CHAPTER 8

# Advanced PML formulations

In the previous chapter, we discussed the perfectly matched layers (PMLs) as a boundary to terminate the finite-difference time-domain (FDTD) lattices to simulate open boundary problems. However, the PML is shown to be ineffective for absorbing evanescent waves. As a result, the PML must be placed sufficiently far from an obstacle such that the evanescent waves have sufficiently decayed [23]. However, this increases the number of cells in an FDTD computational domain and, hence, the computational memory and time requirements. Another problem reported for PML is that it suffers from late-time reflections when terminating highly elongated lattices or when simulating fields with very long time signatures [23]. This is partly due to the weakly causal nature of the PML [24].

A strictly causal form of the PML, which is referred to as the complex frequency-shifted PML (CFS-PML) was developed in [25]. It has been shown that the CFS-PML is highly effective at absorbing evanescent waves and signals with a long time signature. Therefore, using the CFS-PML, the boundaries can be placed closer to the objects in the problem space and a time and memory saving can be achieved, thus avoiding the aforementioned weaknesses of the PML. An efficient implementation of the CFS-PML, known as the convolutional PML (CPML), is introduced in [23]. The theoretical analyses of the CPML and other forms of PML can be found in [26]. In this chapter we introduce the formulation of the CPML based on [23] and provide a MATLAB® implementation of the concept.

## 8.1 Formulation of CPML

In Chapter 7 we provided the PML equations for terminating a problem space surrounded by free space. However, in general a PML can be constructed to terminate a problem space surrounded by an arbitrary media. Furthermore, it can even simulate infinite length structures; these structures penetrate into the PML, and the PML absorbs the waves traveling on them.

### 8.1.1 PML in stretched coordinates

Without loss of generality, the PML equations for a lossy medium are posed in the stretched coordinate space [27] as

$$j\omega\varepsilon_x E_x + \sigma_x^e E_x = \frac{1}{S_{ey}}\frac{\partial H_z}{\partial y} - \frac{1}{S_{ez}}\frac{\partial H_y}{\partial z}, \tag{8.1a}$$

$$j\omega\varepsilon_y E_y + \sigma_y^e E_y = \frac{1}{S_{ez}}\frac{\partial H_x}{\partial z} - \frac{1}{S_{ex}}\frac{\partial H_z}{\partial x}, \tag{8.1b}$$

$$j\omega\varepsilon_z E_z + \sigma_z^e E_z = \frac{1}{S_{ex}}\frac{\partial H_y}{\partial x} - \frac{1}{S_{ey}}\frac{\partial H_x}{\partial y}, \tag{8.1c}$$

where $S_{ex}$, $S_{ey}$, and $S_{ez}$ are the stretched coordinate metrics, and $\sigma_x^e$, $\sigma_y^e$, and $\sigma_z^e$ are the electric conductivities of the terminating media. One can notice that (8.1) is in the frequency domain with $e^{j\omega t}$ time-harmonic convention.

Equations (8.1) reduce to Berenger's PML in the case where

$$S_{ex} = 1 + \frac{\sigma_{pex}}{j\omega\varepsilon_0}, \quad S_{ey} = 1 + \frac{\sigma_{pey}}{j\omega\varepsilon_0}, \quad \text{and} \quad S_{ez} = 1 + \frac{\sigma_{pez}}{j\omega\varepsilon_0}. \tag{8.2}$$

Here $\sigma_{pex}$, $\sigma_{pey}$, and $\sigma_{pez}$ are the PML conductivities.

It should be noted that (8.1a)–(8.1c) and 8.2 follow the form given in [23], but the subscript notations have been modified to indicate the electric and magnetic parameters separately. This form of notation facilitates establishing the connection between the parameters in the formulations and their counterparts in program implementation.

The other three scalar equations that are used to construct the magnetic field update equations are

$$j\omega\mu_x H_x + \sigma_x^m H_x = -\frac{1}{S_{my}}\frac{\partial E_z}{\partial y} + \frac{1}{S_{mz}}\frac{\partial E_y}{\partial z}, \tag{8.3a}$$

$$j\omega\mu_y H_y + \sigma_y^m H_y = -\frac{1}{S_{mz}}\frac{\partial E_x}{\partial z} + \frac{1}{S_{mx}}\frac{\partial E_z}{\partial x}, \tag{8.3b}$$

$$j\omega\mu_z H_z + \sigma_z^m H_z = -\frac{1}{S_{mx}}\frac{\partial E_y}{\partial x} + \frac{1}{S_{my}}\frac{\partial E_x}{\partial y}. \tag{8.3c}$$

Equations (8.3) reduce to the PML in the case where

$$S_{mx} = 1 + \frac{\sigma_{pmx}}{j\omega\mu_0}, \quad S_{my} = 1 + \frac{\sigma_{pmy}}{j\omega\mu_0}, \quad \text{and} \quad S_{mz} = 1 + \frac{\sigma_{pmz}}{j\omega\mu_0}. \tag{8.4}$$

## 8.1.2 Complex stretching variables in CFS-PML

In the CPML method, the choice of the complex stretching variables follows the new definition proposed by Kuzuoglu and Mittra [25], such that

$$S_{ex} = 1 + \frac{\sigma_{pex}}{j\omega\varepsilon_0}, \quad \Rightarrow \quad S_{ex} = \kappa_{ex} + \frac{\sigma_{pex}}{\alpha_{ex} + j\omega\varepsilon_0}.$$

Then the complex stretching variables are given as

$$S_{ei} = \kappa_{ei} + \frac{\sigma_{pei}}{\alpha_{ei} + j\omega\varepsilon_0}, \quad S_{mi} = \kappa_{mi} + \frac{\sigma_{pmi}}{\alpha_{mi} + j\omega\mu_0}, \quad i = x,\ y, \text{ or } z,$$

where the parameters $\kappa_{ei}$, $\kappa_{mi}$, $\alpha_{ei}$, and $\alpha_{mi}$ are the new parameters taking the values

$$\kappa_{ei} \geq 1, \quad \kappa_{mi} \geq 1, \quad \alpha_{ei} \geq 0, \quad \text{and} \quad \alpha_{mi} \geq 0.$$

## 8.1.3 The matching conditions at the PML–PML interface

For zero reflection at the PML–PML interface, one should have

$$S_{ei} = S_{mi}. \tag{8.5}$$

This condition leads to

$$\kappa_{ei} = \kappa_{mi}, \tag{8.6a}$$

$$\frac{\sigma_{pei}}{\alpha_{ei} + j\omega\varepsilon_0} = \frac{\sigma_{pmi}}{\alpha_{mi} + j\omega\mu_0}. \tag{8.6b}$$

To satisfy (8.6b), we must have

$$\frac{\sigma_{pei}}{\varepsilon_0} = \frac{\sigma_{pmi}}{\mu_0}, \quad and \quad \frac{\alpha_{ei}}{\varepsilon_0} = \frac{\alpha_{mi}}{\mu_0}. \tag{8.7}$$

## 8.1.4 Equations in the time domain

As mentioned before, (8.1) and (8.3) are in frequency domain. We need to have them expressed in time to obtain the field updating equations from them. For instance, (8.1a) is expressed in the time domain as

$$\varepsilon_x \frac{\partial E_x}{\partial t} + \sigma_x^e E_x = \overline{S}_{ey} * \frac{\partial H_z}{\partial y} - \overline{S}_{ez} * \frac{\partial H_y}{\partial z}, \tag{8.8}$$

where $\overline{S}_{ey}$ is a function of time that is the inverse Laplace transform of $S_{ey}^{-1}$, and $\overline{S}_{ez}$ is the inverse Laplace transform of $S_{ez}^{-1}$. One should notice that the *product* operations in the frequency-domain equations (8.1) and (8.3) are expressed as *convolution* operations in the time domain.

