CHAPTER 5

Source waveforms and time to frequency domain transformation

Sources are necessary components of a finite-difference time-domain (FDTD) simulation and their types vary depending on the type of problem under consideration. Usually sources are of two types: (1) *near*, such as the voltage and current sources described in Chapter 4; and (2) *far*, such as the incident fields appearing in scattering problems. In any case, a source excites electric and magnetic fields with a waveform as a function of time. The type of waveform can be selected specific to the problem under consideration. However, some limitations of the FDTD method should be kept in mind while constructing the source waveforms to obtain a valid and accurate simulation result.

One of the considerations for the source waveform construction is the spectrum of the frequency components of the waveform. A temporal waveform is the sum of time-harmonic waveforms with a spectrum of frequencies that can be obtained using the Fourier transform. The temporal waveform is said to be in *time domain*, and its Fourier transformed form is said to be in *frequency domain*. The Fourier transform of the source waveforms and other output parameters can be used to obtain some useful results in the frequency domain. In this chapter, we discuss selected types of waveforms and then introduce the numerical algorithms for calculating the Fourier transform of temporal functions in discrete time.

5.1 Common source waveforms for FDTD simulations

A source waveform should be chosen such that its frequency spectrum includes all the frequencies of interest for the simulation, and it should have a smooth turn-on and turn-off to minimize the undesired effects of high-frequency components. A sine or a cosine function is a single-frequency waveform, whereas other waveforms such as Gaussian pulse, time derivative of Gaussian pulse, cosine-modulated Gaussian pulse, and Blackman-Harris window are multiple frequency waveforms. The type and parameters of the waveforms can be chosen based on the frequency spectrum of the waveform of interest.

5.1.1 Sinusoidal waveform

Definition and construction of a sinusoidal waveform was presented in Chapter 4. As mentioned before, a sinusoidal waveform is ideally a single-frequency waveform. However, in an FDTD simulation the waveform can be excited for a limited duration, and the turn-on and turn-off of the finite duration of the waveform is going to add other frequency components to the spectrum. The initial conditions for the sources are zero before the simulation is started in an FDTD simulation, and the sources are active during the duration of simulation. Therefore, if a sinusoidal signal ($\sin(2\pi t)$, $0 < t < 4$) is used for a source its duration is finite as shown in Figure 5.1(a).

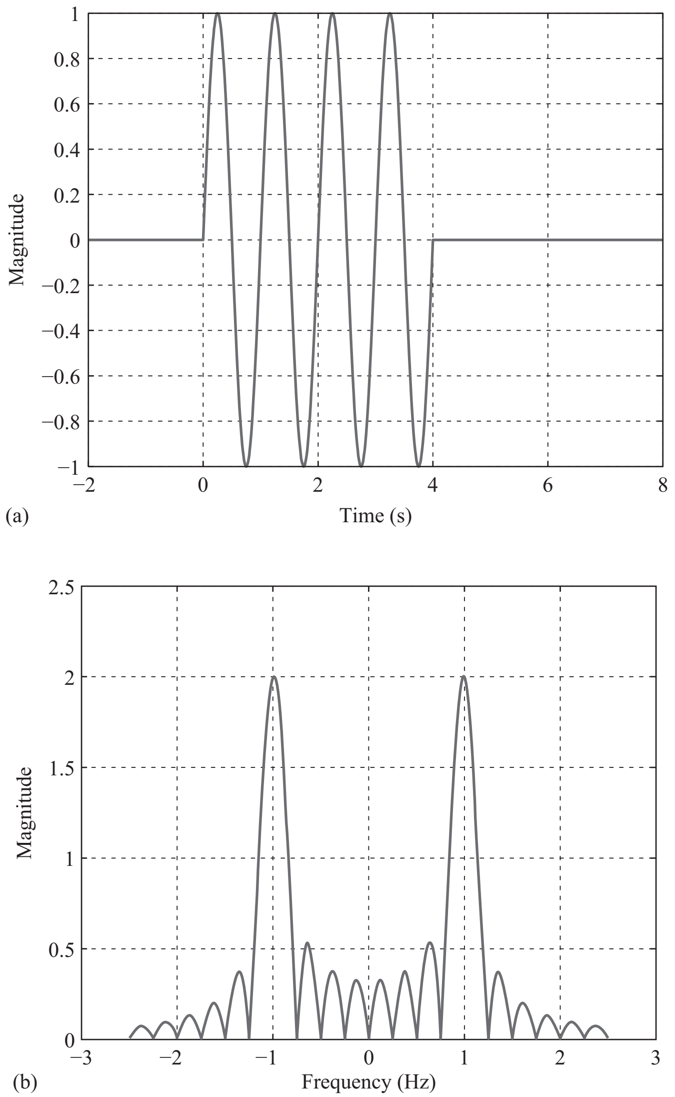


Figure 5.1 A sinusoidal waveform excited for 4 s: (a) $x(t)$ and (b) $X(\omega)$ magnitude.

The Fourier transform of a continuous time function $x(t)$ is $X(\omega)$ given by

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt, \tag{5.1}$$

while the inverse Fourier transform is

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}d\omega. \tag{5.2}$$

The Fourier transform of the signal in Figure 5.1(a) is a complex function, and its magnitude is plotted in Figure 5.1(b). The frequency spectrum of the signal in Figure 5.1(a) covers a range of frequencies while being maximum at 1 Hz, which is the frequency of the finite sinusoidal function. If the simulation runs for a longer duration as shown in Figure 5.2(a), the Fourier transform of the waveform becomes more pronounced at 1 Hz as shown in Figure 5.2(b). In Figure 5.1(b) the ratio of the highest harmonic to the main signal at 1 Hz is $0.53/2 \equiv -11.53$ dB, while in Figure 5.2(b) this ratio is $0.96/4 \equiv -12.39$ dB. In this case, running the simulation twice as long in time reduced the effect of the highest harmonic.

If it is intended to observe the response of an electromagnetic problem due to a sinusoidal excitation, the simulation should be run long enough such that the transient response due to the turn on of the sources die out and only the sinusoidal response persists.

5.1.2 Gaussian waveform

Running an FDTD simulation will yield numerical results for a dominant frequency as well as for other frequencies in the spectrum of the finite sinusoidal excitation. However, the sinusoidal waveform is not an appropriate choice if simulation results for a wideband of frequencies are sought.

The frequency spectrum of the source waveform determines the range of frequencies for which valid and accurate results can be obtained. As the frequency increases, the wavelength decreases and becomes comparable to the cell size of the problem space. If the cell size is too large compared with a fraction of a wavelength, the signal at that frequency cannot be sampled accurately in space. Therefore, the highest frequency in the source waveform spectrum should be chosen such that the cell size is not larger than a fraction of the highest-frequency wavelength. For many applications, setting the highest-frequency wavelength larger than 20 cells size is sufficient for a reasonable FDTD simulation. A Gaussian waveform is the best choice for a source waveform, since it can be constructed to contain all frequencies up to a highest frequency that is tied to a cell size by a factor. This factor, which is the proportion of the highest frequency wavelength to the unit cell size, is referred as *number of cells per wavelength* n_c.

