CHAPTER 13

Dispersive material modeling

One of the attractive features of the finite-difference time-domain (FDTD) method is that it allows for modeling behavior of complex media such as dispersion and nonlinearity. Accurate algorithms can be developed to model the electromagnetic properties of these media and to integrate within the discrete time domain solution. In this chapter we will present the simulation of dispersive materials using the FDTD method.

The electromagnetic material properties – permittivity, permeability, and conductivities – were treated as constants for a medium while developing the updating equations throughout the previous chapters. In many applications these parameters are relatively constant over the frequency band of interest and simulations based on constant parameter assumption yield sufficiently accurate results in many applications. On the other hand, in some applications, the values of material parameters vary significantly as functions of frequency in the frequency band of interest. For instance, electromagnetic properties of biological tissues, earth, and artificial metamaterials are highly dependent on frequency and their dispersive nature needs to be modeled accurately in FDTD simulations.

Dispersion modeling has been one of the key topics in FDTD research over the years and many algorithms have been introduced by researchers. Most of these algorithms are classified mainly under the auxiliary differential equation (ADE) [48–52], recursive convolution (RC) [53–57], and *Z*-transform [58,59] techniques. An in-depth review of these techniques for modeling dispersive and other complex media in the FDTD method is presented in [60].

In general, frequency dependence of permittivity, permeability, or conductivity can be expressed as a sum of rational functions of angular frequency, ω. The most common types of these rational functions are referred to as Debye, Lorentz, or Drude model. Debye models are commonly used to approximate the frequency behavior of biological tissues and soil permittivities. Lorentz models describe the frequency behavior of some metamaterials such as Double-Negative media close to resonances, while Drude models are useful in describing the behavior of metals at optical frequencies, and they can sometimes also be augmented by Lorentz terms [60]. It should be noted that some applications employ some other dispersion models described in the literature besides these three models. For instance, Cole-Cole is another model, yet a more general one than Debye, for biological applications, while Condon is a model used to describe chiral media.

In the following sections, we will present integration of Debye, Lorentz, and Drude models into FDTD by using the ADE technique. Then, we will illustrate an implementation of the resulting algorithms in the MATLAB® FDTD code.

13.1 Modeling dispersive media using ADE technique

13.1.1 Modeling Debye medium using ADE technique

In this section we illustrate an algorithm to model Debye medium based on the ADE technique as presented by [51]. Here we consider dispersive permittivity, while development of an algorithm for dispersive permeability or conductivity follows the same procedure.

Ampere's law is expressed for a dispersive medium with P poles as

$$\nabla \times \tilde{H} = j\omega\varepsilon_0\varepsilon_\infty\tilde{E} + \sigma\tilde{E} + \sum_{k=1}^{P}\tilde{J}_k \tag{13.1}$$

in frequency domain, while in the time domain

$$\nabla \times \overline{H} = \varepsilon_0\varepsilon_\infty\frac{\partial}{\partial t}\overline{E} + \sigma\overline{E} + \sum_{k=1}^{P}\overline{J}_k, \tag{13.2}$$

where ε_∞ is the relative permittivity of the medium at infinite frequencies, and $\overline{J}_k$ is the kth polarization current. In the frequency domain, the polarization current $\tilde{J}_k$ can be written for Debye model as

$$\tilde{J}_k = j\omega\varepsilon_0\frac{A_k(\varepsilon_s - \varepsilon_\infty)}{1 + j\omega\tau_k}\tilde{E} = j\omega\frac{\zeta_k}{1 + j\omega\tau_k}\tilde{E}. \tag{13.3}$$

where ε_s is the static relative permittivity, A_k is the amplitude of the kth term, and τ_k is the relaxation time of the kth term. Here we define $\zeta_k = A_k\varepsilon_0(\varepsilon_s - \varepsilon_\infty)$ for brevity. While developing the FDTD updating equations, as the first step, we need to express (13.2) in discrete time at time step $n + 0.5$ as

$$\nabla \times \overline{H}^{n+0.5} = \varepsilon_0\varepsilon_\infty\frac{\partial}{\partial t}\overline{E}^{n+0.5} + \sigma\overline{E}^{n+0.5} + \sum_{k=1}^{P}\overline{J}_k^{n+0.5}, \tag{13.4}$$

which requires the value of $\overline{J}_k^{n+0.5}$. This value can be retrieved as follows. One can arrange (13.3) as

$$j\omega\tau_k\tilde{J}_k + \tilde{J}_k = j\omega\zeta_k\tilde{E}, \tag{13.5}$$

which, then, can be transformed to time domain as a differential equation

$$\tau_k\frac{\partial}{\partial t}\overline{J}_k + \overline{J}_k = \zeta_k\frac{\partial}{\partial t}\overline{E}. \tag{13.6}$$

This equation is called an ADE, which can be used to retrieve a value for $\overline{J}_k^{n+0.5}$. It should be noted that various approaches can be employed to retrieve a value for the polarization current term in (13.6). The following approach is employed in [51].

