\sigma_z^e(i,j,k)\Delta t}{2\varepsilon_z(i,j,k)}E_z^n(i,j,k)\\ +\frac{\Delta t}{\varepsilon_z(i,j,k)}\frac{H_y^{n+\frac{1}{2}}(i,j,k)-H_y^{n+\frac{1}{2}}(i-1,j,k)}{\Delta x}\\ -\frac{\Delta t}{\varepsilon_z(i,j,k)}\frac{H_x^{n+\frac{1}{2}}(i,j,k)-H_x^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y}\\ -\frac{I_0\Delta t}{\varepsilon_z(i,j,k)\Delta x\Delta y}e^{(q\Delta z/2kT)E_z^n(i,j,k)}e^{(q\Delta z/2kT)E_z^{n+1}(i,j,k)}\\ +\frac{I_0\Delta t}{\varepsilon_z(i,j,k)\Delta x\Delta y}\end{aligned} \tag{4.35}$$

and can be arranged in the following form

$$Ae^{Bx}+x+C=0, \tag{4.36}$$

where

$$\begin{aligned}&x=E_z^{n+1}(i,j,k),\quad A=-C_{ezd}(i,j,k)e^{B\times E_z^n(i,j,k)},\quad B=(q\Delta z/2kT),\\ &C=C_{eze}(i,j,k)\times E_z^n(i,j,k)+C_{ezhy}(i,j,k)\times\left(H_y^{n+\frac{1}{2}}(i,j,k)-H_y^{n+\frac{1}{2}}(i-1,j,k)\right)\\ &\quad+C_{ezhx}(i,j,k)\times\left(H_x^{n+\frac{1}{2}}(i,j,k)-H_x^{n+\frac{1}{2}}(i,j-1,k)\right)+C_{ezd}(i,j,k),\\ &C_{eze}(i,j,k)=-\frac{2\varepsilon_z(i,j,k)-\Delta t\sigma_z^e(i,j,k)}{2\varepsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)},\\ &C_{ezhy}(i,j,k)=-\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)\right)\Delta x},\\ &C_{ezhx}(i,j,k)=\frac{2\Delta t}{\left(2\varepsilon_z(i,j,k)+\Delta t\sigma_z^e(i,j,k)\right)\Delta y},\\ &C_{ezd}(i,j,k)=-\frac{2\Delta tI_0}{2\varepsilon_z(i,j,k)\Delta x\Delta y+\Delta t\sigma_z^e(i,j,k)\Delta x\Delta y}.\end{aligned}$$

Here the parameters A and C are dependent on $E_z^n(i,j,k)$. Therefore, at every time step the terms A and C must be calculated using the past values of magnetic and electric field components, and then (4.36) must be solved to obtain $E_z^{n+1}(i,j,k)$.

Equation (4.36) can easily be solved numerically by using the Newton–Raphson method, which is one of the most widely used methods for finding roots of a nonlinear function. It is an iterative process that starts from an initial point in the proximity of a root and approaches to the root by the use of the derivative of the function. For instance, Figure 4.8 shows a function for which we want to find the value of x that satisfies $f(x)=0$. We can start with an initial guess x_0 as shown in the figure. The derivative of the function $f(x)$ at point x_0 is

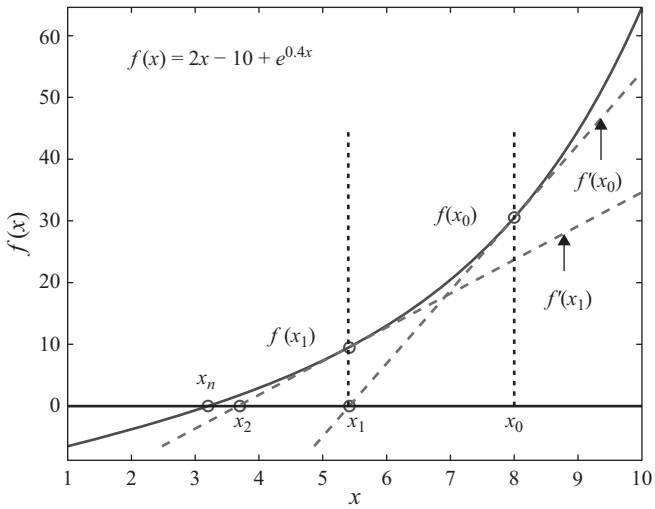

Figure 4.8 A function $f(x)$ and the points approaching to the root of the function iteratively calculated using the Newton–Raphson method.