A Gaussian waveform can be written as a function of time as

$$g(t) = e^{-\frac{t^2}{\tau^2}}, \tag{5.3}$$

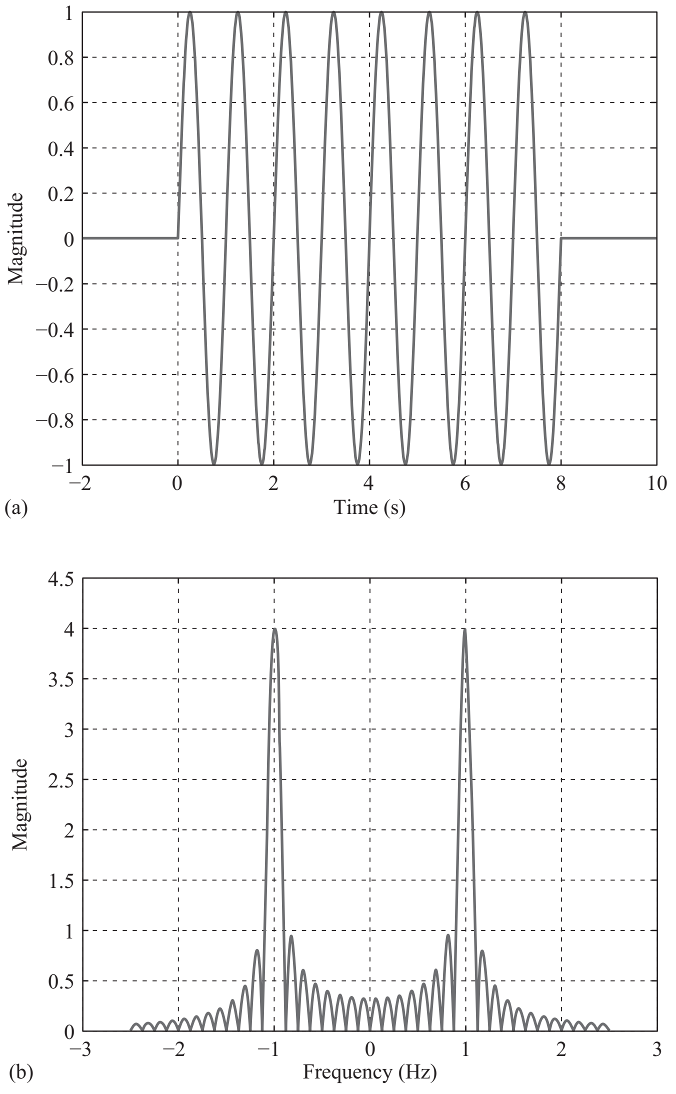


Figure 5.2 A sinusoidal waveform excited for 8 s: (a) $x(t)$ and (b) $X(\omega)$ magnitude.

where τ is a parameter that determines the width of the Gaussian pulse both in the time domain and the frequency domain. The Fourier transform of a Gaussian waveform is also a Gaussian waveform, which can be expressed as a function of frequency as

$$G(w) = \tau\sqrt{\pi}e^{-\frac{\tau^2\omega^2}{4}}. \tag{5.4}$$

The highest frequency that is available out of an FDTD calculation can be determined by the accuracy parameter *number of cells per wavelength* such that

$$f_{\max} = \frac{c}{\lambda_{\min}} = \frac{c}{n_c\Delta s_{\max}}, \tag{5.5}$$

where c is the speed of light in free space, $\Delta s_{\max}$ is the maximum of the cell dimensions (Δx, Δy, or Δz), and $\lambda_{\min}$ is the wavelength of the highest frequency in free space. Once the highest frequency in the spectrum of the frequency domain Gaussian waveform is determined, it is possible to find a τ that constructs the corresponding time-domain Gaussian waveform. Following is a procedure that determines τ, so that one can construct the Gaussian waveform in the time domain before the simulation starts with a good estimate of the highest frequency at which the simulation results will be acceptable.

Consider the Gaussian waveform plotted in Figure 5.3(a) and the magnitude of its Fourier transform pair plotted in Figure 5.3(b) using (5.4). The function (5.4) is real, and its phase

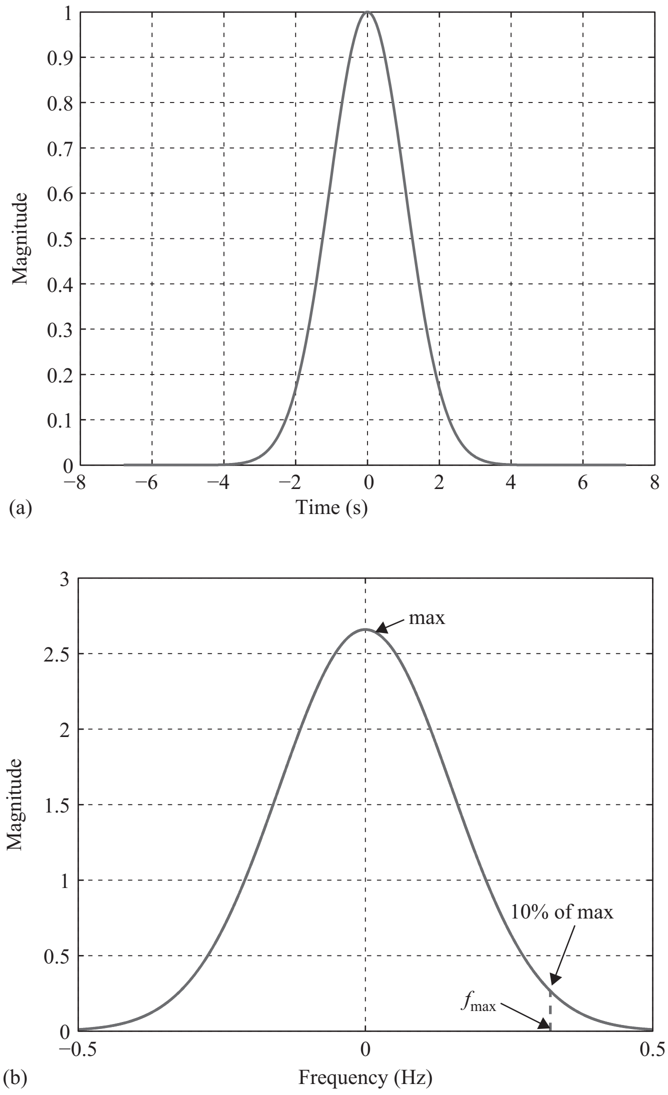


Figure 5.3 A Gaussian waveform and its Fourier transform: (a) $g(t)$ and (b) $G(\omega)$ magnitude.

vanishes; hence, the phase of the Fourier transform is not plotted here. We consider the maximum valid frequency of the Gaussian spectrum as the frequency where the magnitude of $G(\omega)$ is 10% of its maximum magnitude as illustrated in Figure 5.3(b). Therefore, by solving the equation

$$0.1 = e^{-\frac{\tau^2\omega_{\max}^2}{4}},$$

we can find a relation between τ and $f_{\max}$, such that

$$\tau = \frac{\sqrt{2.3}}{\pi f_{\max}}. \tag{5.6}$$

Using (5.5) in (5.6), we can tie n_c and $\Delta s_{\max}$ to τ as

$$\tau = \frac{\sqrt{2.3}n_c\Delta s_{\max}}{\pi c} \cong \frac{n_c\Delta s_{\max}}{2c}. \tag{5.7}$$

Once the parameters n_c and $\Delta s_{\max}$ are defined, the parameter τ can be calculated and can be used in (5.3) to construct the Gaussian waveform spectrum that allows one to obtain a valid frequency response up to $f_{\max}$.

Determination of the parameter τ is not enough to construct the Gaussian waveform that can be directly used in an FDTD simulation. One should notice in Figure 5.3(a) that the Gaussian waveform's maximum coincides with time instant zero. However, in an FDTD simulation the initial conditions of the fields are zero; hence, the sources must also be zero. This can be achieved by shifting the Gaussian waveform in time such that at time instant zero the waveform value is zero or is (practically) negligible. Furthermore, time shifting preserves the frequency content of the waveform.

With time shifting the Gaussian waveform takes the form

$$g(t) = e^{-\frac{(t-t_0)^2}{\tau^2}}, \tag{5.8}$$

where t_0 is the amount of time shift. We can find t_0 as the time instant in (5.3) where $g(t=0) = e^{-20}$, which is a negligible value and is good for more than 16 digits of numerical accuracy. Solving (5.8) with $g(0) = e^{-20}$ we can find

$$t_0 = \sqrt{20}\tau \cong 4.5\tau. \tag{5.9}$$

Using the time shift t_0, we obtain a Gaussian waveform as illustrated in Figure 5.4, which can be used as a source waveform in FDTD simulations to obtain acceptable results for frequencies up to $f_{\max}$.