Equation (13.6) can be represented in discrete time at time step $n + 0.5$ as

$$\tau_k \frac{\overline{J}_k^{n+1} - \overline{J}_k^n}{\Delta t} + \frac{\overline{J}_k^{n+1} + \overline{J}_k^n}{2} = \zeta_k \frac{\overline{E}^{n+1} - \overline{E}^n}{\Delta t}, \tag{13.7}$$

which leads to

$$\overline{J}_k^{n+1} = \frac{(2\tau_k - \Delta t)}{(2\tau_k + \Delta t)} \overline{J}_k^n + \frac{2\zeta_k}{(2\tau_k + \Delta t)} \left(\overline{E}^{n+1} - \overline{E}^n \right). \tag{13.8}$$

This equation yields $\overline{J}_k^{n+1}$ if the values of $\overline{J}_k^n$, $\overline{E}^{n+1}$, and $\overline{E}^n$ are known, however, the value at $n + 0.5$, $\overline{J}_k^{n+0.5}$, is needed in (13.4). One can write

$$\overline{J}_k^{n+0.5} = \frac{\overline{J}_k^{n+1} + \overline{J}_k^n}{2}, \tag{13.9}$$

which can be expressed, using $\overline{J}_k^{n+1}$ from (13.8), as

$$\overline{J}_k^{n+0.5} = \frac{2\tau_k}{(2\tau_k + \Delta t)} \overline{J}_k^n + \frac{\zeta_k}{(2\tau_k + \Delta t)} \left(E^{n+1} - E^n \right) \tag{13.10}$$

Now, (13.10) can be integrated into (13.4) as

$$\begin{aligned} \nabla \times \overline{H}^{n+0.5} = \varepsilon_0 \varepsilon_\infty \frac{\overline{E}^{n+1} - \overline{E}^n}{\Delta t} + \sigma \frac{\overline{E}^{n+1} + \overline{E}^n}{2} \\ + \sum_{k=1}^{P} \left(\frac{2\tau_k}{(2\tau_k + \Delta t)} \overline{J}_k^n + \frac{\zeta_k}{(2\tau_k + \Delta t)} \left(E^{n+1} - E^n \right) \right). \end{aligned} \tag{13.11}$$

which can be arranged, by moving the $\overline{E}^{n+1}$ term to left-hand side, to obtain an updating equation for the electric field as

$$\begin{aligned} \overline{E}^{n+1} = \frac{2\Delta t}{(2\varepsilon_0 \varepsilon_\infty + \sigma \Delta t + \xi)} \nabla \times \overline{H}^{n+0.5} + \frac{2\varepsilon_0 \varepsilon_\infty - \sigma \Delta t + \xi}{2\varepsilon_0 \varepsilon_\infty + \sigma \Delta t + \xi} \overline{E}^n \\ - \frac{2\Delta t}{(2\varepsilon_0 \varepsilon_\infty + \sigma \Delta t + \xi)} \sum_{k=1}^{P} \frac{2\tau_k}{(2\tau_k + \Delta t)} \overline{J}_k^n, \end{aligned} \tag{13.12}$$

where

$$\xi = \sum_{k=1}^{P} \frac{2\Delta t \zeta_k}{(2\tau_k + \Delta t)}. \tag{13.13}$$

An algorithm can be constructed using (13.8) and (13.12) as illustrated in Figure 13.1. At every time step, magnetic field components are updated as usual. Next, electric field components are updated using the past values of electric and magnetic field components, as well as the polarization current components following (13.12). Then, the polarization

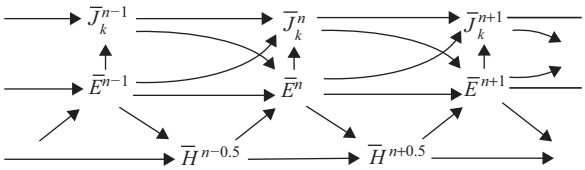

Figure 13.1 Update sequence of fields in the presented Debye modeling algorithm.

current components are calculated using the current and past values of electric field components and the past values of the polarization current components following (13.8). One should notice that this formulation requires an additional array to store $\overline{E}^n$ in order to implement (13.8).

13.1.2 Modeling Lorentz medium using ADE technique

In this section we illustrate an algorithm to model Lorentz medium based on the ADE technique. It should be noted that the procedure to develop an ADE and find a solution to the developed ADE is not unique. Here we discuss a procedure presented by [52] as an alternative to the one introduced by [51] that is presented in the previous section.

Ampere's law can be expressed for a dispersive medium with P poles, alternatively, as

$$\nabla \times \tilde{H} = j\omega\varepsilon_0\varepsilon_\infty\tilde{E} + \sigma\tilde{E} + j\omega\sum_{k=1}^{P}\tilde{Q}_k \tag{13.14}$$

in frequency domain, while in time domain

$$\nabla \times \overline{H} = \varepsilon_0\varepsilon_\infty\frac{\partial}{\partial t}\overline{E} + \sigma\overline{E} + \sum_{k=1}^{P}\frac{\partial}{\partial t}\overline{Q}_k. \tag{13.15}$$

Comparing (13.14) with (13.1), one can notice that they are the same when $\tilde{J}_k = j\omega\tilde{Q}_k$. The difference is that $\tilde{J}_k$ terms are used in [51] to develop ADEs for Debye and Lorentz dispersion models while $\tilde{Q}_k$ terms are used in [52] instead.

In the frequency domain, the term $\tilde{Q}_k$ can be written for Lorentz model as

$$\tilde{Q}_k = \frac{A_k\varepsilon_0(\varepsilon_s - \varepsilon_\infty)\omega_k^2}{\omega_k^2 + 2j\omega\delta_k - \omega^2}\tilde{E} = \frac{\psi_k}{\omega_k^2 + 2j\omega\delta_k - \omega^2}\tilde{E}, \tag{13.16}$$

where ω_k is the pole location and δ_k is the damping factor of the kth term. Here we define $\psi_k = A_k\varepsilon_0(\varepsilon_s - \varepsilon_\infty)\omega_k^2$ for brevity. An ADE can be constructed from (13.16) as

$$\left(\frac{\partial^2}{\partial t^2} + 2\delta_k\frac{\partial}{\partial t} + \omega_k^2\right)\overline{Q}_k = \psi_k\overline{E},$$

which can be expressed in discrete time using central difference approximation at time step n as

$$\frac{\overline{Q}_k^{n+1} - 2\overline{Q}_k^n + \overline{Q}_k^{n-1}}{\Delta t^2} + \delta_k \frac{\overline{Q}_k^{n+1} - \overline{Q}_k^{n-1}}{\Delta t} + \omega_k^2 \overline{Q}_k^n = \psi_k \overline{E}^n. \tag{13.17}$$