the slope of the tangential line passing through the point $f(x_0)$ and intersecting the x axis at point x_1. We can easily calculate point x_1 as

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \tag{4.37}$$

Then x_1 can be used as the reference point, and a second point x_2 can be obtained similarly by

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}. \tag{4.38}$$

As can be followed from Figure 4.8, the new calculated x leads to the target point where $f(x)$ intersects with the x axis. Due to the high convergence rate of this procedure a point x_n can be achieved after couple of iterations, which is in the very close proximity of the exact root of the function $f(x)$. This iterative procedure can be continued until a stopping criteria or a maximum number of iterations is reached. Such a stopping criteria can be given by

$$|f(x_n)| < \varepsilon, \tag{4.39}$$

where ε is a very small positive number accepted as the error tolerance accepted for $f(x)$ approximately equals to zero.

It is very convenient to use the Newton–Raphson method to solve the diode equation (4.36), since the nature of this equation does not allow local minima or maxima, which can be an obstacle for the convergence of this method. Furthermore, the value of the electric field component $E_z^n(i,\ j,\ k)$ can be used as the initial guess for $E_z^{n+1}(i,\ j,\ k)$. Since the value of E_z (i, j, k) will change only a small amount between the consecutive time steps, the initial guess

will be fairly close to the target value, and it would take only a couple of Newton–Raphson iterations to reach the desired solution.

The diode equation given by (4.36) is for a positive z-directed diode illustrated in Figure 4.7(a), which is characterized by the voltage–current relation (4.32). A diode oriented in the negative z direction, as shown in 4.7(b), can be characterized by the voltage–current relation

$$I = -I_d = -I_0\left[e^{(qV_d/kT)} - 1\right] \tag{4.40}$$

and $V_d = -\Delta V$ due to the inversion of the diode polarity. Therefore, one can reconstruct (4.34) and (4.36) considering the reversed polarities of V_d and I_d, which yields the updating equation for a negative z-directed diode as

$$Ae^{Bx} + x + C = 0, \tag{4.41}$$

where

$$x = E_z^{n+1}(i, j, k), \quad A = -C_{ezd}(i, j, k)e^{B\times E_z^n(i, j, k)}, \quad B = -(q\Delta z/2kT),$$

$$C = C_{eze}(i, j, k) \times E_z^n(i, j, k) + C_{ezhy}(i, j, k) \times \left(H_y^{n+\frac{1}{2}}(i, j, k) - H_y^{n+\frac{1}{2}}(i-1, j, k)\right)$$

$$+ C_{ezhx}(i, j, k) \times \left(H_x^{n+\frac{1}{2}}(i, j, k) - H_x^{n+\frac{1}{2}}(i, j-1, k)\right) + C_{ezd}(i, j, k),$$

$$C_{eze}(i, j, k) = -\frac{2\varepsilon_z(i, j, k) - \Delta t\sigma_z^e(i, j, k)}{2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)},$$

$$C_{ezhy}(i, j, k) = -\frac{2\Delta t}{\left(2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)\right)\Delta x},$$

$$C_{ezhx}(i, j, k) = \frac{2\Delta t}{\left(2\varepsilon_z(i, j, k) + \Delta t\sigma_z^e(i, j, k)\right)\Delta y},$$

$$C_{ezd}(i, j, k) = \frac{2\Delta t I_0}{2\varepsilon_z(i, j, k)\Delta x\Delta y + \Delta t\sigma_z^e(i, j, k)\Delta x\Delta y}.$$

Here, one should notice that only the coefficients B and $C_{ezd}(i, j, k)$ are reversed when compared with the respective coefficients of (4.36), and the rest of the terms are the same.

Due to the nonlinear nature of the diode voltage–current relation, it is not convenient to define a diode extending over a number of cells in the FDTD problem grid. If a diode is going to be placed between two nodes that are apart from each other more than one cell size, it is more convenient to place the diode between two neighboring nodes and then to establish connection between the other nodes by other means, (e.g., by thin wires). The thin-wires FDTD updating procedure and equations are discussed in Chapter 10.

4.1.9 Summary

In this chapter, we have provided the derivation of FDTD updating equations for modeling some common lumped element circuit components. We have shown the construction of

updating equations for components placed between two neighboring nodes and have illustrated how these components can be modeled if they extend over a number of cells on the problem grid.

