SSY200

Computational Electromagnetics

Assignment 2, 2022

Assignment 3, 2021

Finite-difference Time-Domain Scheme

Anna-Maria Unterberger (CID: annunt)

Friday 25th February, 2022

CHALMERS
UNIVERSITY OF TECHNOLOGY

Problem Description

In this assignment a rectangular waveguide of fixed dimensions is modeled and evaluated using the FDTD scheme. The input signal is a Gaussian pulse containing a range of frequencies and we obtain the transmission and reflection coefficients of the system in frequency domain.

Solution

Numerical Implementation

The finite-difference time-domain (FDTD) method of solving differential equations involves finite-difference approximate evaluations of both spatial and time derivatives. Its major strength is that it allows explicit time-stepping, meaning that computationally expensive systems of equations are avoided. Its two major weaknesses are that it is unsuited to dealing with boundaries that do not conform to the underlying Cartesian grid, and that there is an inherent limit on the time-step size $\Delta t < h/c\sqrt{3}$, making it practically useful only for problems involving characteristic lengths on the order of a wavelength. This makes it particularly suited to microwave problems.

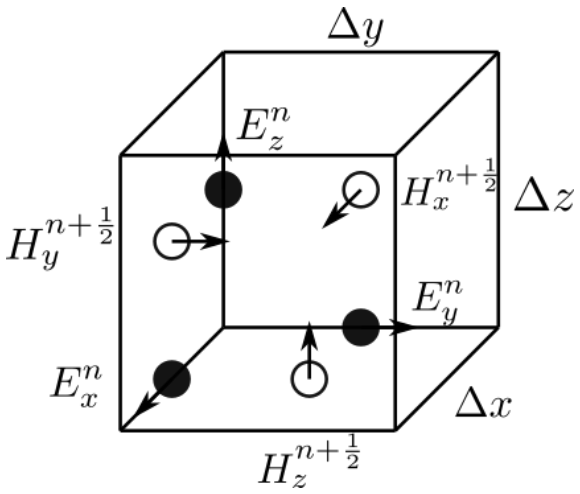

Figure 1: Typical Yee-cell. $\vec{E}$ on edges, $\vec{H}$ on faces.
Source: Zohar0729, CC BY-SA 4.0 `https://creativecommons.org/licenses/by-sa/4.0`, via Wikimedia Commons

In this task, the waveguide interior is discretised using a 3D grid of Yee-cells. A Yee-cell is a construct used to dicretise 3D space into two staggered grids, spaced a half grid-space apart. This is particularly suited to solving the coupled first-order differential form of Maxwell's equations.

For first-order derivatives, derivatives taken across a single cell, with the derivative value on the half-grid, has a smaller leading error than that of the derivative taken across two cells with the derivative value all on a single grid. This is the major advantage of staggered grids. Another issue affecting single grid discretisation in this case is that the two field variables have the potential to uncouple under certain conditions, making it unsuited.

To generalise these staggered grids to three dimensions a so-called Yee-cell is used, a cuboid (usually with sides of equal length) with one variable's vector components centered on its edges and the other variable's vector components centered on its faces, in this case the electric and magnetic fields, respectively. The center of the edges is taken to be on the full grid, and the center of the faces is thus on the half-grid.

The differential equations that govern the electromagnetic wave, namely *Ampère's law* and *Fara-*

day's law are as follows (in differential form)

$$\nabla \times \vec{H} = \vec{J}_v + \varepsilon \frac{\partial \vec{E}}{\partial t} \tag{1}$$

$$\nabla \times \vec{E} = -\mu \frac{\partial \vec{H}}{\partial t} \tag{2}$$

It is assumed here that volume current density $\vec{J}_v = \vec{0}$.

The above equations are rearranged to solve for $\partial\vec{E}$ and $\partial\vec{H}$, and cumulatively summed in each time-step, essentially integrating, to give $\vec{E}$ and $\vec{H}$.