Equation (13.17) can be arranged to calculate the future value of $\overline{Q}_k$, such that

$$\overline{Q}_k^{n+1} = \frac{2 - (\Delta t)^2 \omega_k^2}{(\delta_k \Delta t + 1)} \overline{Q}_k^n + \frac{(\delta_k \Delta t - 1)}{(\delta_k \Delta t + 1)} \overline{Q}_k^{n-1} + \frac{(\Delta t)^2 \psi_k}{(\delta_k \Delta t + 1)} \overline{E}^n. \tag{13.18}$$

Meanwhile, (13.15) can be represented in discrete time at time step $n + 0.5$ as

$$\nabla \times \overline{H}^{n+0.5} = \varepsilon_0 \varepsilon_\infty \frac{\overline{E}^{n+1} - \overline{E}^n}{\Delta t} + \sigma \frac{\overline{E}^{n+1} + \overline{E}^n}{2} + \frac{1}{\Delta t} \sum_{k=1}^{P} \left(Q_k^{n+1} - Q_k^n \right), \tag{13.19}$$

which can be arranged, by moving the $\overline{E}^{n+1}$ term to left-hand side, to obtain an updating equation for the electric field as

$$\overline{E}^{n+1} = \frac{2\Delta t}{2\varepsilon_0 \varepsilon_\infty + \Delta t \sigma} \nabla \times \overline{H}^{n+0.5} + \frac{2\varepsilon_0 \varepsilon_\infty - \Delta t \sigma}{2\varepsilon_0 \varepsilon_\infty + \Delta t \sigma} \overline{E}^n - \frac{2}{2\varepsilon_0 \varepsilon_\infty + \Delta t \sigma} \sum_{k=1}^{P} \left(Q_k^{n+1} - Q_k^n \right) \tag{13.20}$$

An algorithm can be constructed using (13.18) and (13.20) as illustrated in Figure 13.2. At every time step, magnetic field components are updated as usual. Next, the new values of $\overline{Q}_k$ are calculated using their past values and the past values of the electric field components following (13.18). Then, electric field components are updated using the past values of electric and magnetic field components as well as the new and past values of $\overline{Q}_k$ following (13.20). One should notice that this formulation requires additional arrays to store $\overline{Q}_k^n$ and $\overline{Q}_k^{n-1}$ in order to implement (13.18).

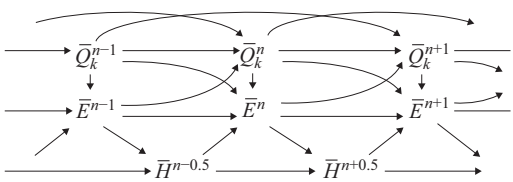

Figure 13.2 Update sequence of fields in the presented Lorentz and Drude modeling algorithms.

13.1.3 Modeling Drude medium using ADE technique

The ADE technique introduced by [52], discussed above, was employed to model Drude medium in [61]. In this section we illustrate the formulation presented in [61].

Here we consider (13.15) as the starting point to develop an algorithm for Drude medium. In the frequency domain, the term $\tilde{Q}_k$ can be written for Drude model as

$$\tilde{Q}_k = \frac{\varepsilon_0 \omega_k^2}{\omega^2 - j\omega\gamma_k} \tilde{E}, \tag{13.21}$$

where ω_k is the plasma frequency, and γ_k is the inverse of the pole relaxation time. An ADE can be constructed from (13.21) as

$$\left(-\frac{\partial^2}{\partial t^2} - \gamma_k \frac{\partial}{\partial t}\right) \overline{Q}_k = \varepsilon_0 \omega_k^2 \overline{E}. \tag{13.22}$$

Then, (13.22) can be expressed in discrete time using central difference approximation at time step n, and rearranged to calculate the future value of $\overline{Q}_k$ as

$$\overline{Q}_k^{n+1} = \frac{4}{\Delta t\gamma_k + 2} \overline{Q}_k^n + \frac{\Delta t\gamma_k - 2}{\Delta t\gamma_k + 2} \overline{Q}_k^{n-1} - \frac{2\Delta t^2 \varepsilon_0 \omega_k^2}{\Delta t\gamma_k + 2} \overline{E}^n. \tag{13.23}$$

Equation (13.23) is in the same form as (13.18), while its coefficients are different. Therefore, we can use (13.23) and (13.20) to construct an ADE algorithm to model Drude medium, which is essentially the same as the one constructed for Lorentz medium as shown in Figure 13.2: at every time step, magnetic field components are updated as usual. Next, the new values of $\overline{Q}_k$ are calculated following (13.23). Then, electric field components are updated following (13.20).

13.2 MATLAB® implementation of ADE algorithm for Lorentz medium

The presented Lorentz medium modeling ADE algorithm is implemented into the MATLAB FDTD code that also includes the TF/SF formulation. While a Lorentz model of a single term ($P = 1$ in (13.14)) is considered in the presented implementation, it is possible to extend the implementation to accommodate multiple terms as well, where $P > 1$. The details of the implementation are presented in the following subsections.

13.2.1 Definition of Lorentz material parameters

A dispersive material is described using additional parameters along with the standard permittivity, permeability, and conductivity parameters. In the case of the Lorentz model infinite relative permittivity ε_∞, static relative permittivity ε_s, amplitude A_k, pole location ω_k, and damping factor δ_k of the kth term are the parameters that need to be defined for an FDTD simulation.