It is possible to obtain updating equations for the circuits composed of combinations of these lumped elements; one only needs to obtain the appropriate voltage–current relations, express them in discrete time and space, and establish the relation between the impressed current densities and the field components. Then impressed current density terms can be used in the general form of the updating equation to obtain the specific updating equations that model the lumped element circuit under consideration.

4.2 Definition, initialization, and simulation of lumped elements

We discussed modeling of several types of lumped element components in the FDTD method in previous sections. In this section we demonstrate the implementation of the aforementioned concepts in MATLAB. Furthermore, we show the implementation of other routines that initialize the FDTD problem space, including some auxiliary parameters, field arrays, and updating coefficient arrays. Then the MATLAB workspace will be ready to run an FDTD simulation, and we show how the FDTD time-marching loop can be implemented and some sample results of interest can be obtained.

4.2.1 Definition of lumped elements

First we show the implementation of the definition of the lumped element components. As discussed before, these components can be defined as prism-like objects distributed over a volume in the FDTD problem space. Any prism-like object can be defined with its lower and upper coordinates in the Cartesian coordinate system. Therefore, we define the positions of the lumped components the same as the brick object, as demonstrated in Section 3.1. Referring to the FDTD solver main program ***fdtd_solve***, which is shown in Listing 3.1, the implementation of the definition of lumped element components is done in the subroutine ***define_sources_and_lumped_elements***, a sample content of which is given as a template in Listing 4.1. As can be seen in Listing 4.1, the structure arrays **voltage_sources**, **current_sources**, **resistors**, **inductors**, **capacitors**, and **diodes** are defined to store the properties of the respective types of lumped components, and these arrays are initialized as empty arrays. Similar to a brick, the positions and dimensions of these components are indicated by the parameters **min_x**, **min_y**, **min_z**, **max_x**, **max_y**, and **max_z**. One should be careful when defining the coordinates of diodes; as discussed in Section 4.8, we assume diodes as objects having zero thickness in two dimensions.

Besides the parameters defining the positioning, some additional parameters as well are needed to specify the properties of these components. The lumped element components are modeled in FDTD by the way the respective electric field components are updated due to the voltage–current relations. These field components that need specific updates are determined by the functional directions of these components. For voltage sources, current sources, and diodes there are six directions in which they can be defined in the staircased FDTD grid: "***xn***," "***xp***," "***yn***," "***yp***," "***zn***," or "***zp***." Here "***p***" refers to positive direction, whereas "***n***"

refers to negative direction. The other lumped elements (resistors, capacitors, and inductors) can be defined with the directions "***x***," "***y***," or "***z***." The other parameters needed to specify the lumped components are **magnitude**, **resistance**, **inductance**, and **capacitance**, which represent the characteristics of the lumped elements as shown in Listing 4.1.

Listing 4.1 define_sources_and_lumped_elements.m

```
disp('defining sources and lumped element components');

voltage_sources = [];
current_sources = [];
diodes = [];
resistors = [];
inductors = [];
capacitors = [];

% define source waveform types and parameters
waveforms.sinusoidal(1).frequency = 1e9;
waveforms.sinusoidal(2).frequency = 5e8;
waveforms.unit_step(1).start_time_step = 50;

% voltage sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude   : volts
voltage_sources(1).min_x = 0;
voltage_sources(1).min_y = 0;
voltage_sources(1).min_z = 0;
voltage_sources(1).max_x = 1.0e-3;
voltage_sources(1).max_y = 2.0e-3;
voltage_sources(1).max_z = 4.0e-3;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'sinusoidal';
voltage_sources(1).waveform_index = 2;

% current sources
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
% resistance : ohms, magitude   : amperes
current_sources(1).min_x = 30*dx;
current_sources(1).min_y = 10*dy;
current_sources(1).min_z = 10*dz;
current_sources(1).max_x = 36*dx;
current_sources(1).max_y = 10*dy;
current_sources(1).max_z = 13*dz;
current_sources(1).direction = 'xp';
current_sources(1).resistance = 50;
current_sources(1).magnitude = 1;
current_sources(1).waveform_type = 'unit_step';
current_sources(1).waveform_index = 1;

% resistors
% direction: 'x', 'y', or 'z'
% resistance : ohms
resistors(1).min_x = 7.0e-3;
resistors(1).min_y = 0;
resistors(1).min_z = 0;
```

```
resistors(1).max_x = 8.0e-3;
resistors(1).max_y = 2.0e-3;
resistors(1).max_z = 4.0e-3;
resistors(1).direction = 'z';
resistors(1).resistance = 50;

% inductors
% direction: 'x', 'y', or 'z'
% inductance : henrys
inductors(1).min_x = 30*dx;
inductors(1).min_y = 10*dy;
inductors(1).min_z = 10*dz;
inductors(1).max_x = 36*dx;
inductors(1).max_y = 10*dy;
inductors(1).max_z = 13*dz;
inductors(1).direction = 'x';
inductors(1).inductance = 1e-9;

% capacitors
% direction: 'x', 'y', or 'z'
% capacitance : farads
capacitors(1).min_x = 30*dx;
capacitors(1).min_y = 10*dy;
capacitors(1).min_z = 10*dz;
capacitors(1).max_x = 36*dx;
capacitors(1).max_y = 10*dy;
capacitors(1).max_z = 13*dz;
capacitors(1).direction = 'x';
capacitors(1).capacitance = 1e-12;

% diodes
% direction: 'xp', 'xn', 'yp', 'yn', 'zp', or 'zn'
diodes(1).min_x = 30*dx;
diodes(1).min_y = 10*dy;
diodes(1).min_z = 10*dz;
diodes(1).max_x = 36*dx;
diodes(1).max_y = 10*dy;
diodes(1).max_z = 13*dz;
diodes(1).direction = 'xp';
```