The expression for the curl of a vector field in Cartesian coordinates is given by

$$\nabla \times \vec{F} = \begin{bmatrix} \frac{\partial F_z}{\partial y} - \frac{\partial F_y}{\partial z} \\ \frac{\partial F_x}{\partial z} - \frac{\partial F_z}{\partial x} \\ \frac{\partial F_y}{\partial x} - \frac{\partial F_x}{\partial y} \end{bmatrix} \tag{3}$$

Thus, the above expressions can be rearranged as

$$\partial\vec{E} = \frac{\partial t}{\varepsilon} \begin{bmatrix} \frac{\partial H_z}{\partial y} - \frac{\partial H_y}{\partial z} \\ \frac{\partial H_x}{\partial z} - \frac{\partial H_z}{\partial x} \\ \frac{\partial H_y}{\partial x} - \frac{\partial H_x}{\partial y} \end{bmatrix} \tag{4}$$

$$\partial\vec{H} = -\frac{\partial t}{\mu} \begin{bmatrix} \frac{\partial E_z}{\partial y} - \frac{\partial E_y}{\partial z} \\ \frac{\partial E_x}{\partial z} - \frac{\partial E_z}{\partial x} \\ \frac{\partial E_y}{\partial x} - \frac{\partial E_x}{\partial y} \end{bmatrix} = \frac{\partial t}{\mu} \begin{bmatrix} \frac{\partial E_y}{\partial z} - \frac{\partial E_z}{\partial y} \\ \frac{\partial E_z}{\partial x} - \frac{\partial E_x}{\partial z} \\ \frac{\partial E_x}{\partial y} - \frac{\partial E_y}{\partial x} \end{bmatrix} \tag{5}$$

Differentiation is carried out using the MATLAB `diff()` function, which, for a vector of length N, returns a vector of length N-1 with the difference between adjacent elements i.e. $[V(2) - V(1), V(3) - V(2), ..., V(N) - V(N-1)]$. The behaviour is similar for matrices: a dimension to differentiate along is specified and each vector along that dimension is processed as before.

As an example, $\frac{\partial E_y}{\partial z}$ is processed using `diff(Ey, 1, 3)`, where E_y is a three dimensional matrix representing a scalar field (a single component of a vector field), the second argument is the order of differentiation and the third argument is the dimension along which to differentiate.

A matrix of $N \times N \times N$ elements, representing the scalar field will return an $N \times N \times N-1$ matrix in this case, which must be taken into consideration when setting the dimensions of the field matrices.

The discretised versions of equations 4 and 5 are as follows:

$$\epsilon \frac{E_x|^{n+1}_{p+\frac{1}{2},q,r} - E_x|^{n}_{p+\frac{1}{2},q,r}}{\Delta t} \tag{6}$$

$$= \frac{H_z|_{p+\frac{1}{2},q+\frac{1}{2},r} - H_z|_{p+\frac{1}{2},q-\frac{1}{2},r}}{\Delta y} - \frac{H_y|^{n+\frac{1}{2}}_{p+\frac{1}{2},q,r+\frac{1}{2}} - H_y|^{n+\frac{1}{2}}_{p+\frac{1}{2},q,r-\frac{1}{2}}}{\Delta z}$$

$$\implies E_x|^{n+1}_{p+\frac{1}{2},q,r} = E_x|^{n}_{p+\frac{1}{2},q,r} + \frac{\Delta t}{\epsilon} \left[\frac{H_z|_{p+\frac{1}{2},q+\frac{1}{2},r} - H_z|_{p+\frac{1}{2},q-\frac{1}{2},r}}{\Delta y} - \frac{H_y|^{n+\frac{1}{2}}_{p+\frac{1}{2},q,r+\frac{1}{2}} - H_y|^{n+\frac{1}{2}}_{p+\frac{1}{2},q,r-\frac{1}{2}}}{\Delta z} \right]$$

E_y and E_z are similarly discretised.