In the MATLAB FDTD code, the material parameters are defined in the subroutine ***define_problem_space_parameters***, where a structure array, referred to as **material_types**, is constructed to store the electromagnetic parameters of materials that are used in the simulation. In the presented implementation, new fields are added to the **material_types** structure

to support Lorentz media modeling. Listing 13.1 shows an element of the **material_types** array including the Lorentz parameters. Here, **lorentz_eps_s** refers to the static relative permittivity, **lorentz_A** refers to the amplitude, **lorentz_omega** refers to the pole location, and **lorentz_delta** refers to the damping factor. The parameter **lorentz_dispersive** is used to identify the material type if it is Lorentz dispersive or not. As for the infinite permittivity, one can realize in (13.1) and (13.2) that it is actually the same as the relative permittivity of a nondispersive medium. Therefore, the parameter **eps_r** is used to store the relative infinite permittivity if the material is of Lorentz type.

Listing 13.1 define_problem_sec_parameters.m

```
% lorentz material
material_types(4).eps_r    =2; % eps_inf for dispersive material
material_types(4).mu_r     =1;
material_types(4).sigma_e  =0;
material_types(4).sigma_m  =0;
material_types(4).color    =[0 0 1];
material_types(4).lorentz_dispersive = true;
material_types(4).lorentz_eps_s = 5;
material_types(4).lorentz_A = 1;
material_types(4).lorentz_omega = 2*pi*2e9;
material_types(4).lorentz_delta = pi*2e9;
```

13.2.2 Material grid construction for Lorentz objects

The FDTD material grid is constructed in the subroutine ***initialize_fdtd_material_grid***, where three-dimensional arrays are constructed for permittivity, permeability, and conductivity distributions in the problem space. Later, these arrays are used to calculate updating coefficients. In order to account for Lorentz modeling, a new subroutine, referred to as ***create_dispersive_objects***, is implemented and called at the end of the subroutine ***initialize_fdtd_material_grid***.

Listing 13.2 shows the subroutine ***create_dispersive_objects***. At the first part of the code, three-dimensional arrays are created to store the x, y, and z coordinates of the electric field components. These arrays are then used to calculate the distances of the field components from the center of a sphere, and identify the indices of the field components that are located inside the sphere. Once the indices of these field components are identified, corresponding elements in three-dimensional arrays **dispersive_Ex**, **dispersive_Ey**, and **dispersive_Ez** are assigned with the material index of the sphere if the sphere is dispersive. Then similarly, the field components that are located in a brick are identified and the corresponding elements in **dispersive_Ex**, **dispersive_Ey**, and **dispersive_Ez** are assigned with the material index of the brick if the brick is dispersive. Once these calculations are completed, these three arrays will include distribution of the material type indices of the dispersive material types.

Next, a new structure array, referred to as **lorentz**, is constructed. Each element in this array corresponds to a dispersive material type in the **material_types** structure array. Here, for a dispersive material type, the **dispersive_Ex**, **dispersive_Ey**, and **dispersive_Ez** arrays are

Listing 13.2 create_dispersive_objects.m

```
% Three dimensional arrays to store field coordinates
% Used to identify field components that are inside a sphere
Ex_x_coordinates = zeros(nx, nyp1, nzp1);
Ex_y_coordinates = zeros(nx, nyp1, nzp1);
Ex_z_coordinates = zeros(nx, nyp1, nzp1);

Ey_x_coordinates = zeros(nxp1, ny, nzp1);
Ey_y_coordinates = zeros(nxp1, ny, nzp1);
Ey_z_coordinates = zeros(nxp1, ny, nzp1);

Ez_x_coordinates = zeros(nxp1, nyp1, nz);
Ez_y_coordinates = zeros(nxp1, nyp1, nz);
Ez_z_coordinates = zeros(nxp1, nyp1, nz);

% Arrays to store material indices distribution of dispersive objects
dispersive_Ex = zeros(nx, nyp1, nzp1);
dispersive_Ey = zeros(nxp1, ny, nzp1);
dispersive_Ez = zeros(nxp1, nyp1, nz);

% calculate field coordinates
for ind = 1:nx
    Ex_x_coordinates(ind,:,:) = (ind - 0.5) * dx + fdtd_domain.min_x;
end
for ind = 1:nyp1
    Ex_y_coordinates(:,ind,:) = (ind - 1) * dy + fdtd_domain.min_y;
end
for ind = 1:nzp1
    Ex_z_coordinates(:,:,ind) = (ind - 1) * dz + fdtd_domain.min_z;
end

for ind = 1:nxp1
    Ey_x_coordinates(ind,:,:) = (ind - 1) * dx + fdtd_domain.min_x;
end
for ind = 1:ny
    Ey_y_coordinates(:,ind,:) = (ind - 0.5) * dy + fdtd_domain.min_y;
end
for ind = 1:nzp1
    Ey_z_coordinates(:,:,ind) = (ind - 1) * dz + fdtd_domain.min_z;
end

for ind = 1:nxp1
    Ez_x_coordinates(ind,:,:) = (ind - 1) * dx + fdtd_domain.min_x;
end
for ind = 1:nyp1
    Ez_y_coordinates(:,ind,:) = (ind - 1) * dy + fdtd_domain.min_y;
end
for ind = 1:nz
    Ez_z_coordinates(:,:,ind) = (ind - 0.5) * dz + fdtd_domain.min_z;
end

disp('creating dispersive spheres');
for ind=1:number_of_spheres
    mat_ind = spheres(ind).material_type;
    material = material_types(mat_ind);
    if material.lorentz_dispersive
```

```
        % distance of the Ex components from the center of the sphere
        distance = sqrt((spheres(ind).center_x - Ex_x_coordinates).^2 ...
              + (spheres(ind).center_y - Ex_y_coordinates).^2 ...
              + (spheres(ind).center_z - Ex_z_coordinates).^2);
        I = find(distance<=spheres(ind).radius);
        dispersive_Ex(I) = mat_ind;

        % distance of the Ey components from the center of the sphere
        distance = sqrt((spheres(ind).center_x - Ey_x_coordinates).^2 ...
              + (spheres(ind).center_y - Ey_y_coordinates).^2 ...
              + (spheres(ind).center_z - Ey_z_coordinates).^2);
        I = find(distance<=spheres(ind).radius);
        dispersive_Ey(I) = mat_ind;

        % distance of the Ez components from the center of the sphere
        distance = sqrt((spheres(ind).center_x - Ez_x_coordinates).^2 ...
              + (spheres(ind).center_y - Ez_y_coordinates).^2 ...
              + (spheres(ind).center_z - Ez_z_coordinates).^2);
        I = find(distance<=spheres(ind).radius);
        dispersive_Ez(I) = mat_ind;
    end
end

disp('creating dispersive bricks');
for ind = 1:number_of_bricks
    mat_ind = bricks(ind).material_type;
    material = material_types(mat_ind);
    if material.lorentz_dispersive
        % convert brick end coordinates to node indices
        ni = get_node_indices(bricks(ind), fdtd_domain);
        is = ni.is; js = ni.js; ks = ni.ks;
        ie = ni.ie; je = ni.je; ke = ni.ke;

        dispersive_Ex (is:ie-1, js:je, ks:ke) = mat_ind;
        dispersive_Ey (is:ie, js:je-1, ks:ke) = mat_ind;
        dispersive_Ez (is:ie, js:je, ks:ke-1) = mat_ind;
    end
end

% create a new structure array to store indices of field components
% that need to be updated using Lorentz formulation
number_of_lorentz = 0;
for ind = 1:number_of_material_types
    material = material_types(ind);
    if material.lorentz_dispersive
        number_of_lorentz = number_of_lorentz + 1;

        lorentz(number_of_lorentz).material = ind;

        I = find(dispersive_Ex == ind);
        lorentz(number_of_lorentz).Ex_indices = I;
        eps_r_x(I) = material.eps_r;
        sigma_e_x(I) = material.sigma_e;
```

```
        I = find(dispersive_Ey == ind);
        lorentz(number_of_lorentz).Ey_indices = I;
        eps_r_y(I) = material.eps_r;
        sigma_e_y(I) = material.sigma_e;

        I = find(dispersive_Ez == ind);
        lorentz(number_of_lorentz).Ez_indices = I;
        eps_r_z(I) = material.eps_r;
        sigma_e_z(I) = material.sigma_e;
    end
end

clear Ex_x_coordinates Ex_y_coordinates Ex_z_coordinates
clear Ey_x_coordinates Ey_y_coordinates Ey_z_coordinates
clear Ez_x_coordinates Ez_y_coordinates Ez_z_coordinates
clear dispersive_Ex dispersive_Ey dispersive_Ez
```