The terms $\overline{S}_{ei}$ and $\overline{S}_{mi}$ are then given in open form as

$$\overline{S}_{ei}(t) = \frac{\delta(t)}{\kappa_{ei}} - \frac{\sigma_{pei}}{\varepsilon_0 \kappa_{ei}^2} e^{-\left(\frac{\sigma_{pei}}{\varepsilon_0 \kappa_{ei}} + \frac{\alpha_{pei}}{\varepsilon_0}\right)t} u(t) = \frac{\delta(t)}{\kappa_{ei}} + \xi_{ei}(t), \tag{8.9a}$$

$$\overline{S}_{mi}(t) = \frac{\delta(t)}{\kappa_{mi}} - \frac{\sigma_{pmi}}{\mu_0 \kappa_{mi}^2} e^{-\left(\frac{\sigma_{pmi}}{\mu_0 \kappa_{mi}} + \frac{\alpha_{pmi}}{\mu_0}\right)t} u(t) = \frac{\delta(t)}{\kappa_{mi}} + \xi_{mi}(t), \tag{8.9b}$$

where $\delta(t)$ is the unit impulse function and $u(t)$ is the unit step function. Inserting (8.9) in (8.8) leads to

$$\varepsilon_x \frac{\partial E_x}{\partial t} + \sigma_x^e E_x = \frac{1}{\kappa_{ey}} \frac{\partial H_z}{\partial y} - \frac{1}{\kappa_{ez}} \frac{\partial H_y}{\partial z} + \xi_{ey}(t) * \frac{\partial H_z}{\partial y} - \xi_{ez}(t) * \frac{\partial H_y}{\partial z}. \tag{8.10}$$

At this point, the central difference approximation of the derivatives can be used to express (8.10) in discrete time and space and then to obtain the field updating equation for $E_x^{n+1}$. However, (8.10) includes two convolution terms, and these terms also need to be expressed in discrete time and space before proceeding with the construction of the updating equations.

## 8.1.5 Discrete convolution

The convolution terms in (8.10) can be written in open form, for instance, as

$$\xi_{ey} * \frac{\partial H_z}{\partial y} = \int_{\tau=0}^{\tau=t} \xi_{ey}(\tau) \frac{\partial H_z(t-\tau)}{\partial y} d\tau. \tag{8.11}$$

In the discrete domain, (8.11) takes the form

$$\int_{\tau=0}^{\tau=t} \xi_{ey}(\tau)\frac{\partial H_z(t-\tau)}{\partial y}d\tau \simeq \sum_{m=0}^{m=n-1} Z_{0ey}(m)\Big(H_z^{n-m+\frac{1}{2}}(i,j,k) - H_z^{n-m+\frac{1}{2}}(i,j-1,k)\Big), \quad (8.12)$$

where

$$\begin{aligned} Z_{0ey}(m) &= \frac{1}{\Delta y}\int_{\tau=m\Delta_t}^{\tau=(m+1)\Delta_t} \xi_{ey}(\tau)d\tau = -\frac{\sigma_{pey}}{\Delta y \varepsilon_0 \kappa_{ey}^2}\int_{\tau=m\Delta_t}^{\tau=(m+1)\Delta_t} e^{-\left(\frac{\sigma_{pey}}{\varepsilon_0\kappa_{ey}}+\frac{\alpha_{ey}}{\varepsilon_0}\right)\tau} d\tau \\ &= a_{ey} e^{-\left(\frac{\sigma_{pey}}{\kappa_{ey}}+\alpha_{ey}\right)\frac{m\ \Delta t}{\varepsilon_0}} \end{aligned} \quad (8.13)$$

and

$$a_{ey} = \frac{\sigma_{pey}}{\Delta y\left(\sigma_{pey}\kappa_{ey} + \alpha_{ey}\kappa_{ey}^2\right)}\left[e^{-\left(\frac{\sigma_{pey}}{\kappa_{ey}}+\alpha_{ey}\right)\frac{\Delta t}{\varepsilon_0}} - 1\right]. \quad (8.14)$$

Having derived the expression for $Z_{0ey}(m)$, we can express the discrete convolution term in (8.12) with a new parameter $\psi_{exy}^{n+\frac{1}{2}}(i,j,k)$, such that

$$\psi_{exy}^{n+\frac{1}{2}}(i,j,k) = \sum_{m=0}^{m=n-1} Z_{0ey}(m)\Big(H_z^{n-m+\frac{1}{2}}(i,j,k) - H_z^{n-m+\frac{1}{2}}(i,j-1,k)\Big). \quad (8.15)$$

Here the subscript *exy* indicates that this term is updating $E_x$ and is associated with the derivative of the magnetic field term with respect to $y$. Then (8.10) can be written in discrete form as

$$\begin{aligned} &\varepsilon_x(i,j,k)\frac{E_x^{n+1}(i,j,k) - E_x^n(i,j,k)}{\Delta t} + \sigma_x^e(i,j,k)\frac{E_x^{n+1}(i,j,k) + E_x^n(i,j,k)}{2} \\ &= \frac{1}{\kappa_{ey}(i,j,k)}\frac{H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)}{\Delta y} \\ &- \frac{1}{\kappa_{ez}(i,j,k)}\frac{H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)}{\Delta z} \\ &+ \psi_{exy}^{n+\frac{1}{2}}(i,j,k) - \psi_{exz}^{n+\frac{1}{2}}(i,j,k). \end{aligned} \quad (8.16)$$

## 8.1.6 The recursive convolution method

As can be followed from (8.16), the parameter $\psi_{exy}$ has to be recalculated at every time step of the FDTD time-marching loop. However, observing expression (8.15) one can see that the values of the magnetic field component $H_z$ calculated at all previous time steps have to be readily available to perform the discrete convolution. This implies that all the previous history of $H_z$ must be stored in the computer memory, which is not feasible with the current computer resources. The recursive convolution technique, which is frequently used to overcome the same problem while modeling dispersive media in the FDTD method, is employed to obtain a new expression for $\psi_{exy}$ that does not require all the previous history of $H_z$.

The general form of the discrete convolution (8.15) can be written as

$$\psi(n) = \sum_{m=0}^{m=n-1} Ae^{mT}B(n-m), \tag{8.17}$$

where, for instance,

$$\begin{aligned} A &= a_{ey} \\ T &= -\left(\frac{\sigma_{ey}}{\kappa_{ey}} + \alpha_{ey}\right)\frac{\Delta t}{\varepsilon_0} \\ B &= H_z^{n-m+\frac{1}{2}}(i,j,k) - H_z^{n-m+\frac{1}{2}}(i,j-1,k). \end{aligned}$$

Equation (8.17) can be written in open form as

$$\psi(n) = AB(n) + Ae^{T}B(n-1) + Ae^{2T}B(n-2) + \cdots + Ae^{(n-2)T}B(2) + Ae^{(n-1)T}B(1). \tag{8.18}$$

We can write the same equation in open form for one previous time step value $\psi(n-1)$ as

$$\psi(n-1) = AB(n-1) + Ae^{T}B(n-2) + Ae^{2T}B(n-3) + \cdots + Ae^{(n-2)T}B(2) + Ae^{(n-2)T}B(1). \tag{8.19}$$

Comparing the right-hand side of (8.18) with (8.19), one can realize that the right-hand side of (8.18), except the first term, is nothing but a multiplication of $e^T$ by $\psi(n-1)$. Therefore, (8.18) can be rewritten as

$$\psi(n) = AB(n) + e^T\psi(n-1). \tag{8.20}$$

In this form only the previous time step value $\psi(n-1)$ is required to calculate the new value of $\psi(n)$. Hence, the need for the storage of all previous values is eliminated. Since the new value of $\psi(n)$ is calculated recursively in (8.20), this technique is known as recursive convolution.

After applying this technique to simplify (8.15), we obtain

$$\psi_{exy}^{n+\frac{1}{2}}(i,j,k) = b_{ey}\psi_{exy}^{n-\frac{1}{2}}(i,j,k) + a_{ey}\left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right), \tag{8.21}$$

where

$$a_{ey} = \frac{\sigma_{pey}}{\Delta y\left(\sigma_{pey}\kappa_{ey} + \alpha_{ey}\kappa_{ey}^2\right)}\left[b_{ey} - 1\right], \tag{8.22}$$

$$b_{ey} = e^{-\left(\frac{\sigma_{pey}}{\kappa_{ey}} + \alpha_{ey}\right)\frac{\Delta t}{\varepsilon_0}}. \tag{8.23}$$

## 8.2 The CPML algorithm

Examining the discrete domain equation (8.16), one can see that the form of the equation is the same as the one for a lossy medium except that two new auxiliary terms are added to the expression. This form of equation indicates that the CPML algorithm is independent of the material medium and can be extended for dispersive media, anisotropic media, or nonlinear media. In each of these cases, the left-hand side must be modified to treat the specific host medium. Yet the application of the CPML remains unchanged [23].