5.1.3 Normalized derivative of a Gaussian waveform

In some applications it may be desired to excite the problem space with a pulse that has a wide frequency spectrum but does not include zero frequency or very low-frequency

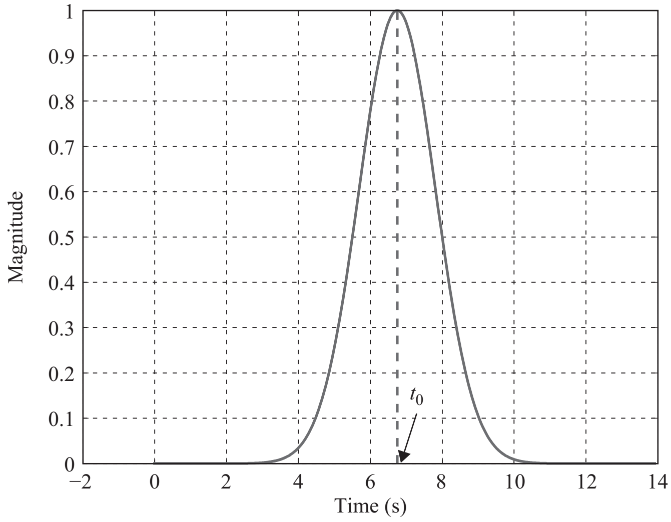


Figure 5.4 Gaussian waveform shifted by $t_0 = 4.5\tau$ in time.

components. Furthermore, if it is not necessary, it is better to avoid the simulation of low-frequency components since simulation of these components need long simulation time. For such cases the derivative of the Gaussian waveform is a suitable choice. The derivative of a Gaussian waveform normalized to the maximum can be found as

$$g(t) = \frac{\sqrt{2e}}{\tau} t e^{-\frac{t^2}{\tau^2}}. \tag{5.10}$$

The Fourier transform of this function can be determined as

$$G(\omega) = \frac{j\omega\tau^2\sqrt{\pi e}}{\sqrt{2}} e^{-\frac{\tau^2\omega^2}{4}}. \tag{5.11}$$

The normalized derivative of a Gaussian waveform and its Fourier transform are illustrated in Figure 5.5(a) and 5.5(b), respectively. One can notice that the function in the frequency domain vanishes at zero frequency. The form of the function suggests that there is a minimum frequency, $f_{\min}$, and a maximum frequency, $f_{\max}$, that determine the bounds of the valid frequency spectrum for an FDTD simulation considering 10% of the maximum as the limit. Having determined the maximum usable frequency from the *number of cells per wavelength* and maximum cell size using (5.5), one can find the parameter τ that sets $G(\omega)$ to 10% of its maximum value at $f_{\max}$ and can use τ in (5.10) to construct a time-domain waveform for the FDTD simulation. However, this procedure requires solution of a nonlinear equation and is not convenient. The equations readily available for a Gaussian waveform can be used instead for convenience. For instance, τ can be calculated using (5.7). Similar to the Gaussian waveform, the derivative of a Gaussian waveform is also centered at time instant zero as it appears in (5.10) and as shown in Figure 5.5(a). Therefore, it also needs a time shift

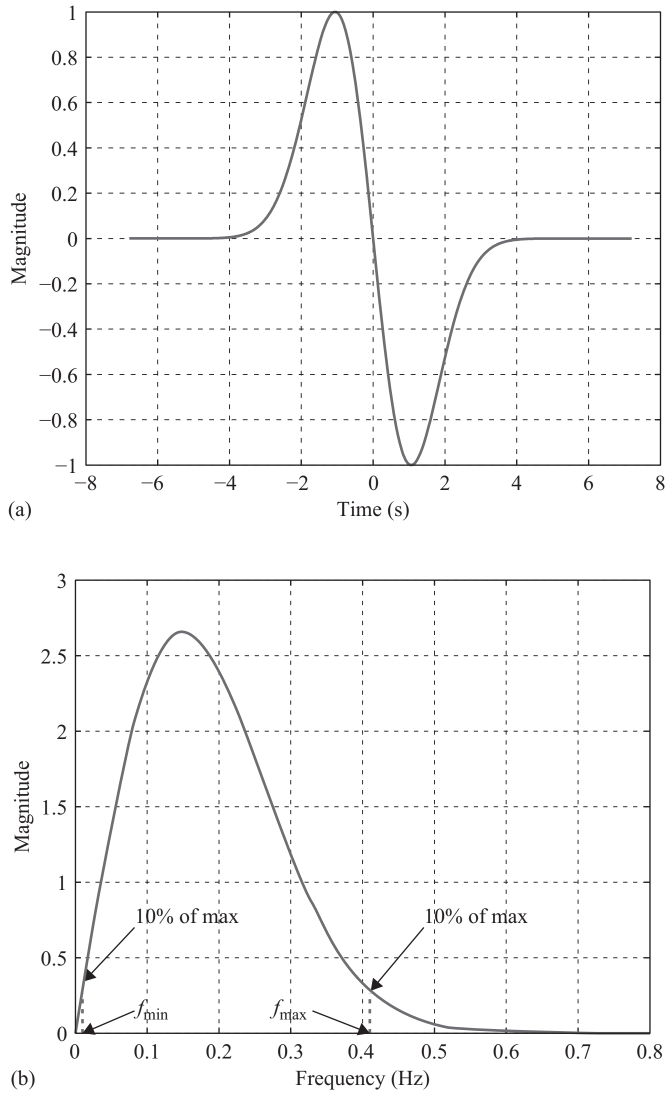


Figure 5.5 Normalized derivative of a Gaussian waveform and its Fourier transform: (a) $g(t)$ and (b) $G(\omega)$ magnitude.

to have practically zero value at time instant zero; hence, it can be written as

$$g(t) = \frac{\sqrt{2e}}{\tau}(t - t_0)e^{-\frac{(t-t_0)^2}{\tau^2}}. \tag{5.12}$$

The time shift value t_0 can be obtained as well using the readily available equation (5.9).

5.1.4 Cosine-modulated Gaussian waveform

Another common waveform used in FDTD applications is the cosine-modulated Gaussian waveform. A Gaussian waveform includes frequency components over a band centered at zero frequency as illustrated in Figure 5.3(b). Modulating a signal with a cosine function having frequency f_c corresponds to shifting the frequency spectrum of the modulated signal by f_c in the frequency domain. Therefore, if it is desired to perform a simulation to obtain results for a frequency band centered at a frequency f_c, the cosine-modulated Gaussian pulse is a suitable choice. One can first construct a Gaussian waveform with a desired bandwidth and then can multiply the Gaussian pulse with a cosine function with frequency f_c. Then the corresponding waveform can be expressed as

$$g(t) = \cos(\omega_c t) e^{-\frac{t^2}{\tau^2}}, \tag{5.13}$$

while its Fourier transform can be written as

$$G(\omega) = \frac{\tau\sqrt{\pi}}{2} e^{-\frac{\tau^2(\omega-\omega_c)^2}{4}} + \frac{\tau\sqrt{\pi}}{2} e^{-\frac{\tau^2(\omega+\omega_c)^2}{4}}, \tag{5.14}$$

where $\omega_c = 2\pi f_c$. A cosine-modulated Gaussian waveform and its Fourier transform are illustrated in Figure 5.6. To construct a cosine-modulated Gaussian waveform with spectrum having Δf bandwidth centered at f_c, first we find τ in (5.14) that satisfies the Δf bandwidth requirement. We can utilize (5.6) to find τ such that

$$\tau = \frac{2\sqrt{2.3}}{\pi \Delta f} \cong \frac{0.966}{\Delta f}. \tag{5.15}$$

Then τ can be used in (5.13) to construct the intended waveform. However, we still need to apply a time shift to the waveform such that initial value of the excitation is zero. We can use (5.9) to find the required time shift value, t_0. Then the final equation for the cosine-modulated Gaussian waveform takes the form

$$g(t) = \cos(\omega_c(t - t_0)) \times e^{-\frac{(t-t_0)^2}{\tau^2}}. \tag{5.16}$$

5.2 Definition and initialization of source waveforms for FDTD simulations

In the previous section, we discussed some common types of source waveforms that can be used to excite an FDTD problem space, and we derived equations for waveform function parameters to construct waveforms for desired frequency spectrum properties. In this section, we discuss the definition and implementation of these source waveforms in an FDTD MATLAB® code.