queried for the index of the material type in consideration, and the indices of the elements associated with this material type are identified. Then these indices are stored as linear arrays **Ex_indices**, **Ey_indices**, and **Ez_indices** in the **lorentz** structure. The indices of the field components that need to be updated using Lorentz formulation are now known through the arrays **Ex_indices**, **Ey_indices**, and **Ez_indices**.

13.2.3 Initialization of updating coefficients

The updating coefficients are initialized in the subroutine ***initialize_updating_coefficients*** in the MATLAB FDTD code. A new subroutine, ***initialize_dispersive_coefficients***, is implemented and called after the calculation of general updating coefficients in ***initialize_updating_coefficients***.

It should be noted that (13.18) and (13.20) are vector equations and they need to decomposed as scalar equations in terms of x, y, and z components. For instance, (13.18) can be expressed for the x, y, and z components as

$$Q_{k,x}^{n+1}(i,j,k) = C_{qq}Q_{k,x}^{n}(i,j,k) + C_{qqm}Q_{k,x}^{n-1}(i,j,k) + C_{qe}E_{x}^{n}(i,j,k), \tag{13.24a}$$

$$Q_{k,y}^{n+1}(i,j,k) = C_{qq}Q_{k,y}^{n}(i,j,k) + C_{qqm}Q_{k,y}^{n-1}(i,j,k) + C_{qe}E_{y}^{n}(i,j,k), \tag{13.24b}$$

$$Q_{k,z}^{n+1}(i,j,k) = C_{qq}Q_{k,z}^{n}(i,j,k) + C_{qqm}Q_{k,z}^{n-1}(i,j,k) + C_{qe}E_{z}^{n}(i,j,k). \tag{13.24c}$$

Here, C_{qq}, C_{qqm}, and C_{qe} are coefficients associated with the respective field components and they are defined as

$$C_{qq} = \frac{2-(\Delta t)^2\omega_k^2}{(\delta_k\Delta t+1)}, \quad C_{qqm} = \frac{(\delta_k\Delta t-1)}{(\delta_k\Delta t+1)}, \quad C_{qe} = \frac{(\Delta t)^2\psi_k}{(\delta_k\Delta t+1)}.$$

Similarly, (13.20) can be expressed for E_x^{n+1} as

$$\begin{aligned} E_x^{n+1}(i,j,k) = {} & C_{exe}(i,j,k)E_x^n(i,j,k) \\ & + C_{exhz}(i,j,k)\left(H_z^{n+0.5}(i,j,k) - H_z^{n+0.5}(i,j-1,k)\right) \\ & + C_{exhy}(i,j,k)\left(H_y^{n+0.5}(i,j,k) - H_y^{n+0.5}(i,j,k-1)\right) \\ & - \frac{\Delta y}{\Delta t} C_{exhz}(i,j,k)\sum_{k=1}^{P}\left(Q_{k,x}^{n+1} - Q_{k,x}^{n}\right). \end{aligned} \tag{13.25}$$

One can notice that C_{exe}, C_{exhz}, and C_{exhy} are the same as the general updating coefficients (i.e., Chapter 1 (1.26)). Also, the coefficient that is multiplied by the summation of the differences of $Q_{k,x}^{n+1}$ and $Q_{k,x}^{n}$ terms is simply C_{exhz} scaled by $\frac{\Delta y}{\Delta t}$. Equation (13.25) is essentially the same as (1.26) in Chapter 1, except that it has an additional last term due to the dispersion.