The FDTD flowchart can be modified to include the steps of the CPML as shown in Figure 8.1. The auxiliary parameters and coefficients required by the CPML are initialized before the time-marching loop. During the time-marching loop, at every time step, first the magnetic field components are updated in the problem space using the regular updating equation derived for the host medium. Then the CPML terms ($\psi_{mxy}$, $\psi_{mxz}$, $\psi_{myx}$, $\psi_{myz}$, $\psi_{mzx}$, $\psi_{mzy}$) are calculated using their previous time step values and the new electric field values. Afterward, these terms are added to their respective magnetic field components only at the

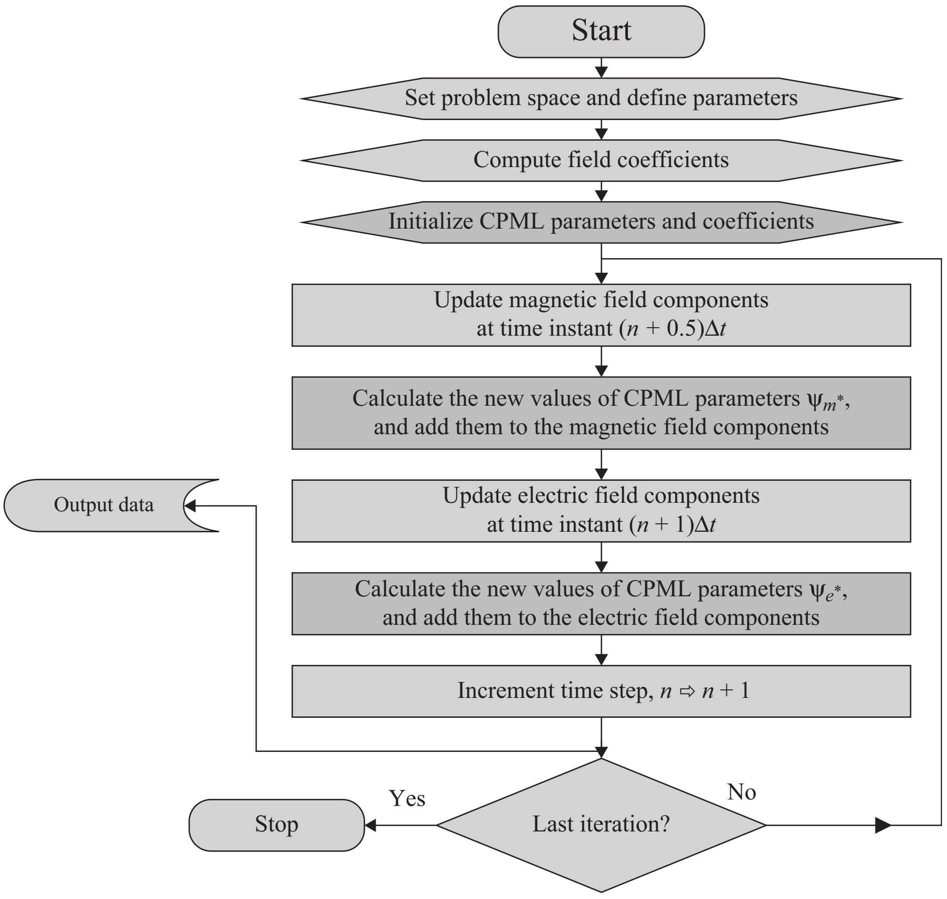
Start
Set problem space and define parameters
Compute field coefficients
Initialize CPML parameters and coefficients
Update magnetic field components at time instant $(n + 0.5)\Delta t$
Calculate the new values of CPML parameters $\psi_{m}*$, and add them to the magnetic field components
Output data
Update electric field components at time instant $(n + 1)\Delta t$
Calculate the new values of CPML parameters $\psi_{e}*$, and add them to the electric field components
Increment time step, $n \Rightarrow n + 1$
Yes
No
Stop
Last iteration?

**Figure 8.1** The FDTD flowchart including the steps of CPML algorithm.

CPML regions for which they were defined. The next step is updating the electric field components in the problem space using the regular updating equation derived for the host medium. Then the CPML terms ($\psi_{exy}$, $\psi_{exz}$, $\psi_{eyx}$, $\psi_{eyz}$, $\psi_{ezx}$, $\psi_{ezy}$) are calculated using their previous time step values and the new magnetic field values and are added to their respective electric field components only at the CPML regions for which they were defined. The details of the steps of this algorithm are discussed further in subsequent sections of this chapter.

## 8.2.1 Updating equations for CPML

The general form of the updating equation for a non-CPML lossy medium is given in Chapter 1 and is repeated here for the $E_x$ component for convenience as

$$\begin{aligned} E_x^{n+1}(i,j,k) = C_{exe}(i,j,k) &\times E_x^n(i,j,k) \\ &+ C_{exhz}(i,j,k) \times \left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right) \\ &+ C_{exhy}(i,j,k) \times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)\right). \end{aligned} \tag{8.24}$$

Then the updating equation for the CPML region takes the form

$$\begin{aligned} E_x^{n+1}(i,j,k) = C_{exe}(i,j,k) \times E_x^n(i,j,k) &\\ + \left(1/\kappa_{ey}(i,j,k)\right) \times C_{exhz}(i,j,k) &\times \left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right) \\ + \left(1/\kappa_{ez}(i,j,k)\right) \times C_{exhy}(i,j,k) &\times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)\right) \\ + \left(\Delta y C_{exhz}(i,j,k)\right) \times \psi_{exy}^{n+\frac{1}{2}}(i,j,k) &+ \left(\Delta z C_{exhy}(i,j,k)\right) \times \psi_{exz}^{n+\frac{1}{2}}(i,j,k). \end{aligned} \tag{8.25}$$

Notice that the negative sign in front of the $\psi_{exz}^{n+\frac{1}{2}}(i,j,k)$ term in (8.16) is embedded in the coefficient $C_{exhy}(i,j,k)$ as can be followed from (1.26). Here two new coefficients can be defined such that

$$C_{\psi exy}(i,j,k) \Leftarrow \Delta y C_{exhz}(i,j,k). \tag{8.26a}$$

$$C_{\psi exz}(i,j,k) \Leftarrow \Delta z C_{exhy}(i,j,k). \tag{8.26b}$$

Then (8.25) reduces to

$$\begin{aligned} E_x^{n+1}(i,j,k) = C_{exe}(i,j,k) \times E_x^n(i,j,k) &\\ + \left(1/\kappa_{ey}(i,j,k)\right) \times C_{exhz}(i,j,k) &\times \left(H_z^{n+\frac{1}{2}}(i,j,k) - H_z^{n+\frac{1}{2}}(i,j-1,k)\right) \\ + \left(1/\kappa_{ez}(i,j,k)\right) \times C_{exhy}(i,j,k) &\times \left(H_y^{n+\frac{1}{2}}(i,j,k) - H_y^{n+\frac{1}{2}}(i,j,k-1)\right) \\ + C_{\psi exy}(i,j,k) \times \psi_{exy}^{n+\frac{1}{2}}(i,j,k) &+ C_{\psi exz}(i,j,k) \times \psi_{exz}^{n+\frac{1}{2}}(i,j,k). \end{aligned} \tag{8.27}$$

Furthermore, the terms $(1/\kappa_{ey}(i, j, k))$ and $(1/\kappa_{ez}(i, j, k))$ can be embedded into $C_{exhz}(i, j, k)$ and $C_{exhy}(i, j, k)$, respectively, such that

$$C_{exhz}(i, j, k) \Leftarrow \left(1/\kappa_{ey}(i, j, k)\right) \times C_{exhz}(i, j, k), \tag{8.28a}$$

$$C_{exhy}(i, j, k) \Leftarrow (1/\kappa_{ez}(i, j, k)) \times C_{exhy}(i, j, k). \tag{8.28b}$$

These modifications are applied to the coefficients only at the positions overlapping with the respective CPML regions and further reduce (8.27) to

$$\begin{aligned} E_x^{n+1}(i, j, k) = {} & C_{exe}(i, j, k) \times E_x^n(i, j, k) \\ & + C_{exhz}(i, j, k) \times \left(H_z^{n+\frac{1}{2}}(i, j, k) - H_z^{n+\frac{1}{2}}(i, j-1, k)\right) \\ & + C_{exhy}(i, j, k) \times \left(H_y^{n+\frac{1}{2}}(i, j, k) - H_y^{n+\frac{1}{2}}(i, j, k-1)\right) \\ & + C_{\psi exy}(i, j, k) \times \psi_{exy}^{n+\frac{1}{2}}(i, j, k) + C_{\psi exz}(i, j, k) \times \psi_{exz}^{n+\frac{1}{2}}(i, j, k). \end{aligned} \tag{8.29}$$

With the given form of updating equation and coefficients, $E_x^{n+1}(i,j,k)$ is updated using the first three terms on the right side in the entire domain as usual. Then $\psi_{exy}^{n+\frac{1}{2}}(i,j,k)$ and $\psi_{exz}^{n+\frac{1}{2}}(i,j,k)$ are calculated (e.g., using (8.21) for $\psi_{exy}^{n+\frac{1}{2}}(i,j,k)$), and the last two terms of (8.29) are added to $E_x^{n+1}(i,j,k)$ at their respective CPML regions. It should be mentioned that the same procedure applies for updating the other electric and magnetic field components as well by using their respective updating equations and CPML parameters.