In Section 4.2.1, we showed that we can define the parameters specific to various types of waveforms in a structure named waveforms. We have shown how a unit step function and a sinusoidal waveform can be defined as subfields of the structure **waveforms**. We can add new

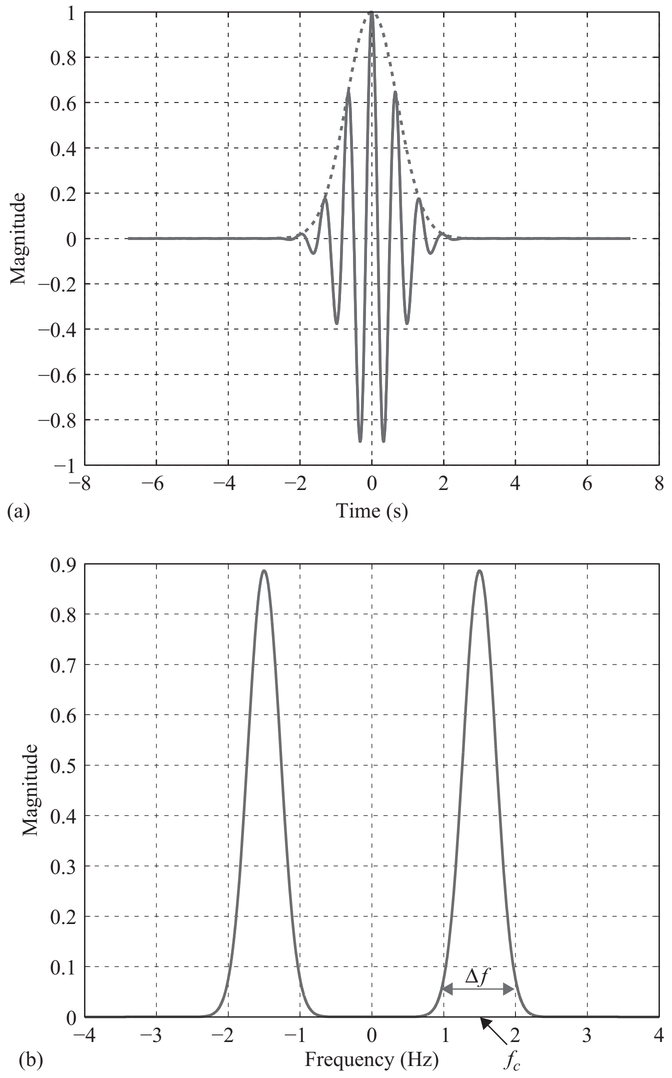


Figure 5.6 Cosine-modulated Gaussian waveform and its Fourier transform: (a) $g(t)$ and (b) $G(\omega)$ magnitude.

subfields to the structure **waveforms** to define the new types of source waveforms. For instance, consider the Listing 5.1, where a sample section of the subroutine ***define_sources_ and_lumped_elements*** illustrates the definition of the source waveforms. In addition to the unit step function and sinusoidal waveform, the Gaussian waveform is defined as the subfield **waveforms.gaussian**, the derivative of a Gaussian waveform is defined as **waveforms.**

derivative_gaussian, and the cosine-modulated Gaussian waveform is defined as **waveforms. cosine_modulated_gaussian**. Each of these subfields is an array, so more than one waveform of the same type can be defined in the same structure **waveforms**. For instance, in this listing two sinusoidal waveforms and two Gaussian waveforms are defined. The Gaussian and the derivative of Gaussian waveforms have the parameter **number_of_cells_per_wavelength**, which is the accuracy parameter n_c described in the previous section used to determine the maximum frequency of the spectrum of the Gaussian waveform. If the **number_ of_cells_per_wavelength** is assigned zero, then the **number_of_cells_per_wavelength** that is defined in the subroutine ***define_problems_space_parameters*** will be used as the default value.

The cosine-modulated Gaussian waveform has the parameter **bandwidth**, which specifies the width of the frequency band Δf. One more parameter is **modulation_frequency**, which is the center frequency of the band, f_c.

Listing 5.1 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;
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;
waveforms.derivative_gaussian(1).number_of_cells_per_wavelength = 20;
waveforms.cosine_modulated_gaussian(1).bandwidth = 1e9;
waveforms.cosine_modulated_gaussian(1).modulation_frequency = 2e9;

% 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 = 'gaussian';
voltage sources(1).waveform index = 2;
```

Once the waveform types are defined, the desired waveform type can be assigned to the sources as illustrated in Listing 5.1. For instance, for the voltage source **voltage_sources(1)** the parameter **waveform_type** is assigned the value "***gaussian***," and the parameter **waveform_index** is assigned the value 2, indicating that the waveform **waveforms.gaussian(2)** is the waveform of the voltage source.

The initialization of the waveforms is performed in the subroutine ***initialize_waveforms***, which is called in ***initialize_sources_and_lumped_elements*** before the initialization of lumped sources as discussed in Section 4.2.3. The implementation of ***initialize_waveforms*** is extended to include the Gaussian, derivative of Gaussian, and cosine-modulated Gaussian waveforms based on the discussions in Section 5 as shown in Listing 5.2. After the waveforms are initialized, the constructed waveforms are copied to appropriate subfields in the source structures in ***initialize_sources_and_lumped_elements***.

Listing 5.2 initialize_waveforms.m

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

% initialize sinusoidal waveforms
if isfield(waveforms,'sinusoidal')
    for ind=1:size(waveforms.sinusoidal,2)
        waveforms.sinusoidal(ind).waveform = ...
            sin(2 * pi * waveforms.sinusoidal(ind).frequency * time);
        waveforms.sinusoidal(ind).t_0 = 0;
    end
end

% initialize unit step waveforms
if isfield(waveforms,'unit_step')
    for ind=1:size(waveforms.unit_step,2)
        start_index = waveforms.unit_step(ind).start_time_step;
        waveforms.unit_step(ind).waveform(1:number_of_time_steps) = 1;
        waveforms.unit_step(ind).waveform(1:start_index-1) = 0;
        waveforms.unit_step(ind).t_0 = 0;
    end
end

% initialize Gaussian waveforms
if isfield(waveforms,'gaussian')
    for ind=1:size(waveforms.gaussian,2)
        if waveforms.gaussian(ind).number_of_cells_per_wavelength == 0
            nc = number_of_cells_per_wavelength;
        else
            nc = waveforms.gaussian(ind).number_of_cells_per_wavelength;
        end
        waveforms.gaussian(ind).maximum_frequency = ...
            c/(nc*max([dx,dy,dz]));
        tau = (nc*max([dx,dy,dz]))/(2*c);
        waveforms.gaussian(ind).tau = tau;
        t_0 = 4.5 * waveforms.gaussian(ind).tau;
```

```
      waveforms.gaussian(ind).t_0 = t_0;
      waveforms.gaussian(ind).waveform = exp(-((time - t_0)/tau).^2);
   end
end

% initialize derivative of Gaussian waveforms
if isfield(waveforms,'derivative_gaussian')
  for ind=1:size(waveforms.derivative_gaussian,2)
  wfrm = waveforms.derivative_gaussian(ind);
  if wfrm.number_of_cells_per_wavelength == 0
    nc = number_of_cells_per_wavelength;
  else
    nc = ...
    waveforms.derivative_gaussian(ind).number_of_cells_per_wavelength;
  end
    waveforms.derivative_gaussian(ind).maximum_frequency = ...
        c/(nc*max([dx,dy,dz]));
    tau = (nc*max([dx,dy,dz]))/(2*c);
    waveforms.derivative_gaussian(ind).tau = tau;
    t_0 = 4.5 * waveforms.derivative_gaussian(ind).tau;
    waveforms.derivative_gaussian(ind).t_0 = t_0;
    waveforms.derivative_gaussian(ind).waveform = ...
        -(sqrt(2*exp(1))/tau)*(time - t_0).*exp(-((time - t_0)/tau).^2);
    end
end

% initialize cosine modulated Gaussian waveforms
if isfield(waveforms,'cosine_modulated_gaussian')
    for ind=1:size(waveforms.cosine_modulated_gaussian,2)
    frequency = ...
    waveforms.cosine_modulated_gaussian(ind).modulation_frequency;
    tau = 0.966/waveforms.cosine_modulated_gaussian(ind).bandwidth;
    waveforms.cosine_modulated_gaussian(ind).tau = tau;
    t_0 = 4.5 * waveforms.cosine_modulated_gaussian(ind).tau;
    waveforms.cosine_modulated_gaussian(ind).t_0 = t_0;
    waveforms.cosine_modulated_gaussian(ind).waveform = ...
    cos(2*pi*frequency*(time - t_0)).*exp(-((time - t_0)/tau).^2);
    end
end
```