For the magnetic field the discretisation is similar, now expressed for H_x at half time-steps.

$$\mu \frac{H_x|^{n+\frac{1}{2}}_{p,q+\frac{1}{2},r+\frac{1}{2}} - H_x|^{n-\frac{1}{2}}_{p,q+\frac{1}{2},r+\frac{1}{2}}}{\Delta t} \tag{7}$$
$$= \frac{E_y|^{n}_{p,q+\frac{1}{2},r+1} - E_y|^{n}_{p,q+\frac{1}{2},r}}{\Delta z} - \frac{E_z|^{n}_{p,q+1,r+\frac{1}{2}} - E_z|^{n}_{p,q,r+\frac{1}{2}}}{\Delta y}$$
$$\implies H_x|^{n+\frac{1}{2}}_{p,q+\frac{1}{2},r+\frac{1}{2}} = H_x|^{n-\frac{1}{2}}_{p,q+\frac{1}{2},r+\frac{1}{2}} + \frac{\Delta t}{\mu}\left[\frac{E_y|^{n}_{p,q+\frac{1}{2},r+1} - E_y|^{n}_{p,q+\frac{1}{2},r}}{\Delta z} - \frac{E_z|^{n}_{p,q+1,r+\frac{1}{2}} - E_z|^{n}_{p,q,r+\frac{1}{2}}}{\Delta y}\right]$$

H_y and H_z are, again, similarly discretised.

These equations can then be directly expressed in Matlab using the `diff()` command, as previously discussed:

```
Dx = h;
Dy = h;
Dz = h;

Hx = Hx + (Dt/mu0) * (diff(Ey,1,3)/Dz - diff(Ez,1,2)/Dy);
Hy = Hy + (Dt/mu0) * (diff(Ez,1,1)/Dx - diff(Ex,1,3)/Dz);
Hz = Hz + (Dt/mu0) * (diff(Ex,1,2)/Dy - diff(Ey,1,1)/Dx);

Ex(: ,2:Ny, 2:Nz) = Ex(: ,2:Ny, 2:Nz) + (Dt/eps0) * ...
                    (diff(Hz(:,:,2:Nz),1,2)/Dy - diff(Hy(:,2:Ny,:),1,3)/Dz);
Ey(2:Nx ,: ,2:Nz) = Ey(2:Nx ,: ,2:Nz) + (Dt/eps0) * ...
                    (diff(Hx(2:Nx,:,:),1,3)/Dz - diff(Hz(:,:,2:Nz),1,1)/Dx);
Ez(2:Nx, 2:Ny, :) = Ez(2:Nx, 2:Ny, :) + (Dt/eps0) * ...
                    (diff(Hy(:,2:Ny,:),1,1)/Dx - diff(Hx(2:Nx,:,:),1,2)/Dy);
```

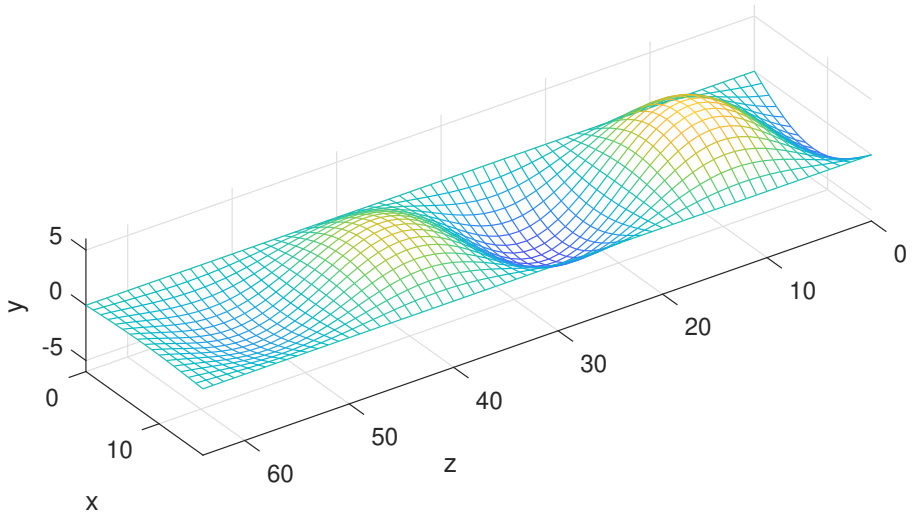

Figure 2: E_y for TE_{10} in the empty waveguide as the mode propagates. The half-wave transverse pattern indicates that only TE_{10} is propagating, as expected.