## 8.2.2 Addition of auxiliary CPML terms at respective regions

While discussing the PML we showed in Figure 7.4 that certain PML conductivity parameters take nonzero value at certain respective regions. In the CPML case there are three types of parameters, and the regions for which they are defined are shown in Figure 8.2, which is similar to Figure 7.4. In the CPML case the parameters $\sigma_{pei}$, $\sigma_{pmi}$, $\alpha_{ei}$, and $\alpha_{mi}$ are nonzero, whereas the parameters $\kappa_{ei}$ and $\kappa_{mi}$ take values larger than one at their respective regions. These regions are named *xn*, *xp*, *yn*, *yp*, *zn*, and *zp*. The CPML auxiliary terms $\psi$ are associated with these CPML parameters. Therefore, each term $\psi$ is defined at the region where its associated parameters are defined. For instance, $\psi_{exy}$ is associated with $\sigma_{pey}$, $\alpha_{ey}$, and $\kappa_{ey}$ as can be observed in (8.21). Then the term $\psi_{exy}$ is defined for the *yn* and *yp* regions. Similarly, the term $\psi_{exz}$ is defined for the *zn* and *zp* regions. One should notice in (8.27) that both the $\psi_{exy}$ and $\psi_{exz}$ terms are added to $E_x$ as the requirement of the CPML algorithm; however, $\psi_{exy}$ and $\psi_{exz}$ are defined in different regions. This implies that the terms $\psi_{exy}$ and $\psi_{exz}$ should be added to $E_x$ only in their respective regions. Therefore, one should take care while determining the regions where the CPML terms are added to the field components. For instance, $\psi_{exy}$ should be added to $E_x$ in the *yn* and *yp* regions, whereas $\psi_{exz}$ should be added to $E_x$ in the *zn* and *zp* regions. A detailed list of the CPML updating equations for all six electric and magnetic field components and the regions at which these equations are applied is given in Appendix B.

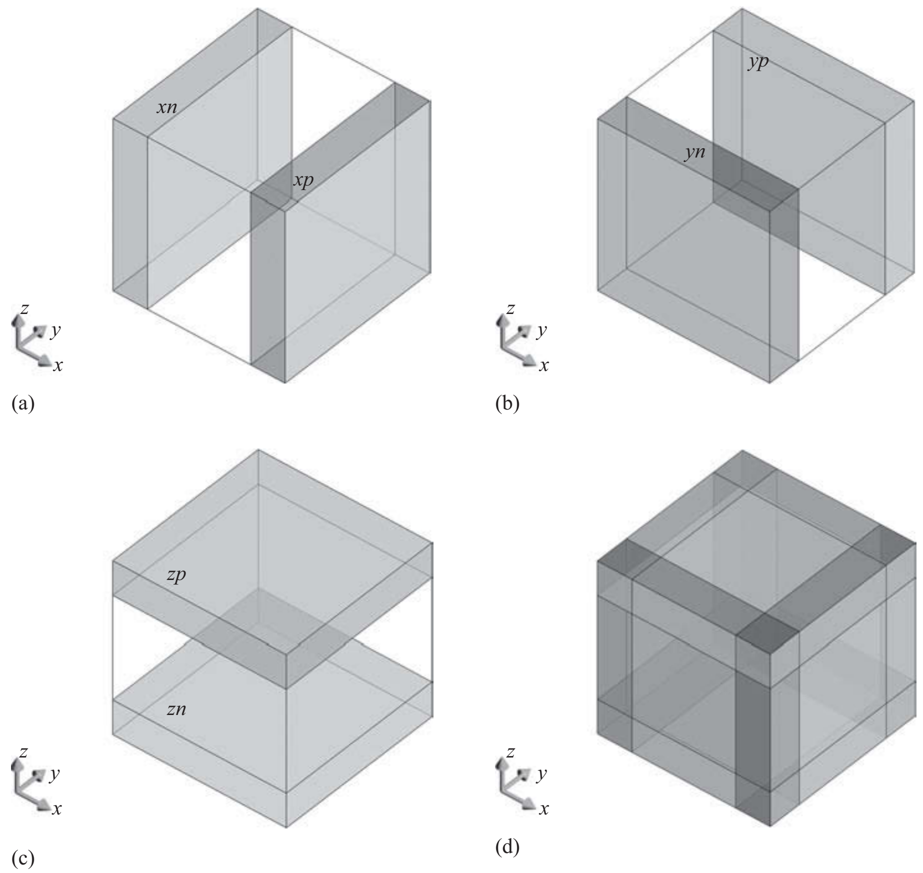

**Figure 8.2** Regions where CPML parameters are defined: (a) $\sigma_{pex}$, $\sigma_{pmx}$, $\alpha_{ex}$, $\alpha_{mx}$, $\kappa_{ex}$, and $\kappa_{mx}$; (b) $\sigma_{pey}$, $\sigma_{pmy}$, $\alpha_{ey}$, $\alpha_{my}$, $\kappa_{ey}$, and $\kappa_{my}$; (c) $\sigma_{pez}$, $\sigma_{pmz}$, $\alpha_{ez}$, $\alpha_{mz}$, $\kappa_{ez}$, and $\kappa_{mz}$; and (d) overlapping CPML regions.

## 8.3 CPML parameter distribution

As described in the previous chapter, the PML conductivities are scaled along the PML region starting from a zero value at the inner domain–PML interface and increasing to a maximum value at the outer boundary. In the CPML case there are two additional types of parameter scaling profiles, which are different from the conductivity scaling profiles.

The maximum conductivity of the conductivity profile is computed in [23] using $\sigma_{\max} = \sigma_{factor} \times \sigma_{opt}$, where

$$\sigma_{opt} = \frac{n_{pml} + 1}{150\pi\sqrt{\varepsilon_r}\Delta i}. \tag{8.30}$$

Here $n_{pml}$ is the order of the polynomial scaling, and $\varepsilon_r$ is the relative permittivity of the background material. Then the CPML conductivity $\sigma_{pei}$ can be calculated by

$$\sigma_{pei}(\rho) = \sigma_{\max}\left(\frac{\rho}{\delta}\right)^{n_{pml}}, \tag{8.31}$$

where $\rho$ is the distance from the computational domain–PML interface to the position of the field component and $\delta$ is the thickness of the CPML layer. Similarly, $\sigma_{pmi}$ can be calculated by

$$\sigma_{pmi}(\rho) = \frac{\mu_0}{\varepsilon_0}\sigma_{\max}\left(\frac{\rho}{\delta}\right)^{n_{pml}}, \tag{8.32}$$

which satisfies the reflectionless condition (8.7).

The value of $\kappa_{ei}$ is unity at the inner domain–CPML interface, and it increases to a maximum value $\kappa_{\max}$ such that

$$\kappa_{ei}(\rho) = 1 + (\kappa_{\max} - 1)\left(\frac{\rho}{\delta}\right)^{n_{pml}}, \tag{8.33}$$

whereas $\kappa_{mi}$ is given by

$$\kappa_{mi}(\rho) = 1 + (\kappa_{\max} - 1)\left(\frac{\rho}{\delta}\right)^{n_{pml}}. \tag{8.34}$$

In these equations, $\rho$ indicates the distances of the respective field components from the inner domain–CPML interface. It should be noted that the values that $\rho$ can take in (8.33) and (8.34) are different since the electric and magnetic field components are located at different positions.