Similarly, (13.20) can be expressed for E_y^{n+1} as

$$\begin{aligned} E_y^{n+1}(i,j,k) = {} & C_{eye}(i,j,k)E_y^n(i,j,k) \\ & + C_{eyhx}(i,j,k)\left(H_x^{n+0.5}(i,j,k) - H_x^{n+0.5}(i,j,k-1)\right) \\ & + C_{eyhz}(i,j,k)\left(H_z^{n+0.5}(i,j,k) - H_z^{n+0.5}(i-1,j,k)\right) \\ & - \frac{\Delta z}{\Delta t} C_{eyhx}(i,j,k)\sum_{k=1}^{P}\left(Q_{k,y}^{n+1} - Q_{k,y}^{n}\right), \end{aligned} \tag{13.26}$$

and for E_z^{n+1} as

$$\begin{aligned} E_z^{n+1}(i,j,k) = {} & C_{eze}(i,j,k)E_z^n(i,j,k) \\ & + C_{ezhy}(i,j,k)\left(H_y^{n+0.5}(i,j,k) - H_y^{n+0.5}(i-1,j,k)\right) \\ & + C_{ezhx}(i,j,k)\left(H_x^{n+0.5}(i,j,k) - H_x^{n+0.5}(i,j-1,k)\right) \\ & - \frac{\Delta x}{\Delta t} C_{ezhy}(i,j,k)\sum_{k=1}^{P}\left(Q_{k,z}^{n+1} - Q_{k,z}^{n}\right). \end{aligned} \tag{13.27}$$

Listing 13.3 shows the implementation of ***initialize_dispersive_coefficients***. First, for each Lorentz medium, arrays are constructed to store the additional terms $\overline{Q}_k^{n+1}$ and $\overline{Q}_k^{n}$ required by the Lorentz ADE algorithm and initialized to zero. Then, the coefficients C_{qq}, C_{qqm}, and C_{qe} in (13.24) are calculated and stored. Since the coefficients in (13.25) are the

Listing 13.3 initialize_dispersive_coefficients.m

```
% initialize Lorentz updating coefficients

for ind = 1:number_of_lorentz

    material = material_types(lorentz(ind).material);

    n_elements = size(lorentz(ind).Ex_indices,2);
    lorentz(number_of_lorentz).Qxp = zeros(1, n_elements);
    lorentz(number_of_lorentz).Qx  = zeros(1, n_elements);

    n_elements = size(lorentz(ind).Ey_indices,2);
    lorentz(number_of_lorentz).Qyp = zeros(1, n_elements);
    lorentz(number_of_lorentz).Qy  = zeros(1, n_elements);

    n_elements = size(lorentz(ind).Ez_indices,2);
    lorentz(number_of_lorentz).Qzp = zeros(1, n_elements);
    lorentz(number_of_lorentz).Qz  = zeros(1, n_elements);

    psi = eps_0 * material.lorentz_A ...
       * (material.lorentz_eps_s - material.eps_r) ...
       * material.lorentz_omega^2;

    den = material.lorentz_delta * dt + 1;
    lorentz(number_of_lorentz).Cqq =  (2 - dt^2 ...
       * material.lorentz_omega^2)/den;
    lorentz(number_of_lorentz).Cqqm =  ...
       (material.lorentz_delta * dt - 1)/den;
    lorentz(number_of_lorentz).Cqe =  ...
       dt^2 * psi / den;
end
```

same as the already calculated general updating coefficients, they do not need to be stored separately.

13.2.4 Field updates in time-marching loop

Electric fields are updated in the subroutine ***update_electric_fields*** at every time step of the FDTD time-marching loop. This subroutine is modified based on the sequence of field updates illustrated in Figure 13.2.

Listing 13.4 shows the modified section of the code in ***update_electric_fields***. In the first part of the code the $\overline{Q}_k^{n+1}$ terms are calculated using $\overline{Q}_k^n$, $\overline{Q}_k^{n-1}$, and $\overline{E}^n$ following (13.18). Then regular field updates are performed, where the components of $\overline{E}^{n+1}$ are calculated for the current time step. Then, the difference of Lorentz terms $\overline{Q}_k^{n+1}$ and $\overline{Q}_k^n$ is multiplied by the associated coefficients and added to the electric field components following (13.20). One can notice in the code that the arrays **Ex_indices**, **Ey_indices**, and **Ez_indices** are used to access to the electric field components that are associated with the Lorentz materials in the problem space.