5.3 Transformation from time domain to frequency domain

In the previous sections, we discussed construction of different types of source waveforms for an FDTD simulation. These waveforms are functions of time, and any primary output obtained from an FDTD simulation is in the time domain. The input–output relationship is available in the time domain after an FDTD simulation is completed. The input and output time functions can be transformed to the frequency domain by the use of the Fourier transform to obtain the response of the system in the case of time-harmonic excitations. The Fourier transform is an integration applied to a continuous time function. In the FDTD method the values are sampled at discrete time instants; therefore, the continuous time integration can be approximated by a discrete time summation. The Fourier transform of a continuous time function $x(t)$ is given as $X(\omega)$ using (5.1). In FDTD the time function is sampled at a period Δt; therefore, the values $x(n\Delta t)$ are known. Then the Fourier transform (5.1) can be expressed for discrete samples $x(n\Delta t)$ as

$$X(\omega) = \Delta t \sum_{n=1}^{n=N_{steps}} x(n\Delta t) e^{-j\omega n\Delta t}, \tag{5.17}$$

where N_{steps} is the number of time steps. It is easy to implement a function based on (5.17) that calculates the Fourier transform of the discrete samples $x(n\Delta t)$ for a list of frequencies. Implementation of such a function, ***time_to_frequency_domain***, is shown in Listing 5.3. This function accepts a parameter **x**, which is an array of sampled time-domain values of a function. The parameter **frequency_array** includes list of frequencies for which the transform is to be performed. The output parameter of the function **X** is an array including the transform at the respective frequencies.

Another input parameter that needs consideration is **time_shift**. As discussed in previous chapters, the electric field-related values and magnetic field-related values are sampled at different time instants during the FDTD time-marching scheme. There is a half time step duration in between, and the function ***time_to_frequency_domain*** accepts a value **time_shift** to account for this time shift in the Fourier transform. For electric field-related values (e.g., sampled voltages), **time_shift** can take the value 0, whereas for the magnetic field-related values (e.g., sampled currents), **time_shift** can take the value $-$***dt/2***.

The function implementation given in Listing 5.3 is intended for an easy understanding of the time-to-frequency-domain transform action, and is not necessarily the best or optimum algorithm for this purpose. More efficient discrete Fourier transform implementations, such as fast Fourier transforms, can be used as well.

Listing 5.3 time_to_frequency_domain.m

```
function [X] = time_to_frequency_domain(x,dt,frequency_array,time_shift)
% x  : array including the sampled values at discrete time steps
% dt : sampling period, duration of an FDTD time step
% frequency_array : list of frequencies for which transform is performed
% time_shift : a value in order to account for the time shift between
% electric and magnetic fields

number_of_time_steps  = size(x,1);
number_of_frequencies = size(frequency_array,2);
X = zeros(1, number_of_frequencies);
w = 2 * pi * frequency_array;
for n = 1:number_of_time_steps
    t = n * dt + time_shift;
    X = X + x(n) * exp(-j*w*t);
end
X = X * dt;
```

Similarly, the inverse Fourier transform can also be implemented. Consider a band-limited frequency domain function $X(\omega)$ sampled at uniformly distributed discrete frequency points with sampling period $\Delta\omega$, where the sampling frequencies include zero and positive frequencies. Then the inverse Fourier transform (5.2) can be expressed as

$$x(t) = \frac{\Delta\omega}{2\pi}\left(X(0) + \sum_{m=1}^{m=M-1}\left[X(m\Delta\omega)e^{j\omega t} + X^*(m\Delta\omega)e^{-j\omega t}\right]\right), \qquad (5.18)$$

where X^* is the complex conjugate of X, $X(0)$ is the value of X at zero frequency, and M is the number of sampled frequency points. We have assumed that X includes the samples at zero and positive frequencies. It is evident from (5.2) that the integration includes the negative frequencies as well. However, for a causal time function, the Fourier transform at a negative frequency is the complex conjugate of its positive-frequency counterpart. So, if the Fourier transform is known for the positive frequencies, the transform at the negative frequencies can be obtained by the conjugate. Therefore, (5.18) includes summation at the negative frequencies as well. Since the zero frequency appears once, it is added to the discrete summation separately. Then a function named as ***frequency_to_time_domain*** is constructed based on (5.18) as shown in Listing 5.4.

Listing 5.4 frequency_to_time_domain.m

```
function [x] = frequency_to_time_domain(X, df, time_array)
% X  : array including the sampled values at discrete frequency steps
% df : sampling period in frequency domain
% time_array : list of time steps for which
%              inverse transform is performed

number_of_frequencies  = size(X,2);
number_of_time_points = size(time_array,2);
x = zeros(1, number_of_time_points);
dw = 2 * pi * df;
x = X(1); % zero frequency component
for m = 2:number_of_frequencies
    w = (m-1) * dw;
    x = x + X(m)* exp(j*w*time_array)+ conj(X(m)) ...
        * exp(-j*w*time_array);
end
x = x * df;
```

In general, we can define a list of frequencies for which we seek the frequency-domain response in the ***define_output_parameters*** subroutine, which is illustrated in Listing 5.5. Here a structure **frequency_domain** is defined to store the frequency-domain-specific parameters. The subfields **start**, **end**, and **step** correspond to the respective values of a uniformly sampled list of frequencies.

The desired list of frequencies is calculated and assigned to an array frequency_domain. frequencies in the initialize_output_parameters subroutine as shown in Listing 5.6. The initialization of output parameters is implemented in the subroutine **initialize_output_ parameters**, as shown in Listing 5.6.

Listing 5.5 define_output_parameters.m

```
% frequency domain parameters
frequency_domain.start = 1e7;
frequency_domain.end   = 1e9;
frequency_domain.step  = 1e7;
```

Listing 5.6 initialize_output_parameters.m

```
% intialize frequency domain parameters
frequency_domain.frequencies = [frequency_domain.start: ...
    frequency_domain.step:frequency_domain.end];
frequency_domain.number_of_frequencies = ...
    size(frequency_domain.frequencies ,2);
```

Listing 5.7 post_process_and_display_results.m

```
disp('displaying␣simulation␣results');

display_transient_parameters;
calculate_frequency_domain_outputs;
display_frequency_domain_outputs;
```

The post processing and display of the simulation results are performed in the subroutine ***post_process_and_display_results*** following ***run_fdtd_time_marching_loop*** in **fdtd_solve**. We can add two new subroutines to ***post_process_and_display_results*** as illustrated in Listing 5.7: ***calculate_frequency_domain_outputs*** and ***display_frequency_domain_outputs***. The time-to-frequency-domain transformations can be performed in ***calculate_frequency_domain_outputs***, for which the sample code is shown in Listing 5.8. In this subroutine the time-domain arrays of sampled values are transformed to frequency domain using the function **time_to_frequency_domain**, and the calculated frequency-domain arrays are assigned to the **frequency_domain_value** subfield of the respective structures. Finally, the subroutine ***display_frequency_domain_outputs***, a partial section of which is listed in Listing 5.9, can be called to display the frequency-domain outputs.