The parameter $\alpha_{ei}$ takes a maximum value $\alpha_{\max}$ at the inner domain–CPML interface and is linearly scaled to a minimum value, $\alpha_{\min}$, at the outer boundary such that

$$\alpha_{ei}(\rho) = \alpha_{\min} + (\alpha_{\max} - \alpha_{\min})\left(1 - \frac{\rho}{\delta}\right). \tag{8.35}$$

Similarly, $\alpha_{mi}$ is calculated as

$$\alpha_{mi}(\rho) = \frac{\mu_0}{\varepsilon_0}\left(\alpha_{\min} + (\alpha_{\max} - \alpha_{\min})\left(1 - \frac{\rho}{\delta}\right)\right). \tag{8.36}$$

This scaling of the CPML parameter profile is chosen as before specifically to reduce the reflection error of evanescent modes; $\alpha$ must be nonzero at the front boundary interface. However, for the CFS-PML to absorb purely propagating modes at low frequency, $\alpha$ should actually decrease to zero away from the boundary interface [23].

The choice of the parameters $\sigma_{factor}$, $\kappa_{\max}$, $\alpha_{\max}$, $\alpha_{\min}$, and $n_{pml}$ and the thickness of the CPML layer in terms of number of cells determine the performance of the CPML. In [23] parametric studies have been performed to evaluate the effects of these parameters. Usually, $\sigma_{factor}$ can be taken in the range 0.7–1.5, $\kappa_{\max}$ in the range 5–11, $\alpha_{\max}$ in the range 0–0.05, thickness of CPML as 8 cells, and $n_{pml}$ as 2, 3, or 4.

## 8.4 MATLAB® implementation of CPML in the three-dimensional FDTD method

In this section, we demonstrate the implementation of the CPML in the three-dimensional FDTD MATLAB code.

## 8.4.1 Definition of CPML

In the previous chapters, the only type of boundary available in our three-dimensional program was perfect electric conductor (PEC). The type of the boundaries were defined as PEC by assigning "***pec***" to the parameter **boundary.type**, and the air gap between the objects and the outer boundary was assigned to **boundary.air_buffer_number_of_cells** in the ***define_problem_space_parameters*** subroutine. To define the boundaries as CPML, we can update ***define_problem_space_parameters*** as shown in Listing 8.1. The type of a boundary is determined as CMPL by assigning "***cpml***" to **boundary.type**. It should be noted that the PEC boundaries can be used together with CPML boundaries; that is, some sides can be assigned PEC while the others are CPML. In Listing 8.1, the *zn* and *zp* sides are PEC while other boundaries are CPML.

**Listing 8.1** define_problem_space_parameters

```
% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 4;
boundary.cpml_number_of_cells_xn = 5;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 4;
boundary.cpml_number_of_cells_xp = 5;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 0;
boundary.cpml_number_of_cells_yn = -5;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 0;
boundary.cpml_number_of_cells_yp = -5;

boundary.type_zn = 'pec';
boundary.air_buffer_number_of_cells_zn = 0;
boundary.cpml_number_of_cells_zn = 5;

boundary.type_zp = 'pec';
boundary.air_buffer_number_of_cells_zp = 6;
boundary.cpml_number_of_cells_zp = 5;

boundary.cpml_order = 3;
boundary.cpml_sigma_factor = 1.5;
boundary.cpml_kappa_max = 7;
boundary.cpml_alpha_min = 0;
boundary.cpml_alpha_max = 0.05;
```

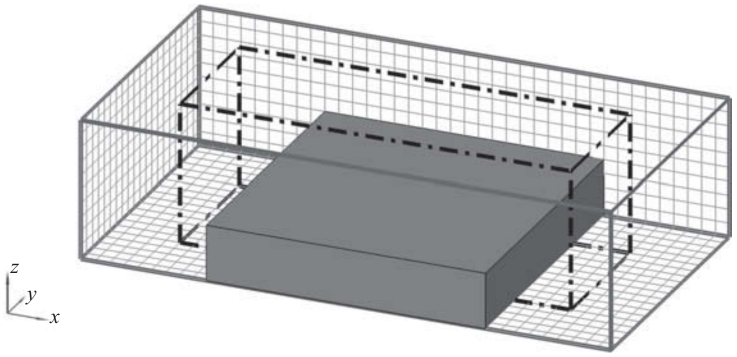

**Figure 8.3** A problem space with 20 × 20 × 4 cells brick.

Another parameter required for a boundary defined as CPML is the thickness of the CPML layer in terms of number of cells. A new parameter **cpml_number_of_cells** is defined as a subfield of the parameter **boundary**, and the thicknesses of the CPML regions are assigned to this new parameter. One can notice that the thickness of CPML is 5 cells in the *xn* and *xp* regions and $-5$ cells in the *yn* and *yp* regions. As discussed before, objects in an FDTD problem space can be defined as penetrating into CPML; thus, these objects resemble structures extending to infinity. As a convention, if **cpml_number_of_cells** is defined as a negative number, it is assumed that the CPML layers are extending to inside of the domain rather than extending outside from the end of the air buffer region. For instance, if **air_buffer_number_of_cells** is zero, and the **cpml_number_of_cells** is a negative number at the *yn* region, then the objects on the *yn* side will be penetrating to the CPML region. Figure 8.3 illustrates a problem space including a 20 × 20 × 4 cells brick and having its boundaries defined as shown in Listing 8.1. The solid thick line shows the boundaries of the problem space. The dash-dot thick line shows the inner boundaries of the CPML regions. The brick penetrates into the CPML in *yn* and *yp* regions. The *zn* and *zp* boundaries are PEC. The other parameters described in Section 8.3 are defined in ***define_problem_space_parameters*** as well. The order of the polynomial distribution $n_{pml}$ is defined as **boundary.cpml_order** as shown in Listing 8.1. The $\sigma_{factor}$ is defined as **boundary.cpml_sigma_factor**, $\kappa_{\max}$ is defined as **boundary.cpml_kappa_max**, $\alpha_{\max}$ is defined as **boundary.cpml_alpha_max**, and similarly, $\alpha_{\min}$ is defined as **boundary.cpml_alpha_min**.

## 8.4.2 Initialization of CPML

To employ the CPML algorithm in the time-marching loop, several coefficient and auxiliary parameter arrays need to be defined and initialized for the respective CPML regions before the time-marching loop starts. Before defining these arrays, the size of the FDTD problem space should be determined by taking the thicknesses of the CPML regions into account. The size of the problem space and the number of cells in the domain are determined in the subroutine ***calculate_domain_size***, which was called in ***initialize_fdtd_material_grid***. The implementation of ***calculate_domain_size*** is given in Listing 3.5, where the boundaries are assumed to be PEC. As discussed in the previous section, if the thickness of the CPML is negative on a side, the CPML region extends into the problem space on this side; therefore, the position of the outer boundary of the problem space remains the same, and the calculation given for determining the position of the outer boundary in Listing 3.5 remains the same. However, if the thickness of the CPML is positive on a side, the CPML adds to the problem space on that side, and the calculation of the position of the outer boundary needs to be updated accordingly. The necessary modifications are implemented in ***calculate_domain_size***, and the modified section of the code is given in Listing 8.2. It can be noticed in the code that, for the sides where the boundary type is CPML, the problem space boundary is extended outward by the thickness of the CPML.

**Listing 8.2** calculate_domain_size

```
% Determine the problem space boundaries including air buffers
fdtd_domain.min_x = fdtd_domain.min_x ...
    - dx * boundary.air_buffer_number_of_cells_xn;
fdtd_domain.min_y = fdtd_domain.min_y ...
    - dy * boundary.air_buffer_number_of_cells_yn;
fdtd_domain.min_z = fdtd_domain.min_z ...
    - dz * boundary.air_buffer_number_of_cells_zn;
fdtd_domain.max_x = fdtd_domain.max_x ...
    + dx * boundary.air_buffer_number_of_cells_xp;
fdtd_domain.max_y = fdtd_domain.max_y ...
    + dy * boundary.air_buffer_number_of_cells_yp;
fdtd_domain.max_z = fdtd_domain.max_z ...
    + dz * boundary.air_buffer_number_of_cells_zp;

% Determine the problem space boundaries including cpml layers
if strcmp(boundary.type_xn, 'cpml') && ...
        (boundary.cpml_number_of_cells_xn >0)
    fdtd_domain.min_x = fdtd_domain.min_x ...
        - dx * boundary.cpml_number_of_cells_xn;
end
if strcmp(boundary.type_xp, 'cpml') && ...
        (boundary.cpml_number_of_cells_xp >0)
    fdtd_domain.max_x = fdtd_domain.max_x ...
        + dx * boundary.cpml_number_of_cells_xp;
end
if strcmp(boundary.type_yn, 'cpml') && ...
        (boundary.cpml_number_of_cells_yn >0)
    fdtd_domain.min_y = fdtd_domain.min_y ...
        - dy * boundary.cpml_number_of_cells_yn;
end
if strcmp(boundary.type_yp, 'cpml') && ...
        (boundary.cpml_number_of_cells_yp >0)
    fdtd_domain.max_y = fdtd_domain.max_y ...
        + dy * boundary.cpml_number_of_cells_yp;
end
if strcmp(boundary.type_zn, 'cpml') && ...
        (boundary.cpml_number_of_cells_zn >0)
    fdtd_domain.min_z = fdtd_domain.min_z ...
        - dz * boundary.cpml_number_of_cells_zn;
end
if strcmp(boundary.type_zp, 'cpml') && ...
        (boundary.cpml_number_of_cells_zp >0)
    fdtd_domain.max_z = fdtd_domain.max_z ...
        + dz * boundary.cpml_number_of_cells_zp;
end
```