Another type of parameter that is required for time-domain simulation of sources is the type of waveforms that the voltages and current sources generate as a function of time. For each source, the voltage or current value generated at every time step can be stored as the waveform in a one-dimensional array with the size of **number_of_time_steps**. However, before associating a waveform to a source, we can define the parameters specific to various types of waveforms in a structure named **waveforms** as illustrated in Listing 4.1. There are various options for constructing a waveform in FDTD. Some of the most common types of waveforms are Gaussian, sinusoidal, cosine modulated Gaussian, and unit step. The definition and construction of these waveforms will be the subject of Chapter 5. However, we provide a template here showing how the waveform parameters can be defined. In Listing 4.1 two fields are defined in the parameter **waveforms**: **sinusoidal** and **unit_step**. Later on we can add new parameters representing other types of waveforms. The parameters representing the waveform types are arrays of structures as illustrated with the indices **waveforms.sinuoidal(i)** and **waveforms.unit_step(i)**. Each waveform type has parameters representing its specific characteristics.

Here **frequency** is a parameter for the sinusoidal waveform **waveforms.sinusoidal(i)**, whereas **start_time_step** is a parameter for the waveform **waveforms.unit_step(i)**.

Having defined the types of waveforms and their parameters, we can associate them to the sources by referring to them with the **waveform_type** and **waveform_index** in the source parameters **voltage_sources(i)** and **current_sources(i)**. In the example code, we assign the type of the waveform "***sinusoidal***" to **voltage_sources(1).waveform_type**, and the index of the sinusoidal waveform **waveforms.sinusoidal(2)**, which is 2, to **voltage_sources(1).waveform_index**. Similarly, the type of the waveform "***unit_step***" is assigned to **current_sources(1).waveform_type** and the index of the waveform **waveforms.unit_step(1)**, which is 1, is assigned to **current_sources(1).waveform_index**.

4.2.2 Initialization of FDTD parameters and arrays

Some constant parameters and auxiliary parameters are needed in various stages of an FDTD program. These parameters can be defined in the subroutine ***initialize_fdtd_parameters_and_arrays***, which is shown in Listing 4.2. The most frequently used ones of these parameters are permittivity, permeability of free space, and speed of light in free space, which are represented in the code with the parameters **eps_0**, **mu_0**, and **c**, respectively. Duration of a time step is denoted by **dt** and is calculated based on the CFL stability limit (2.7) and the **courant_factor** described in Section 3.1.1. Once the duration of a time step is calculated, an auxiliary one-dimensional array, **time**, of size **number_of_time_steps** can be constructed to store the time instant values at the middle of the FDTD time steps. When a new simulation starts, the initial values of electric and magnetic fields are zero. Therefore, the three-dimensional arrays representing the vector components of the fields can be created and initialized using the MATLAB function **zeros** as illustrated in Listing 4.2. One can