Numerical Tests

For an empty wave guide, we do not expect there to be any reflections or attenuation of the propagating modes.

The TE_{10} mode has the lowest cutoff frequency (this is generally true for waveguides with dimen-

sions $L_x > L_y$) and is given by the equation:

$$f_{c_{10}} = \frac{k_{c_{10}}}{2\pi\sqrt{\mu\varepsilon}} = \frac{1}{2\sqrt{\mu\varepsilon}}\sqrt{(\frac{m}{L_x})^2 + (\frac{n}{L_y})^2} = 3,7474\,\text{GHz} \tag{8}$$

where $k_{c_{10}}$ is the cutoff wave number and m,n are the mode indices.

The second lowest cutoff is for mode TE_{01}, which using the equation above has a cutoff frequency of

$$f_{c_{01}} = 6,6621\,\text{GHz}$$

There are no corresponding TM modes for these indices.

For a Gaussian pulse input containing energy between 3.9-6.5 GHz, the entire spectrum is contained in the TE_{10} mode and not in any other mode. Since TE_{10} is the only propagating mode, the entire pulse is propagated.

This indeed appears to be the case based on results in the time-domain in fig. 3. We observe a small amount of dispersion in s_2, since the gaussian pulse is no longer symmetrical. This is due to the frequency dependent velocity of the signal (as seen in eq. 12) as well as numerical dispersion inherent in the FDTD method.

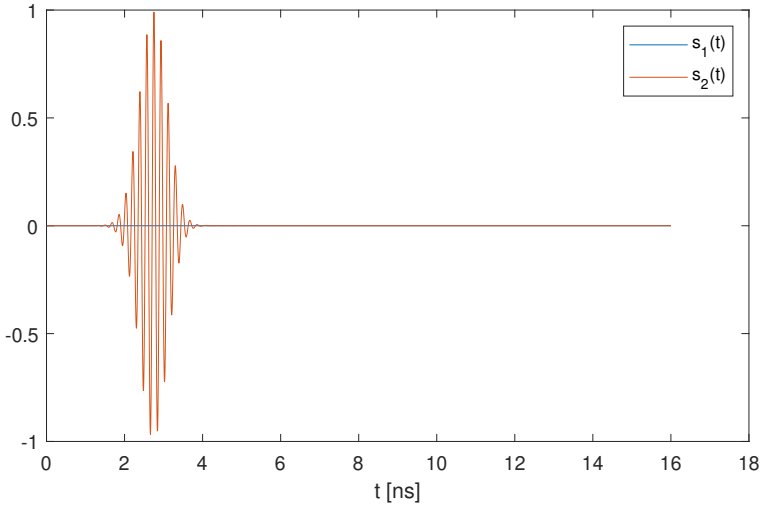

Figure 3: Amplitude of E_y in time domain at z = 0 and z = Lz, respectively. The incident pulse is omitted from s_1, showing only the reflected signal.