The initialization process of the CPML continues in ***initialize_boundary_conditions***, which is a subroutine called in the main program ***fdtd_solve*** as shown in Listing 3.1. So far we have not implemented any code in ***initialize_boundary_conditions***, since the only type of boundary we have considered is PEC; which does not require any special treatment; the tangential electric field components on the outer faces of the problem space are not updated, and their values are left zero during the time-marching loop which naturally simulates the PEC boundaries. However, the CPML requires special treatment, and the initialization of the CPML is coded in ***initialize_boundary_conditions*** accordingly as shown in Listing 8.3.

**Listing 8.3** initialize_boundary_conditions

```
% initialize boundary parameters

% define logical parameters for the conditions that  will be used often
is_cpml_xn = false; is_cpml_xp = false; is_cpml_yn = false;
is_cpml_yp = false; is_cpml_zn = false; is_cpml_zp = false;
is_any_side_cpml = false;
if strcmp(boundary.type_xn, 'cpml')
    is_cpml_xn = true;
    n_cpml_xn = abs(boundary.cpml_number_of_cells_xn);
end
if strcmp(boundary.type_xp, 'cpml')
    is_cpml_xp = true;
    n_cpml_xp = abs(boundary.cpml_number_of_cells_xp);
end
if strcmp(boundary.type_yn, 'cpml')
    is_cpml_yn = true;
    n_cpml_yn = abs(boundary.cpml_number_of_cells_yn);
end
if strcmp(boundary.type_yp, 'cpml')
    is_cpml_yp = true;
    n_cpml_yp = abs(boundary.cpml_number_of_cells_yp);
end
if strcmp(boundary.type_zn, 'cpml')
    is_cpml_zn = true;
    n_cpml_zn = abs(boundary.cpml_number_of_cells_zn);
end
if strcmp(boundary.type_zp, 'cpml')
    is_cpml_zp = true;
    n_cpml_zp = abs(boundary.cpml_number_of_cells_zp);
end

if (is_cpml_xn || is_cpml_xp || is_cpml_yn ...
        || is_cpml_yp || is_cpml_zn || is_cpml_zp)
    is_any_side_cpml = true;
end

% Call CPML initialization routine if any side is CPML
if is_any_side_cpml
    initialize_CPML_ABC;
end
```

In this subroutine several frequently used parameters are defined for convenience. For instance, the parameter **is_cpml_xn** is a logical parameter that indicates whether the *xn* boundary is CPML, and **n_cpml_xn** stores the thickness of the CPML for the *xn* region. Similar parameters are defined for other sides as well. Then another subroutine, ***initialize_CPML_ABC***, is called to continue the CPML specific initialization process.

Listing 8.4 shows the contents of ***initialize_CPML_ABC*** for the *xn* and *xp* regions. The code listing for the other sides follows the same procedure. Let us consider the *xp* region, for example. In the code, distribution of the CPML parameters $\sigma_{pex}$, $\sigma_{pmx}$, $\kappa_{ex}$, $\kappa_{mx}$, $\alpha_{ex}$, and $\alpha_{mx}$ along the CPML thickness are calculated as one-dimensional arrays first, with the respective names **sigma_pex_xp**, **sigma_pmx_xp**, **kappa_ex_xp**, **kappa_mx_xp**, **alpha_ex_xp**, and

**Listing 8.4** initialize_CPML_ABC

```
% Initialize CPML boundary condition

p_order = boundary.cpml_order; % order of the polynomial distribution
sigma_ratio = boundary.cpml_sigma_factor;
kappa_max = boundary.cpml_kappa_max;
alpha_min = boundary.cpml_alpha_min;
alpha_max = boundary.cpml_alpha_max;

% Initialize cpml for xn region
if is_cpml_xn

    % define one-dimensional temporary cpml parameter arrays
    sigma_max = sigma_ratio  * (p_order+1)/(150*pi*dx);
    ncells = n_cpml_xn;
    rho_e = ([ncells:-1:1]-0.75)/ncells;
    rho_m = ([ncells:-1:1]-0.25)/ncells;
    sigma_pex_xn = sigma_max * rho_e.^p_order;
    sigma_pmx_xn = sigma_max * rho_m.^p_order;
    sigma_pmx_xn = (mu_0/eps_0) * sigma_pmx_xn;
    kappa_ex_xn = 1 + (kappa_max - 1) * rho_e.^p_order;
    kappa_mx_xn = 1 + (kappa_max - 1) * rho_m.^p_order;
    alpha_ex_xn = alpha_min + (alpha_max - alpha_min) * (1-rho_e);
    alpha_mx_xn = alpha_min + (alpha_max - alpha_min) * (1-rho_m);
    alpha_mx_xn = (mu_0/eps_0) * alpha_mx_xn;

    % define one-dimensional cpml parameter arrays
    cpml_b_ex_xn = exp((-dt/eps_0) ...
        *((sigma_pex_xn./kappa_ex_xn)+ alpha_ex_xn));
    cpml_a_ex_xn = (1/dx)*(cpml_b_ex_xn-1.0).* sigma_pex_xn ...
        ./(kappa_ex_xn.*(sigma_pex_xn+kappa_ex_xn.*alpha_ex_xn));
    cpml_b_mx_xn = exp((-dt/mu_0) ...
        *((sigma_pmx_xn./kappa_mx_xn)+ alpha_mx_xn));
    cpml_a_mx_xn = (1/dx)*(cpml_b_mx_xn-1.0) .* sigma_pmx_xn ...
        ./(kappa_mx_xn.*(sigma_pmx_xn+kappa_mx_xn.*alpha_mx_xn));

    % Create and initialize 2D cpml convolution parameters
    Psi_eyx_xn = zeros(ncells,ny,nzp1);
    Psi_ezx_xn = zeros(ncells,nyp1,nz);
    Psi_hyx_xn = zeros(ncells,nyp1,nz);
    Psi_hzx_xn = zeros(ncells,ny,nzp1);

    % Create and initialize 2D cpml convolution coefficients
    % Notice that Ey(1,:,:) and Ez(1,:,:) are not updated by cmpl
    CPsi_eyx_xn = Ceyhz(2:ncells+1,:,:)*dx;
    CPsi_ezx_xn = Cezhy(2:ncells+1,:,:)*dx;
    CPsi_hyx_xn = Chyez(1:ncells,:,:)*dx;
    CPsi_hzx_xn = Chzey(1:ncells,:,:)*dx;

    % Adjust FDTD coefficients in the CPML region
    % Notice that Ey(1,:,:) and Ez(1,:,:) are not updated by cmpl
    for i = 1: ncells
        Ceyhz(i+1,:,:) = Ceyhz(i+1,:,:)/kappa_ex_xn(i);
        Cezhy(i+1,:,:) = Cezhy(i+1,:,:)/kappa_ex_xn(i);
        Chyez(i,:,:) = Chyez(i,:,:)/kappa_mx_xn(i);
        Chzey(i,:,:) = Chzey(i,:,:)/kappa_mx_xn(i);
    end

    % Delete temporary arrays. These arrays will not be used any more.
    clear sigma_pex_xn sigma_pmx_xn;
    clear kappa_ex_xn kappa_mx_xn;
    clear alpha_ex_xn alpha_mx_xn;
end

% Initialize cpml for xp region
if is_cpml_xp

    % define one-dimensional temporary cpml parameter arrays
    sigma_max = sigma_ratio  * (p_order+1)/(150*pi*dx);
    ncells = n_cpml_xp;
    rho_e = ([1:ncells]-0.25)/ncells;
    rho_m = ([1:ncells]-0.75)/ncells;
    sigma_pex_xp = sigma_max * rho_e.^p_order;
    sigma_pmx_xp = sigma_max * rho_m.^p_order;
    sigma_pmx_xp = (mu_0/eps_0) * sigma_pmx_xp;
    kappa_ex_xp = 1 + (kappa_max - 1) * rho_e.^p_order;
    kappa_mx_xp = 1 + (kappa_max - 1) * rho_m.^p_order;
    alpha_ex_xp = alpha_min + (alpha_max - alpha_min) * (1-rho_e);
    alpha_mx_xp = alpha_min + (alpha_max - alpha_min) * (1-rho_m);
    alpha_mx_xp = (mu_0/eps_0) * alpha_mx_xp;

    % define one-dimensional cpml parameter arrays
    cpml_b_ex_xp = exp((-dt/eps_0) ...
        *((sigma_pex_xp./kappa_ex_xp)+ alpha_ex_xp));
    cpml_a_ex_xp = (1/dx)*(cpml_b_ex_xp-1.0).* sigma_pex_xp ...
        ./(kappa_ex_xp.*(sigma_pex_xp+kappa_ex_xp.*alpha_ex_xp));
    cpml_b_mx_xp = exp((-dt/mu_0) ...
        *((sigma_pmx_xp./kappa_mx_xp)+ alpha_mx_xp));
    cpml_a_mx_xp = (1/dx)*(cpml_b_mx_xp-1.0) .* sigma_pmx_xp ...
        ./(kappa_mx_xp.*(sigma_pmx_xp+kappa_mx_xp.*alpha_mx_xp));

    % Create and initialize 2D cpml convolution parameters
    Psi_eyx_xp = zeros(ncells,ny,nzp1);
    Psi_ezx_xp = zeros(ncells,nyp1,nz);
    Psi_hyx_xp = zeros(ncells,nyp1,nz);
    Psi_hzx_xp = zeros(ncells,ny,nzp1);

    % Create and initialize 2D cpml convolution coefficients
    % Notice that Ey(nxp1,:,:) and Ez(nxp1,:,:) are not updated by cmpl
    CPsi_eyx_xp = Ceyhz(nxp1-ncells:nx,:,:)*dx;
    CPsi_ezx_xp = Cezhy(nxp1-ncells:nx,:,:)*dx;
    CPsi_hyx_xp = Chyez(nxp1-ncells:nx,:,:)*dx;
    CPsi_hzx_xp = Chzey(nxp1-ncells:nx,:,:)*dx;

    % Adjust FDTD coefficients in the CPML region
    % Notice that Ey(nxp1,:,:) and Ez(nxp1,:,:) are not updated by cmpl
    for i = 1: ncells
        Ceyhz(nx-ncells+i,:,:) = Ceyhz(nx-ncells+i,:,:)/kappa_ex_xp(i);
        Cezhy(nx-ncells+i,:,:) = Cezhy(nx-ncells+i,:,:)/kappa_ex_xp(i);
        Chyez(nx-ncells+i,:,:) = Chyez(nx-ncells+i,:,:)/kappa_mx_xp(i);
        Chzey(nx-ncells+i,:,:) = Chzey(nx-ncells+i,:,:)/kappa_mx_xp(i);
    end

    % Delete temporary arrays. These arrays will not be used any more.
    clear sigma_pex_xp sigma_pmx_xp;
    clear kappa_ex_xp kappa_mx_xp;
    clear alpha_ex_xp alpha_mx_xp;
end
```