Listing 4.2 initialize_fdtd_parameters_and_arrays.m

```
disp('initializing FDTD parameters and arrays');

% constant parameters
eps_0 = 8.854187817e-12; % permittivity of free space
mu_0  = 4*pi*1e-7; % permeability of free space
c = 1/sqrt(mu_0*eps_0); % speed of light in free space

% Duration of a time step in seconds
dt = 1/(c*sqrt((1/dx^2)+(1/dy^2)+(1/dz^2)));
dt = courant_factor*dt;

% time array
time = ([1:number_of_time_steps]-0.5)*dt;

% Create and initialize field and current arrays
disp('creating field arrays');

Hx = zeros(nxp1,ny,nz);
Hy = zeros(nx,nyp1,nz);
Hz = zeros(nx,ny,nzp1);
Ex = zeros(nx,nyp1,nzp1);
Ey = zeros(nxp1,ny,nzp1);
Ez = zeros(nxp1,nyp1,nz);
```

notice that the sizes of the arrays are not the same due to the positioning scheme of field components on the Yee grid as discussed in Section 3.5.

At this point one can ask if there are any arrays that need to be defined to represent impressed currents. The answer is that we do not need to define any arrays representing the impressed currents unless they are explicitly needed in an FDTD simulation. Throughout Chapter 4 we have shown that the concept of impressed currents is used to establish the voltage–current relations for the lumped element components and the final updating equations do not include the impressed current terms except for the inductor. Therefore, we do not need to create three-dimensional arrays expanded to the problem space to represent impressed current components. However, for the case of the inductor, there exists an impressed electric current term. Since an inductor would be expanding only over a couple of cells and only a small number of field components need to be updated using the inductor updating equations, it is sufficient to create auxiliary arrays representing these impressed currents that have the limited size based on the number of field components associated with the respective inductors. The creation and initialization of these arrays are performed in the subroutine where the updating coefficients of the lumped element components are initialized.

4.2.3 Initialization of lumped element components

We have shown templates for defining some types of source waveforms, voltage and current sources, and other lumped element components. The initialization of these components is performed in ***initialize_sources_and_lumped_elements*** as shown in Listing 4.3.