Listing 13.4 update_electric_fields.m

```
% update the additional Lorents terms
for ind = 1:number_of_lorentz

   Cqq  = lorentz(number_of_lorentz).Cqq;
   Cqqm = lorentz(number_of_lorentz).Cqqm;
   Cqe  = lorentz(number_of_lorentz).Cqe;

   Qx = lorentz(ind).Qxp;
   lorentz(ind).Qxp = ...
         Cqq   * lorentz(ind).Qxp ...
       + Cqqm * lorentz(ind).Qx  ...
       + Cqe   * Ex(lorentz(ind).Ex_indices);
   lorentz(ind).Qx = Qx;

   Qy = lorentz(ind).Qyp;
   lorentz(ind).Qyp = ...
         Cqq   * lorentz(ind).Qyp ...
       + Cqqm * lorentz(ind).Qy  ...
       + Cqe   * Ey(lorentz(ind).Ey_indices);
   lorentz(ind).Qy = Qy;

   Qz = lorentz(ind).Qzp;
   lorentz(ind).Qzp = ...
         Cqq   * lorentz(ind).Qzp ...
       + Cqqm * lorentz(ind).Qz  ...
       + Cqe   * Ez(lorentz(ind).Ez_indices);
   lorentz(ind).Qz = Qz;
end

% Regular field updates
Ex(1:nx,2:ny,2:nz) = Cexe(1:nx,2:ny,2:nz).*Ex(1:nx,2:ny,2:nz) ...
                     + Cexhz(1:nx,2:ny,2:nz).*...
                     (Hz(1:nx,2:ny,2:nz)-Hz(1:nx,1:ny-1,2:nz)) ...
                     + Cexhy(1:nx,2:ny,2:nz).*...
                     (Hy(1:nx,2:ny,2:nz)-Hy(1:nx,2:ny,1:nz-1));

Ey(2:nx,1:ny,2:nz) = Ceye(2:nx,1:ny,2:nz).*Ey(2:nx,1:ny,2:nz) ...
                     + Ceyhx(2:nx,1:ny,2:nz).* ...
                     (Hx(2:nx,1:ny,2:nz)-Hx(2:nx,1:ny,1:nz-1)) ...
                     + Ceyhz(2:nx,1:ny,2:nz).* ...
                     (Hz(2:nx,1:ny,2:nz)-Hz(1:nx-1,1:ny,2:nz));

Ez(2:nx,2:ny,1:nz) = Ceze(2:nx,2:ny,1:nz).*Ez(2:nx,2:ny,1:nz) ...
                     + Cezhy(2:nx,2:ny,1:nz).* ...
                     (Hy(2:nx,2:ny,1:nz)-Hy(1:nx-1,2:ny,1:nz)) ...
                     + Cezhx(2:nx,2:ny,1:nz).*...
                     (Hx(2:nx,2:ny,1:nz)-Hx(2:nx,1:ny-1,1:nz));

% Add additional Lorentz terms to electric field components
for ind = 1:number_of_lorentz
   indices = lorentz(ind).Ex_indices;
   Ex(indices) = Ex(indices) ...
      - (dy/dt) * Cexhz(indices) .* (lorentz(ind).Qxp - lorentz(ind).Qx);
```

```
    indices = lorentz(ind).Ey_indices;
    Ey(indices) = Ey(indices) ...
        - (dz/dt) * Ceyhx(indices) .* (lorentz(ind).Qyp - lorentz(ind).Qy);

    indices = lorentz(ind).Ez_indices;
    Ez(indices) = Ez(indices) ...
        - (dx/dt) * Cezhy(indices) .* (lorentz(ind).Qzp - lorentz(ind).Qz);
end
```

13.3 Simulation examples

13.3.1 Scattering from a dispersive sphere

In this first example, we consider a single-term Lorentz dispersive sphere with a radius of 10 cm. The Lorentz parameters of the sphere are $\varepsilon_\infty = 2$, $\varepsilon_s = 5$, $A_1 = 1$, $\omega_1 = 4\pi \times 10^9$, and $\delta_1 = 2\pi \times 10^9$. The definition of these parameters in the FDTD code is shown in Listing 13.1. Examining (13.14) and (13.16), one can describe an equivalent complex relative permittivity for a single term Lorentz medium as

$$\varepsilon_r(\omega) = \varepsilon_\infty + \frac{\sigma}{j\omega\varepsilon_0} + \frac{A_1(\varepsilon_s - \varepsilon_\infty)\omega_1^2}{\omega_1^2 + 2j\omega\delta_1 - \omega^2}. \tag{13.28}$$

The relative permittivity is calculated using (13.28) for the above parameter values as a function of frequency and plotted in Figure 13.3.

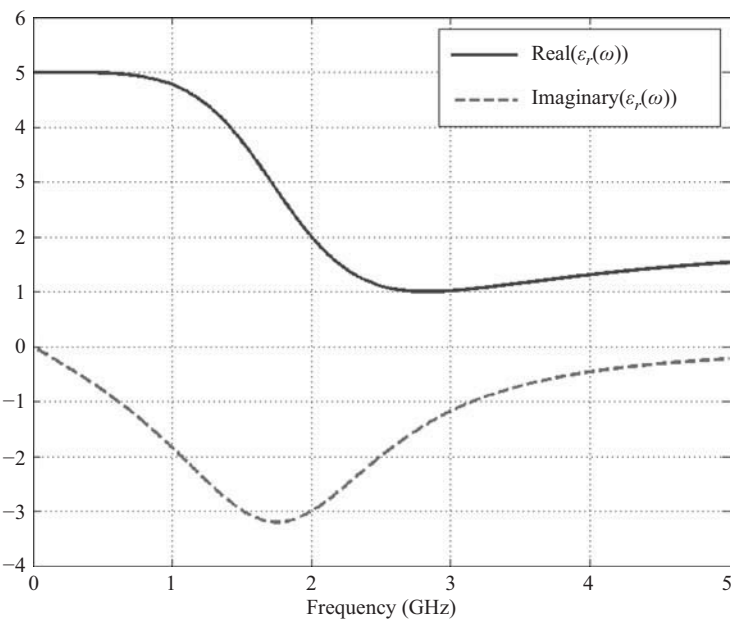

Figure 13.3 Complex relative permittivity versus frequency.