**alpha_mx_xp**, using the equations given in Section 8.3. The parameters $\sigma_{pex}$, $\kappa_{ex}$, and $\alpha_{ex}$ are used to update the $E_y$ and $E_z$ field components, and the distances of these field components from the interior interface of the CPML are used in (8.31), (8.33), and (8.35) as $\rho_e$. The positions of the field components and the respective distances of the field components from the CPML interface are illustrated in Figure 8.4. One can see that CPML interface is assumed to be offset from the first CPML cell by a quarter cell size. This is to ensure that the same number of electric and magnetic field components is updated by the CPML algorithm. Similarly, Figure 8.5 illustrates the magnetic field components of $H_y$ and $H_z$ being updated using the CPML in the *xp* region. The distances of these field components from the CPML interface are denoted as $\rho_h$ and are used in (8.32), (8.34), and (8.36) to calculate $\sigma_{pmx}$, $\kappa_{mx}$, and $\alpha_{mx}$.

Then Listing 8.4 continues with calculation of $a_{ex}$, $b_{ex}$, $a_{mx}$, and $b_{mx}$ as **cpml_a_ex_xp**, **cpml_b_ex_xp**, **cpml_a_mx_xp**, and **cpml_b_mx_xp** using the equations in Appendix B. Then the CPML auxiliary parameters $\psi_{eyx}$, $\psi_{ezx}$, $\psi_{hyx}$, and $\psi_{hzx}$ are defined as two-dimensional arrays with the names **Psi_eyx_xp**, **Psi_ezx_xp**, **Psi_hyx_xp**, and **Psi_hzx_xp** and are initialized with zero value. The coefficients $C_{\psi eyx}$, $C_{\psi ezx}$, $C_{\psi hyx}$, and $C_{\psi hzx}$ are calculated and stored as two-dimensional arrays with the respective names **CPsi_eyx_xp**, **CPsi_ezx_xp**, **CPsi_hyx_xp**, and **CPsi_hzx_xp**. Then the FDTD updating coefficients **Ceyhz**, **Cezhy**, **Chyez**, and **Chzey** are scaled with the respective $\kappa$ values in the *xp* region. Finally, the parameters that will not be used anymore during the FDTD calculations are cleared from MATLAB workspace.

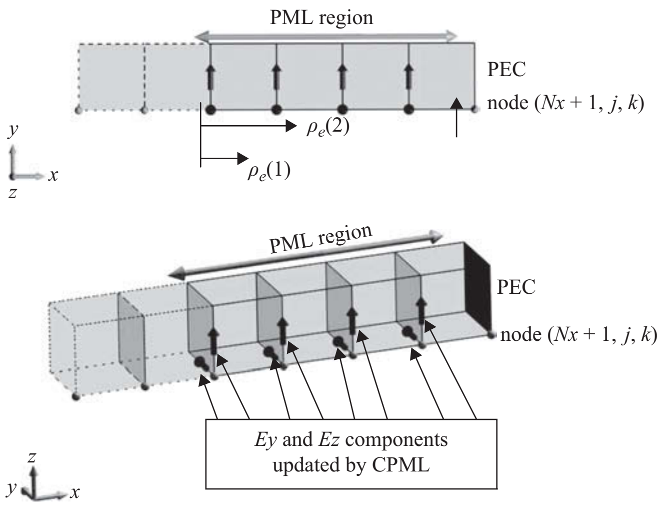

**Figure 8.4** Positions of the electric field components updated by CPML in the $xp$ region.

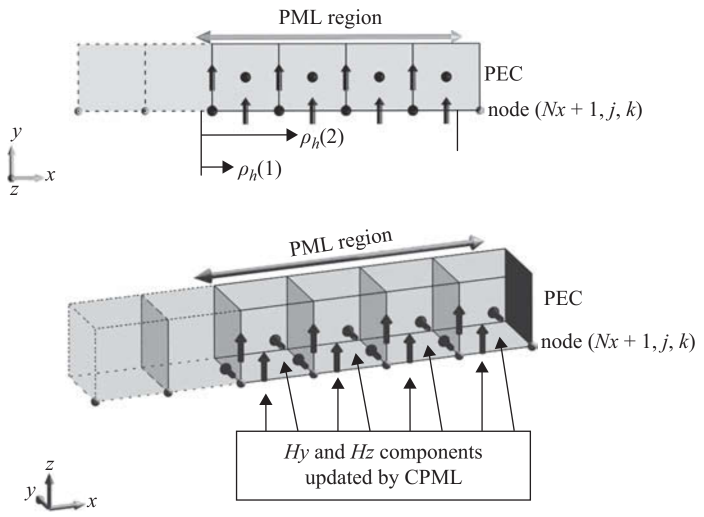

**Figure 8.5** Positions of the magnetic field components updated by CPML in the $xp$ region.

## 8.4.3 Application of CPML in the FDTD time-marching loop

The necessary CPML arrays and parameters are initialized in ***initialize_CPML_ABC***, and they are ready to use for applying the CPML in the FDTD time-marching loop. Two new subroutines are added to the subroutine ***run_fdtd_time_marching_loop*** as shown in Listing 8.5. The first one is ***update_magnetic_field_CPML_ABC***, and it follows ***update_magnetic_fields***. The second one is ***update_electric_field_CPML_ABC***, and it follows ***update_electric_fields***.