Listing 4.3 initialize_sources_and_lumped_elements.m

```
disp('initializing sources and lumped element components');

number_of_voltage_sources   = size(voltage_sources,2);
number_of_current_sources   = size(current_sources,2);
number_of_resistors   = size(resistors,2);
number_of_inductors   = size(inductors,2);
number_of_capacitors  = size(capacitors,2);
number_of_diodes      = size(diodes,2);

% initialize waveforms
initialize_waveforms;

% voltage sources
for ind = 1:number_of_voltage_sources
    is = round((voltage_sources(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((voltage_sources(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((voltage_sources(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((voltage_sources(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((voltage_sources(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((voltage_sources(ind).max_z - fdtd_domain.min_z)/dz)+1;
    voltage_sources(ind).is = is;
```

```
    voltage_sources(ind).js = js;
    voltage_sources(ind).ks = ks;
    voltage_sources(ind).ie = ie;
    voltage_sources(ind).je = je;
    voltage_sources(ind).ke = ke;

    switch (voltage_sources(ind).direction(1))
    case 'x'
        n_fields = ie - is;
        r_magnitude_factor = (1 + je - js) * (1 + ke - ks) / (ie - is);
    case 'y'
        n_fields = je - js;
        r_magnitude_factor = (1 + ie - is) * (1 + ke - ks) / (je - js);
    case 'z'
        n_fields = ke - ks;
        r_magnitude_factor = (1 + ie - is) * (1 + je - js) / (ke - ks);
    end
    if strcmp(voltage_sources(ind).direction(2),'n')
        v_magnitude_factor =  ...
            -1*voltage_sources(ind).magnitude/n_fields;
    else
        v_magnitude_factor =  ...
            1*voltage_sources(ind).magnitude/n_fields;
    end
    voltage_sources(ind).resistance_per_component = ...
        r_magnitude_factor * voltage_sources(ind).resistance;

    % copy waveform of the waveform type to waveform of the source
    wt_str = voltage_sources(ind).waveform_type;
    wi_str = num2str(voltage_sources(ind).waveform_index);
    eval_str = ['a_waveform = waveforms.' ...
        wt_str '(' wi_str ').waveform;'];
    eval(eval_str);
    voltage_sources(ind).voltage_per_e_field = ...
        v_magnitude_factor * a_waveform;
    voltage_sources(ind).waveform = ...
        v_magnitude_factor * a_waveform * n_fields;
end

% current sources
for ind = 1:number_of_current_sources
    is = round((current_sources(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((current_sources(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((current_sources(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((current_sources(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((current_sources(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((current_sources(ind).max_z - fdtd_domain.min_z)/dz)+1;
    current_sources(ind).is = is;
    current_sources(ind).js = js;
    current_sources(ind).ks = ks;
    current_sources(ind).ie = ie;
    current_sources(ind).je = je;
    current_sources(ind).ke = ke;
```

```
75
    switch (current_sources(ind).direction(1))
77  case 'x'
        n_fields = (1 + je - js) * (1 + ke - ks);
79      r_magnitude_factor = (1 + je - js) * (1 + ke - ks) / (ie - is);
    case 'y'
81      n_fields = (1 + ie - is) * (1 + ke - ks);
        r_magnitude_factor = (1 + ie - is) * (1 + ke - ks) / (je - js);
83  case 'z'
        n_fields = (1 + ie - is) * (1 + je - js);
85      r_magnitude_factor = (1 + ie - is) * (1 + je - js) / (ke - ks);
    end
87  if strcmp(current_sources(ind).direction(2),'n')
        i_magnitude_factor = ...
89          -1*current_sources(ind).magnitude/n_fields;
    else
91      i_magnitude_factor =  ...
            1*current_sources(ind).magnitude/n_fields;
93  end
    current_sources(ind).resistance_per_component = ...
95      r_magnitude_factor * current_sources(ind).resistance;

97  % copy waveform of the waveform type to waveform of the source
    wt_str = current_sources(ind).waveform_type;
99  wi_str = num2str(current_sources(ind).waveform_index);
    eval_str = ['a_waveform = waveforms.' ...
101     wt_str '(' wi_str ').waveform;'];
    eval(eval_str);
103 current_sources(ind).current_per_e_field = ...
        i_magnitude_factor * a_waveform;
105 current_sources(ind).waveform = ...
        i_magnitude_factor * a_waveform * n_fields;
107 end

109 % resistors
 for ind = 1:number_of_resistors
111     is = round((resistors(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((resistors(ind).min_y - fdtd_domain.min_y)/dy)+1;
113 ks = round((resistors(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((resistors(ind).max_x - fdtd_domain.min_x)/dx)+1;
115 je = round((resistors(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((resistors(ind).max_z - fdtd_domain.min_z)/dz)+1;
117 resistors(ind).is = is;
    resistors(ind).js = js;
119 resistors(ind).ks = ks;
    resistors(ind).ie = ie;
121 resistors(ind).je = je;
    resistors(ind).ke = ke;
123 switch (resistors(ind).direction)
    case 'x'
125     r_magnitude_factor = (1 + je - js) * (1 + ke - ks) / (ie - is);

    case 'y'
127     r_magnitude_factor = (1 + ie - is) * (1 + ke - ks) / (je - js);
    case 'z'
129     r_magnitude_factor = (1 + ie - is) * (1 + je - js) / (ke - ks);
    end
131 resistors(ind).resistance_per_component = ...
        r_magnitude_factor * resistors(ind).resistance;
133 end
```

```
% inductors
for ind = 1:number_of_inductors
    is = round((inductors(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((inductors(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((inductors(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((inductors(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((inductors(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((inductors(ind).max_z - fdtd_domain.min_z)/dz)+1;
    inductors(ind).is = is;
    inductors(ind).js = js;
    inductors(ind).ks = ks;
    inductors(ind).ie = ie;
    inductors(ind).je = je;
    inductors(ind).ke = ke;
    switch (inductors(ind).direction)
    case 'x'
        l_magnitude_factor = (1 + je - js) * (1 + ke - ks) / (ie - is);
    case 'y'
        l_magnitude_factor = (1 + ie - is) * (1 + ke - ks) / (je - js);
    case 'z'
        l_magnitude_factor = (1 + ie - is) * (1 + je - js) / (ke - ks);
    end
    inductors(ind).inductance_per_component = ...
        l_magnitude_factor * inductors(ind).inductance;
end

% capacitors
for ind = 1:number_of_capacitors
    is = round((capacitors(ind).min_x - fdtd_domain.min_x)/dx)+1;
    js = round((capacitors(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((capacitors(ind).min_z - fdtd_domain.min_z)/dz)+1;
    ie = round((capacitors(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((capacitors(ind).max_y - fdtd_domain.min_y)/dy)+1;
    ke = round((capacitors(ind).max_z - fdtd_domain.min_z)/dz)+1;
    capacitors(ind).is = is;
    capacitors(ind).js = js;
    capacitors(ind).ks = ks;
    capacitors(ind).ie = ie;
    capacitors(ind).je = je;
    capacitors(ind).ke = ke;
    switch (capacitors(ind).direction)
    case 'x'
        c_magnitude_factor = (ie - is) ...
            / ((1 + je - js) * (1 + ke - ks));
    case 'y'
        c_magnitude_factor = (je - js) ...
            / ((1 + ie - is) * (1 + ke - ks));
    case 'z'
        c_magnitude_factor = (ke - ks) ...
            / ((1 + ie - is) * (1 + je - js));
    end
    capacitors(ind).capacitance_per_component = ...
        c_magnitude_factor * capacitors(ind).capacitance;
end

sigma_pec = material_types(material_type_index_pec).sigma_e;
```

```
191
% diodes
193 for ind = 1:number_of_diodes
    is = round((diodes(ind).min_x - fdtd_domain.min_x)/dx)+1;
195 js = round((diodes(ind).min_y - fdtd_domain.min_y)/dy)+1;
    ks = round((diodes(ind).min_z - fdtd_domain.min_z)/dz)+1;
197 ie = round((diodes(ind).max_x - fdtd_domain.min_x)/dx)+1;
    je = round((diodes(ind).max_y - fdtd_domain.min_y)/dy)+1;
199 ke = round((diodes(ind).max_z - fdtd_domain.min_z)/dz)+1;
    diodes(ind).is = is;
201 diodes(ind).js = js;
    diodes(ind).ks = ks;
203 diodes(ind).ie = ie;
    diodes(ind).je = je;
205 diodes(ind).ke = ke;

207 switch (diodes(ind).direction(1))
        case 'x'
209         sigma_e_x(is+1:ie-1,js,ks) = sigma_pec;
        case 'y'
211         sigma_e_y(is,js+1:je-1,ks) = sigma_pec;
        case 'z'
213         sigma_e_z(is,js,ks+1:ke-1) = sigma_pec;
    end
215 end
```