5.4 Simulation examples

So far we have discussed some source waveform types that can be used to excite an FDTD problem space. We considered the relationship between the time- and frequency-domain representations of the waveforms and provided equations that can be used to construct a temporal waveform having specific frequency spectrum characteristics. Then we provided functions that perform time-to-frequency-domain transformation, which can be used to obtain simulation results in the frequency domain after an FDTD simulation is completed. In this section we provide examples that utilize source waveforms and transformation functions.

Listing 5.8 calculate_frequency_domain_outputs.m

```
disp('generating␣frequency␣domain␣outputs');

frequency_array = frequency_domain.frequencies;

% sampled electric fields in frequency domain
for ind=1:number_of_sampled_electric_fields
    x = sampled_electric_fields(ind).sampled_value;
    time_shift = 0;
    [X] = time_to_frequency_domain(x, dt, frequency_array, time_shift);
    sampled_electric_fields(ind).frequency_domain_value = X;
    sampled_electric_fields(ind).frequencies = frequency_array;
end

% sampled currents in frequency domain
for ind=1:number_of_sampled_currents
    x = sampled_currents(ind).sampled_value;
    time_shift = -dt/2;
    [X] = time_to_frequency_domain(x, dt, frequency_array, time_shift);
    sampled_currents(ind).frequency_domain_value = X;
    sampled_currents(ind).frequencies = frequency_array;
end

% voltage sources in frequency domain
for ind=1:number_of_voltage_sources
    x = voltage_sources(ind).waveform;
    time_shift = 0;
    [X] = time_to_frequency_domain(x, dt, frequency_array, time_shift);
    voltage_sources(ind).frequency_domain_value = X;
    voltage_sources(ind).frequencies = frequency_array;
end
```

Listing 5.9 display_frequency_domain_outputs.m

```
% figures for sampled voltages
for ind=1:number_of_sampled_voltages
    frequencies = sampled_voltages(ind).frequencies*1e-9;
    fd_value = sampled_voltages(ind).frequency_domain_value;
    figure;
    title(['sampled␣voltage␣[' num2str(ind) ']'],'fontsize',12);
    subplot(2,1,1);
    plot(frequencies, abs(fd_value),'b-','linewidth',1.5);
    xlabel('frequency␣(GHz)','fontsize',12);
    ylabel('magnitude','fontsize',12);
    grid on;
    subplot(2,1,2);
    plot(frequencies, angle(fd_value)*180/pi,'r-','linewidth',1.5);
    xlabel('frequency␣(GHz)','fontsize',12);
    ylabel('phase␣(degrees)','fontsize',12);
    grid on;
    drawnow;
end
```

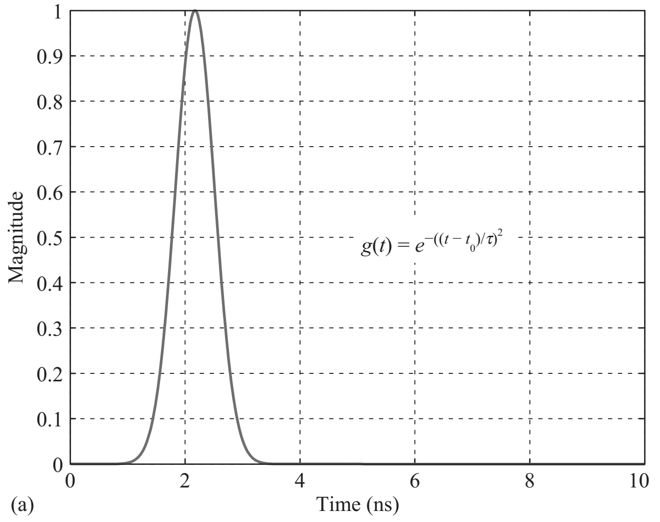
5.4.1 Recovering a time waveform from its Fourier transform

The first example illustrates how the functions ***time_to_frequency_domain*** and ***frequency_to_time_domain*** can be used. Consider the program ***recover_a_time_waveform*** given in Listing 5.10.

Listing 5.10 recover_a_time_waveform.m

```
clc; close all; clear all;
% Construct a Gaussian Waveform in time, frequency spectrum of which
% has its magnitude at 1 GHz as 10% of the maximum.
maximum_frequency = 1e9;
tau = sqrt(2.3)/(pi*maximum_frequency);
t_0 = 4.5 * tau;
time_array = [1:1000]*1e-11;
g = exp(-((time_array - t_0)/tau).^2);
figure(1);
plot(time_array*1e9, g,'b-','linewidth',1.5);
title('g(t)=e^{-((t-t_0)/\tau)^2}','fontsize',14);
xlabel('time␣(ns)','fontsize',12);
ylabel('magnitude','fontsize',12);
set(gca,'fontsize',12);
grid on;

% Perform time to frequency domain transform
frequency_array = [0:1000]*2e6;
dt = time_array(2)-time_array(1);
G = time_to_frequency_domain(g, dt, frequency_array, 0);

figure(2);
subplot(2,1,1);
plot(frequency_array*1e-9, abs(G),'b-','linewidth',1.5);
title('G(\omega)␣=␣F(g(t))','fontsize',12);
xlabel('frequency␣(GHz)','fontsize',12);
ylabel('magnitude','fontsize',12);
set(gca,'fontsize',12);
grid on;
subplot(2,1,2);
plot(frequency_array*1e-9, angle(G)*180/pi,'r-','linewidth',1.5);
xlabel('frequency␣(GHz)','fontsize',12);
ylabel('phase␣(degrees)','fontsize',12);
set(gca,'fontsize',12);
grid on;
drawnow;

% Perform frequency to time domain transform
df = frequency_array(2)-frequency_array(1);
g2 = frequency_to_time_domain(G, df, time_array);

figure(3);
plot(time_array*1e9, abs(g2),'b-','linewidth',1.5);
title('g(t)=␣F^{-1}(G(\omega))','fontsize',14);
xlabel('time␣(ns)','fontsize',12);
ylabel('magnitude','fontsize',12);
set(gca,'fontsize',12);
grid on;
```

First, a Gaussian waveform, $g(t)$, for which the frequency spectrum is 1 GHz wide, is constructed and is time shifted as shown in Figure 5.7(a). The Fourier transform of $g(t)$ is obtained as $G(\omega)$, as plotted in Figure 5.7(b), using ***time_to_frequency_domain***. One should notice that the value of $G(\omega)$ is 10% of the maximum at 1 GHz. Furthermore, the phase vanishes for the Fourier transform of a time-domain Gaussian waveform having its center at time instant zero. However, the phase shown in Figure 5.7(b) is nonzero. This is due to the time shift introduced to the Gaussian waveform in the time domain. Finally, the time-domain Gaussian waveform is reconstructed from $G(\omega)$ using the function ***frequency_to_time_domain*** and is plotted in Figure 5.7(c).