**Listing 8.5** run_fdtd_time_marching_loop

```
disp (['Starting the time marching loop']);
disp(['Total number of time steps : ' ...
    num2str(number_of_time_steps)]);

start_time = cputime;
current_time = 0;

for time_step = 1:number_of_time_steps
    update_magnetic_fields;
    update_magnetic_field_CPML_ABC;
    capture_sampled_magnetic_fields;
    capture_sampled_currents;
    update_electric_fields;
    update_electric_field_CPML_ABC;
    update_voltage_sources;
    update_current_sources;
    update_inductors;
    update_diodes;
    capture_sampled_electric_fields;
    capture_sampled_voltages;
    display_sampled_parameters;
end

end_time = cputime;
total_time_in_minutes = (end_time - start_time)/60;
disp(['Total simulation time is ' ...
    num2str(total_time_in_minutes) ' minutes.']);
```

The implementation of ***update_magnetic_field_CPML_ABC*** is given in Listing 8.6. The magnetic field components are updated for the current time step using the regular updating equations in ***update_magnetic_fields***, and then in ***update_magnetic_field_CPML_ABC***, first the auxiliary $\psi$ parameters are calculated using the current values of the electric field components. These terms are then added to the appropriate magnetic field components in their respective CPML regions.

Similarly, the implementation of ***update_electric_field_CPML_ABC*** is given in Listing 8.7. The electric field components are updated for the current time step using the regular updating equations in ***update_electric_fields***, and then in ***update_electric_field_CPML_ABC***, first the auxiliary $\psi$ parameters are calculated using the current values of the magnetic field components. These terms are then added to the appropriate electric field components in their respective CPML regions.

One can notice in the code listings that the arrays representing $\psi_e$ and $\psi_h$ are two-dimensional, whereas the arrays representing $a_{ei}$, $a_{mi}$, $b_{ei}$, and $b_{mi}$ are one-dimensional. Therefore, $\psi_e$ and $\psi_h$ are updated in a "for" loop using the one-dimensional arrays; if they were defined as two-dimensional arrays, the "for" loop would be avoided, which would speed up the calculation with the trade-off of increased memory. In the two-dimensional array case, the coefficients representing $C_{\psi e}$ and $C_{\psi m}$ can be distributed over $a_{ei}$, $a_{mi}$, $b_{ei}$, and $b_{mi}$ in the initialization process ***initialize_CPML_ABC***, which further reduces the amount of calculations during the CPML updates.

**Listing 8.6** update_magnetic_field_CPML_ABC

```
% apply CPML to magnetic field components
if is_cpml_xn
    for i = 1: n_cpml_xn
        Psi_hyx_xn(i,:,:) = cpml_b_mx_xn(i) * Psi_hyx_xn(i,:,:) ...
            + cpml_a_mx_xn(i)*(Ez(i+1,:,:)-Ez(i,:,:));
        Psi_hzx_xn(i,:,:) = cpml_b_mx_xn(i) * Psi_hzx_xn(i,:,:) ...
            + cpml_a_mx_xn(i)*(Ey(i+1,:,:)-Ey(i,:,:));
    end
        Hy(1:n_cpml_xn,:,:) = Hy(1:n_cpml_xn,:,:) ...
            + CPsi_hyx_xn(:,:,:) .* Psi_hyx_xn(:,:,:);
        Hz(1:n_cpml_xn,:,:) = Hz(1:n_cpml_xn,:,:) ...
            + CPsi_hzx_xn(:,:,:) .* Psi_hzx_xn(:,:,:);
end

if is_cpml_xp
    n_st = nx - n_cpml_xp;
    for i = 1:n_cpml_xp
        Psi_hyx_xp(i,:,:) = cpml_b_mx_xp(i) * Psi_hyx_xp(i,:,:) ...
            + cpml_a_mx_xp(i)*(Ez(i+n_st+1,:,:)-Ez(i+n_st,:,:));
        Psi_hzx_xp(i,:,:) = cpml_b_mx_xp(i) * Psi_hzx_xp(i,:,:) ...
            + cpml_a_mx_xp(i)*(Ey(i+n_st+1,:,:)-Ey(i+n_st,:,:));
    end

        Hy(n_st+1:nx,:,:) = Hy(n_st+1:nx,:,:) ...
            + CPsi_hyx_xp(:,:,:) .* Psi_hyx_xp(:,:,:);
        Hz(n_st+1:nx,:,:) = Hz(n_st+1:nx,:,:) ...
            + CPsi_hzx_xp(:,:,:) .* Psi_hzx_xp(:,:,:);
end
```

**Listing 8.7** update_electric_field_CPML_ABC

```
% apply CPML to electric field components
if is_cpml_xn
    for i = 1:n_cpml_xn
        Psi_eyx_xn(i,:,:) = cpml_b_ex_xn(i) * Psi_eyx_xn(i,:,:) ...
            + cpml_a_ex_xn(i)*(Hz(i+1,:,:)-Hz(i,:,:));
        Psi_ezx_xn(i,:,:) = cpml_b_ex_xn(i) * Psi_ezx_xn(i,:,:) ...
            + cpml_a_ex_xn(i)*(Hy(i+1,:,:)-Hy(i,:,:));
    end
    Ey(2:n_cpml_xn+1,:,:) = Ey(2:n_cpml_xn+1,:,:) ...
            + CPsi_eyx_xn .* Psi_eyx_xn;
    Ez(2:n_cpml_xn+1,:,:) = Ez(2:n_cpml_xn+1,:,:) ...
            + CPsi_ezx_xn .* Psi_ezx_xn;
end

if is_cpml_xp
    n_st = nx - n_cpml_xp;
    for i = 1:n_cpml_xp
        Psi_eyx_xp(i,:,:) = cpml_b_ex_xp(i) * Psi_eyx_xp(i,:,:) ...
            + cpml_a_ex_xp(i)*(Hz(i+n_st,:,:)-Hz(i+n_st-1,:,:));
        Psi_ezx_xp(i,:,:) = cpml_b_ex_xp(i) * Psi_ezx_xp(i,:,:) ...
            + cpml_a_ex_xp(i)*(Hy(i+n_st,:,:)-Hy(i+n_st-1,:,:));
    end
```

Next Page

```
    Ey(n_st+1:nx,:,:) = Ey(n_st+1:nx,:,:) ...
        + CPsi_eyx_xp .* Psi_eyx_xp;
    Ez(n_st+1:nx,:,:) = Ez(n_st+1:nx,:,:) ...
        + CPsi_ezx_xp .* Psi_ezx_xp;
end
```

## 8.5 Simulation examples

So far we have discussed the CPML algorithm and have demonstrated its implementation in the MATLAB program. In this section, we provide examples in which the boundaries are realized by the CPML.

### 8.5.1 Microstrip low-pass filter

In Section 6.2, we demonstrated a low-pass filter geometry, which is illustrated in Figure 6.2. We assumed that the boundaries of the problem space were PEC. In this section we repeat the same example but assume that the boundaries are CPML. The section of the code that defines the boundaries in ***define_problem_space_parameters*** is shown in Listing 8.8. As can be followed in the listing, an air gap of 5 cells thickness is surrounding the filter circuit, and the boundaries are terminated by 8 cells thickness of the CPML.

**Listing 8.8** define_problem_space_parameters.m

```
% ==<boundary conditions>========
% Here we define the boundary conditions parameters
% 'pec' : perfect electric conductor
% 'cpml' : conlvolutional PML
% if cpml_number_of_cells is less than zero
% CPML extends inside of the domain rather than outwards

boundary.type_xn = 'cpml';
boundary.air_buffer_number_of_cells_xn = 5;
boundary.cpml_number_of_cells_xn = 8;

boundary.type_xp = 'cpml';
boundary.air_buffer_number_of_cells_xp = 5;
boundary.cpml_number_of_cells_xp = 8;

boundary.type_yn = 'cpml';
boundary.air_buffer_number_of_cells_yn = 5;
boundary.cpml_number_of_cells_yn = 8;

boundary.type_yp = 'cpml';
boundary.air_buffer_number_of_cells_yp = 5;
boundary.cpml_number_of_cells_yp = 8;

boundary.type_zn = 'cpml';
boundary.air_buffer_number_of_cells_zn = 5;
boundary.cpml_number_of_cells_zn = 8;
```