Listing 4.4 initialize_waveforms.m

```
1 disp('initializing source waveforms');

3 % initialize sinusoidal waveforms
  for ind=1:size(waveforms.sinusoidal,2)
5     waveforms.sinusoidal(ind).waveform = ...
          sin(2 * pi * waveforms.sinusoidal(ind).frequency * time);
7 end

9 % initialize unit step waveforms
  for ind=1:size(waveforms.unit_step,2)
11    start_index = waveforms.unit_step(ind).start_time_step;
      waveforms.unit_step(ind).waveform(1:number_of_time_steps) = 1;
13    waveforms.unit_step(ind).waveform(1:start_index-1) = 0;
  end
```

Before initializing the sources, the source waveforms need to be initialized. Therefore, a subroutine, ***initialize_waveforms***, is called in Listing 4.3 before initialization of lumped components. The sample implementation of ***initialize_waveforms*** is given in Listing 4.4, and this subroutine will be expanded as new source waveform types are discussed in the following chapter. In subroutine ***initialize_waveforms*** for every waveform type the respective waveforms are calculated as functions of time based on the waveform type specific parameters. Then the calculated waveforms are stored in the arrays with the associated name **waveform**. For instance, in Listing 4.4 the parameter **waveforms.sinusoidal(i).waveform** is

constructed as a sinusoidal waveform using $sin(2\pi \times f \times t)$, where f is the frequency defined with the parameter **waveforms.sinusoidal(i).frequency** and t is the discrete time array **time** storing the time instants of the time steps.

Once the waveform arrays are constructed for the defined waveform types they can be copied to **waveform** fields of the structures of sources associated with them as demonstrated in Listing 4.3. For example, consider the code section in the loop initializing the **voltage_sources**. The type of source waveform is stored in **voltage_sources(ind).waveform_type** as a string, and the index of the source waveform is stored in **voltage_sources(ind).waveform_index** as an integer number. Here we employ the MATLAB function **eval**, which executes a string containing MATLAB expression. We construct a string using the following expression:

```
wt_str = voltage_sources(ind).waveform_type;
wi_str = num2str(voltage_sources(ind).waveform_index);
eval_str = ['a_waveform = waveforms.' wt_str '(' wi_str ').waveform;'];
```

which constructs the string **eval_str** in the MATLAB workspace for the waveforms and the voltage source **voltage_sources(1)** defined in Listing 4.1 as

```
eval_str = 'a_waveform = waveforms.sinusoidal(2).waveform;'
```

Executing this string using ***eval*** as

```
eval(eval_str);
```

creates a parameter **a_waveform** in the MATLAB workspace, which has a copy of the waveform that the voltage source **voltage_sources(1)** is associated with. Then we can assign **a_waveform** to **voltage_sources(1).waveform** after multiplying with a factor. This multiplication factor is explained in the following paragraphs.