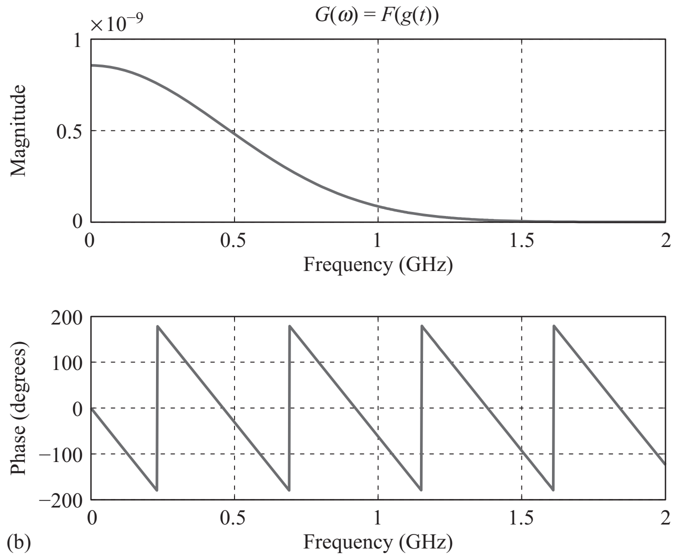


Figure 5.7 A Gaussian waveform and its Fourier transform: (a) $g(t)$; (b) $G(\omega)$; and (c) $g(t)$ recovered from $G(\omega)$.

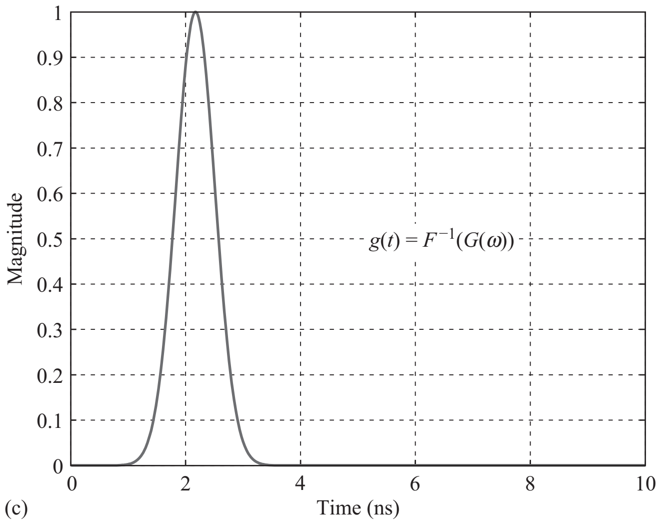


Figure 5.7 (*Continued*)

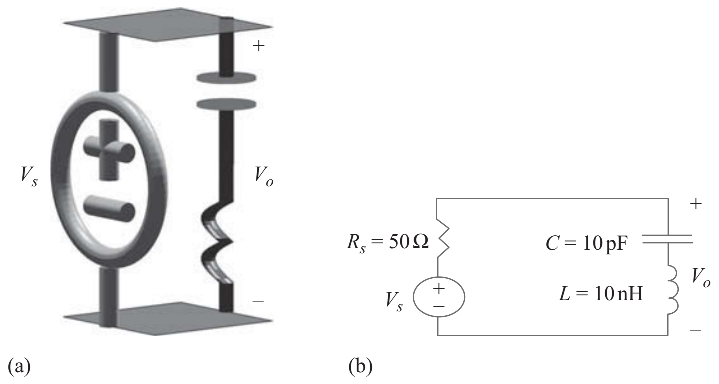


Figure 5.8 An RLC circuit: (a) circuit simulated in FDTD and (b) lumped element equivalent circuit.

5.4.2 An RLC circuit excited by a cosine-modulated Gaussian waveform

The second example is a simulation of a 10 nH inductor and a 10 pF capacitor connected in series and excited by a voltage source with 50 Ω internal resistor. The geometry of the problem is shown in Figure 5.8(a) as it is simulated by the FDTD program. The equivalent circuit representation of the problem is illustrated in Figure 5.8(b). The size of a unit cell is set to 1 mm on each side. The number of time steps to run the simulation is 2000. Two parallel PEC plates are defined 2 mm apart from each other as shown in Listing 5.11. A voltage source is placed in between these plates at one end, and an inductor and a capacitor are placed in series at the other end as shown in Listing 5.12.

Listing 5.11 define_geometry.m

```
disp('defining the problem geometry');

bricks  = [];
spheres = [];

% define a PEC plate
bricks(1).min_x = 0;
bricks(1).min_y = 0;
bricks(1).min_z = 0;
bricks(1).max_x = 1e-3;
bricks(1).max_y = 1e-3;
bricks(1).max_z = 0;
bricks(1).material_type = 2;

% define a PEC plate
bricks(2).min_x = 0;
bricks(2).min_y = 0;
bricks(2).min_z = 2e-3;
bricks(2).max_x = 1e-3;
bricks(2).max_y = 1e-3;
bricks(2).max_z = 2e-3;
bricks(2).material_type = 2;
```

Listing 5.12 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;
waveforms.gaussian(1).number_of_cells_per_wavelength = 0;
waveforms.gaussian(2).number_of_cells_per_wavelength = 15;
waveforms.derivative_gaussian(1).number_of_cells_per_wavelength = 20;
waveforms.cosine_modulated_gaussian(1).bandwidth = 4e9;
waveforms.cosine_modulated_gaussian(1).modulation_frequency = 2e9;

```

```
% 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 = 0;
voltage_sources(1).max_y = 1.0e-3;
voltage_sources(1).max_z = 2.0e-3;
voltage_sources(1).direction = 'zp';
voltage_sources(1).resistance = 50;
voltage_sources(1).magnitude = 1;
voltage_sources(1).waveform_type = 'cosine_modulated_gaussian';
voltage_sources(1).waveform_index = 1;

% inductors
% direction: 'x', 'y', or 'z'
% inductance : henrys
inductors(1).min_x = 1.0e-3;
inductors(1).min_y = 0.0;
inductors(1).min_z = 0.0;
inductors(1).max_x = 1.0e-3;
inductors(1).max_y = 1.0e-3;
inductors(1).max_z = 1.0e-3;
inductors(1).direction = 'z';
inductors(1).inductance = 10e-9;

% capacitors
% direction: 'x', 'y', or 'z'
% capacitance : farads
capacitors(1).min_x = 1.0e-3;
capacitors(1).min_y = 0.0;
capacitors(1).min_z = 1.0e-3;
capacitors(1).max_x = 1.0e-3;
capacitors(1).max_y = 1.0e-3;
capacitors(1).max_z = 2.0e-3;
capacitors(1).direction = 'z';
capacitors(1).capacitance = 10e-12;
```

A cosine-modulated Gaussian waveform with 2 GHz center frequency is assigned to the voltage source. A sampled voltage is defined between the parallel PEC plates at the load end as shown in Listing 5.13.

Figure 5.9 shows the time-domain results of this simulation; V_o is compared with the source waveform, V_s. The frequency-domain counterparts of these waveforms are calculated using time_to_frequency_domain and are plotted in Figure 5.10. We can find the transfer function of this circuit by normalizing the output voltage V_o to V_s. Furthermore, we can perform a circuit analysis to find the transfer function of the equivalent circuit in Figure 5.8(b), which yields

$$T(\omega) = \frac{V_o(\omega)}{V_s(\omega)} = \frac{s^2LC + 1}{s^2LC + sRC + 1}, \tag{5.19}$$

Listing 5.13 define_output_parameters.m

```
disp('defining output parameters');

sampled_electric_fields = [];
sampled_magnetic_fields = [];
sampled_voltages = [];
sampled_currents = [];

% figure refresh rate
plotting_step = 10;

% mode of operation
run_simulation = true;
show_material_mesh = true;
show_problem_space = true;

% frequency domain parameters
frequency_domain.start = 2e7;
frequency_domain.end   = 4e9;
frequency_domain.step  = 2e7;