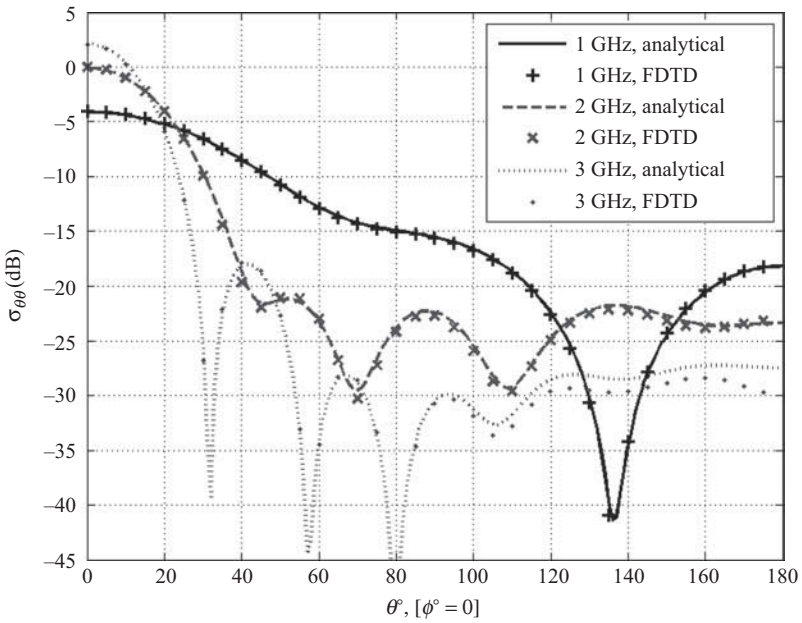

Figure 13.4 Co-polarized RCS of a dispersive sphere at three frequencies.

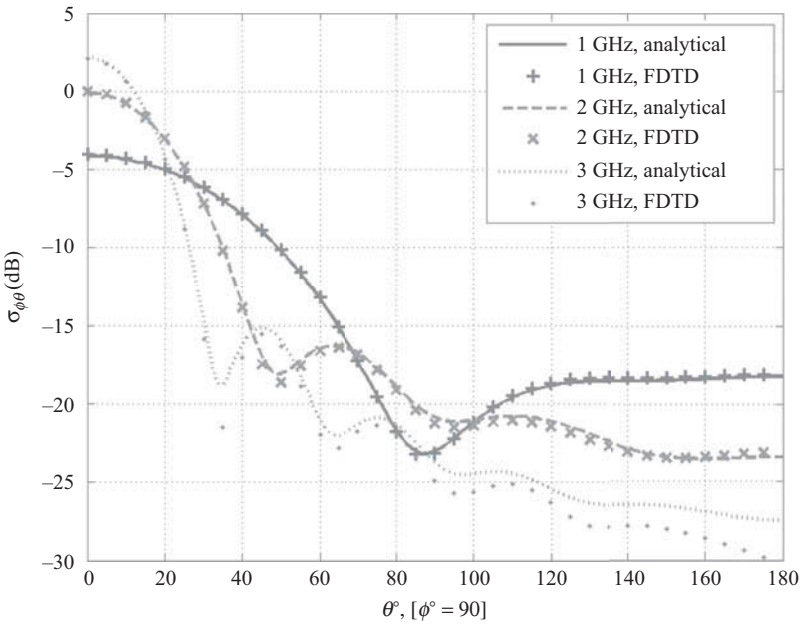

Figure 13.5 Cross-polarized RCS of a dispersive sphere at three frequencies.

The sphere is illuminated by an x-polarized plane wave with a Gaussian waveform that propagates in the positive z direction. The problem space is modeled using a cell size of 5 mm on a side and the simulation is performed for 3,000 time steps. The bistatic radar cross-section of the sphere is calculated at frequencies 1, 2, and 3 GHz. At these frequencies, the relative permittivity values are $\varepsilon_r(1\text{ GHz}) = 4.769 - 1.846j$, $\varepsilon_r(2\text{ GHz}) = 2 - 3j$, and $\varepsilon_r(3\text{ GHz}) = 1.016 - 1.18j$, as can be verified from Figure 13.3.

The results for $\sigma_{\theta\theta}$ in the xz-plane are shown in Figure 13.4, whereas the results for $\sigma_{\phi\theta}$ in the yz-plane are shown in Figure 13.5 along with the analytical solutions obtained using the program presented in [62]. A very good agreement can be observed between the FDTD simulation results and the analytical solutions except for the slight differences at the back-scatter angles at highest frequency. The simulation accuracy can be improved by using smaller cells or by employing an averaging scheme for media properties on the boundary between the sphere surface and surrounding media. For instance, such a method to determine effective permittivity at the interface of dispersive dielectrics is presented in [63].

13.4 Exercises

13.1 Consider the problem in Section 13.3.1. The dispersive sphere material has the relative permittivity of $\varepsilon_r = 4.769 - 1.846j$ at 1 GHz. A lossy and nondispersive medium can be imagined to have this particular relative permittivity value 1 GHz. For instance, the complex relative permittivity of such a medium can be expressed as $\varepsilon_{rc} = \varepsilon_r' - j\frac{\sigma}{\omega\varepsilon_0}$. Hence, $\varepsilon_r' = 4.769$ and $\sigma = 1.846\ \omega\varepsilon_0 = 0.1027$ at 1 GHz. Model the sphere as a lossy and nondispersive by using these relative permittivity and conductivity values and run the simulation to calculate the co- and cross-polarized RCS of the sphere. Verify that the results match with the ones in 13.4 and 13.5 at 1 GHz. Note that the results will match only at 1 GHz.

13.2 Consider the dielectric cube example in Section 11.5.2, and assume that it is made of a dispersive material described by the one in Figure 13.3. Run the simulation and obtain the RCS of the cube at 1, 2, and 3 GHz.

One can employ the procedure discussed in Exercise 13.1 to verify that the calculated results are correct: run the simulation for each of these frequencies and each time model the cube as a lossy and nondispersive object that has the corresponding complex permittivity at the frequency of interest. Result of the dispersive media simulation and the equivalent lossy and nondispersive media simulation should be the same at a frequency of interest.

Follow this approach and verify the results of the dispersive cube simulation. Notice that the complex relative permittivity values at 1, 2, and 3 GHz are listed in Section 13.3.1.