As discussed in Section 4.7, all the lumped element components except the diodes are assumed to be distributed over surfaces or volumes extending over a number of cells. Therefore, a number of field components require special updating due to the lumped elements associated with them. We assume that the surfaces or volumes that the lumped elements are coinciding with are conforming to the cells composing the FDTD problem space. Then we can identify the field components that are associated with lumped components by the indices of the *start* node, (*is*, *js*, *ks*), coinciding with the lower coordinates of the lumped component, and the indices of the *end* node, (*ie*, *je*, *ke*), coinciding with the upper coordinates of the lumped component, as illustrated in Figure 4.6. Therefore, for every lumped element, the start and end node indices are calculated and stored in the parameters **is**, **js**, **ks**, **ie**, **je**, and **ke** as shown in Listing 4.3.

Based on the discussion in Section 4.7, if a lumped element is associated with more than one field component, the value of the lumped element should be assigned to the field components after a modification. Therefore, new parameters are defined in the structures representing the lumped elements, which store the value of the lumped element per field

component. For instance, **resistance_per_component** is defined in **voltage_sources(i)**, which stores the resistance of the voltage source per field component as calculated using the form of (4.26) by taking into account the direction of the component. Similarly, for a resistor **resistance_per_component** is defined and calculated by (4.27). For a current source **resistance_per_component** is defined and calculated by (4.31). The parameter **inductance_per_component** is calculated by (4.28) for an inductor, and **capacitance_per_component** is calculated by (4.29) for a capacitor.

Similarly, the **voltage_sources(i).voltage_per_e_field** is scaled using (4.25), and **current_sources(i).current_per_e_field** is scaled using (4.30). Furthermore, directions of these components as well are taken into account in the scaling factors; if the direction is *negative* the scaling factor is multiplied by -1. Meanwhile, parameters **voltage_sources(i).waveform** and **current_sources(i).waveform** are constructed to store the main voltages and currents of the lumped sources rather than the values used to update the associated field components.

We assume that a diode is defined as a one-dimensional object, such as a line section, extending over a number of electric field components on the cell edges as illustrated in Figure 4.9. Here the diode is extending between the nodes (is, js, ks) and (ie, je, ke) – hence the field components $E_z(is, js, ks : ke - 1)$, where $ie = is$ and $je = js$. However, as discussed in Section 4.8, we update only one of these field components, in this case $E_z(is, js, ks)$, using the diode updating equations, and we establish an electrical connection between other nodes $(is, js, ks + 1)$ and (ie, je, ke). The easiest way of establishing the electrical connection is considering a PEC line between these nodes. Therefore, we can assign conductivity of PEC to the material components associated with the electric field components $E_z(is, js, ks + 1 : ke - 1)$, which are $\sigma_z^e(is, js, ks + 1 : ke - 1)$. Therefore, the necessary material

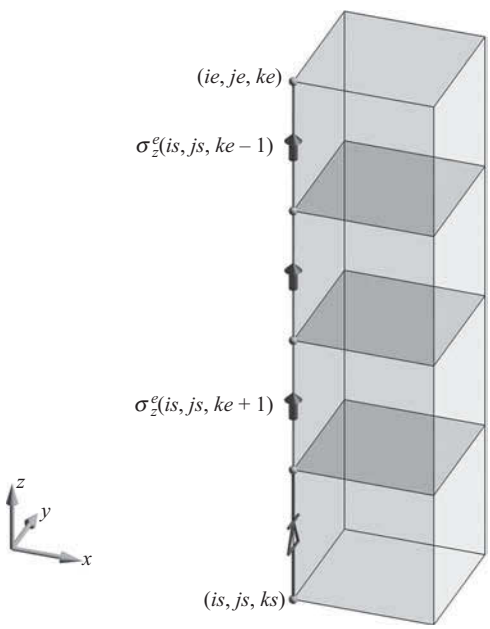

Figure 4.9 A diode defined between the nodes (is, js, ks) and (ie, je, ke).