% define sampled voltages
sampled_voltages(1).min_x = 1.0e-3;
sampled_voltages(1).min_y = 0.0;
sampled_voltages(1).min_z = 0.0;
sampled_voltages(1).max_x = 1.0e-3;
sampled_voltages(1).max_y = 1.0e-3;
sampled_voltages(1).max_z = 2.0e-3;
sampled_voltages(1).direction = 'zp';
sampled_voltages(1).display_plot = true;
```

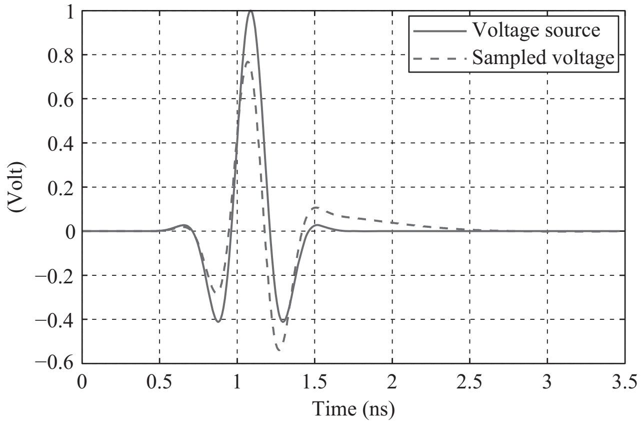
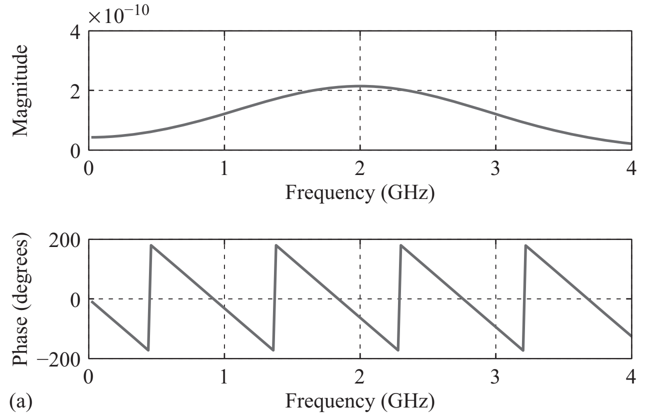


Figure 5.9 Time-domain response, V_o, compared with source waveform, V_s.



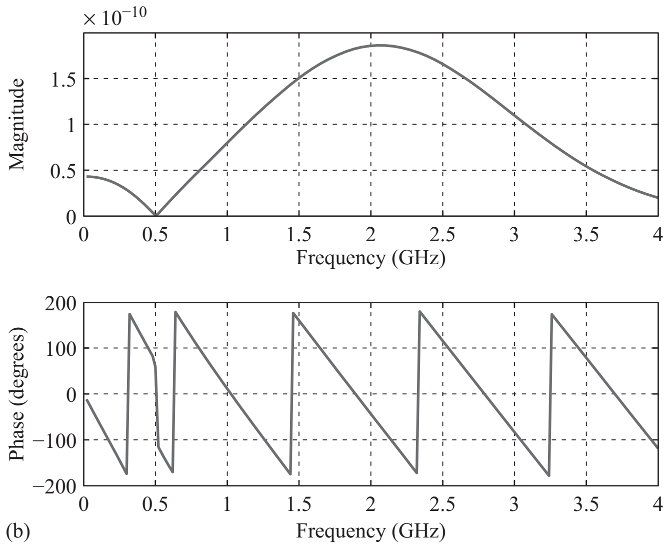


Figure 5.10 Frequency-domain response of source waveform V_s and output voltage V_o: (a) Fourier transform of V_s and (b) Fourier transform of V_o.

where $s = j\omega$. The transfer function obtained from the FDTD simulation and the exact equation (5.19) are compared as shown in Figure 5.11. These two results agree with each other very well at low frequencies; however, there is a deviation at high frequencies. One of the reasons for the difference is that the two circuits in Figure 5.8(a) and 5.8(b) are not exactly the same. For instance, the parallel plates introduce a capacitance, and the overall circuit is a loop and introduces an inductance in Figure 5.8(a). However, these additional effects are not accounted for in the lumped circuit in Figure 5.8(b). Therefore, an electromagnetic simulation including lumped element components should account for the additional effects due to the physical structure of the circuit to obtain more accurate data at higher frequencies.

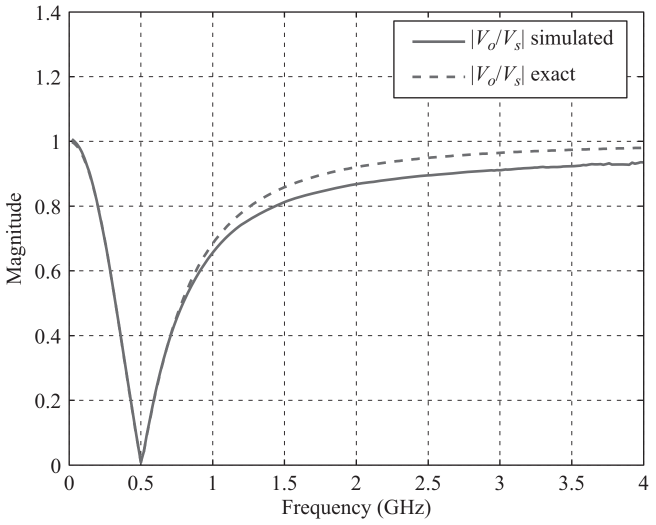


Figure 5.11 Transfer function $T(\omega) = \frac{V_o(\omega)}{V_s(\omega)}$.

5.5 Exercises

5.1 Consider the stripline structure that you constructed in Exercise 4.2. Define a Gaussian source waveform, assign it to the voltage source and run the FDTD simulation. By the time the simulation is completed, the sampled voltage and current will be captured, transformed to frequency domain, and stored in the **frequency_domain_value** field of the parameters **sampled_voltages(i)** and **sampled_currents(i)**. Input impedance of this stripline can be calculated using these sampled voltage and current arrays in the frequency domain. Since the stripline characteristic impedance is expected to be 50 Ω and the stripline is terminated by a 50 Ω resistor, the input impedance shall be 50 Ω as well. Calculate and plot the input impedance for a wide frequency band, and verify that the input impedance is close to 50 Ω.

5.2 Consider the simulation you performed in Exercise 5.1. When you examine the sampled transient voltage you will observe a Gaussian waveform generated by the voltage source followed by some reflections due to the imperfect termination of the 50 Ω resistor. If your stripline structure is long enough and the Gaussian waveform is narrow enough, the Gaussian waveform and reflections should not overlap. Determine from the transient voltage a number of time steps that would be used to run the FDTD simulations such that the simulation would end before the reflections are captured and only the Gaussian waveform would be observed as the sampled transient voltage. This way you would be simulating an infinitely long stripline structure. Therefore, when you calculate the input impedance, you would actually be calculating the characteristic impedance of the stripline. Rerun the FDTD simulation with the time step you have determined as discussed, calculate and plot the input impedance for a wide frequency band, and verify that the characteristic impedance of the stripline is 50 Ω.

5.3 Consider the circuit in Exercise 4.4. Set the waveform of the voltage source as the Gaussian waveform, and define a sampled current to capture the current flowing through the circuit. Rerun the FDTD simulation, calculate and plot the input impedance for a wide frequency band, and verify that the circuit resonates at 100 MHz. Note that you may need to run the simulation for a large number of time steps since it will take a long time for the low-frequency components in the Gaussian waveform to decay.

5.4 Consider Exercise 5.3. Now set the waveform of the voltage-source derivative of the Gaussian waveform. Rerun the FDTD simulation, calculate and plot the input impedance for a wide frequency band, and verify that the circuit resonates at 100 MHz. Note that with the derivative of the Gaussian waveform you should be able to obtain the results with a much smaller number of time steps compared with the amount of steps used in Exercise 5.3 since the derivative of the Gaussian waveform does not include zero frequency and since very low-frequency components are negligible.