The time-domain signals at each port are then transformed to frequency-domain in order to calculate the reflection and transmission coefficients as a function of frequency. This is done using MATLAB's `fft(x)` and `fftshift(X)` functions. The implementation is done using a separate function for the fft procedure: `function X = scaled_shifted_fft(x)`. The scatter parameters are calculated as follows:

```
S1in   = scaled_shifted_fft(s);
S1out  = scaled_shifted_fft(s1R(:,1));
S2out  = scaled_shifted_fft(s2T(:,1));

S11 = S1out./S1in;
S21 = S2out./S1in;
```

```
function X = scaled_shifted_fft(x)

Y = fft(x);
L = length(x);
```

```
P2 = abs(Y./L);      % Normalise amplitude by sample length. Take absolute value.
P1 = fftshift(P2,1); % fft(x) returns vector with *positive* freq components first,
                     % then negative components. fftshift(X) switches the order.

X = P1;
```

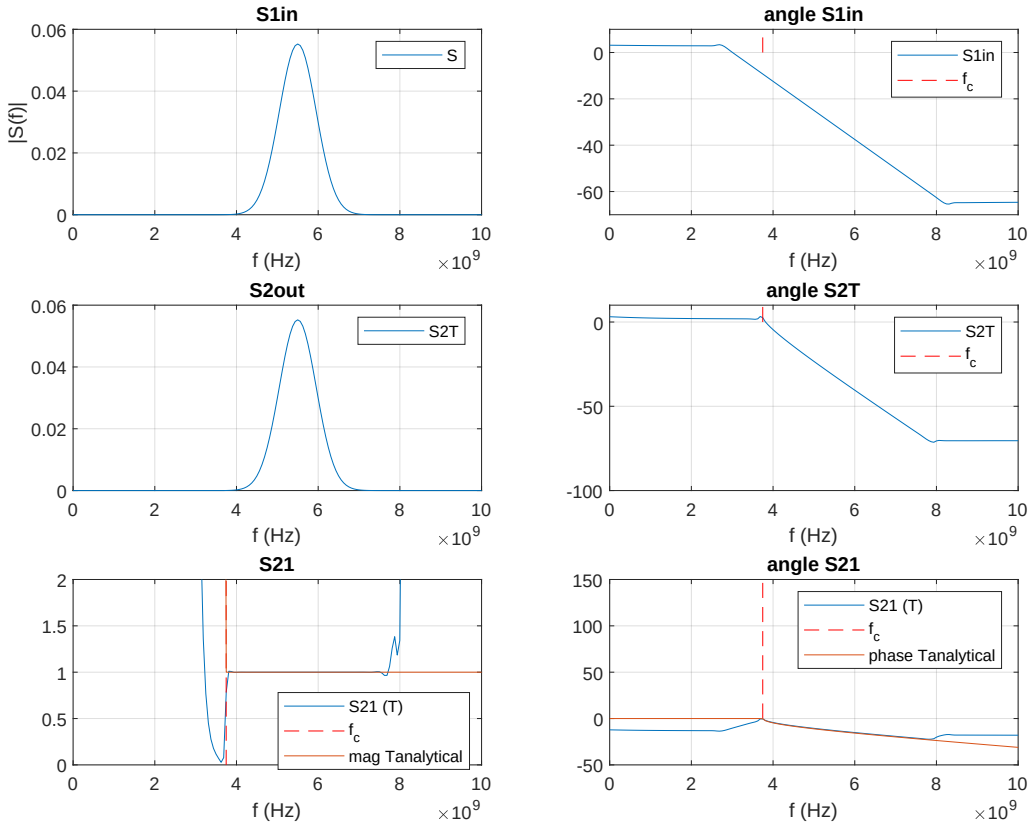

Figure 4: Frequency-domain plots of the input signal and transmitted output signal, as well as the transmission coefficient S21. The reflection coefficient S11 has negligible SNR and is thus omitted. The phases of S21 and the analytical value of T have been normalized at f_c, since only the relative phase above f_c is interesting.

This is what is expected for the magnitude plots, as R = 0 and T = 1 over the frequency range of interest: that of the input signal. The exponential nature of the plot for S21 outside this range of interest is due to the low amount of energy at these frequencies. Calculations at these points are thus subject to numerical noise.

To confirm the results for S21, we can compare it to the analytical frequency function, given by the following formulae.

First, we require the cutoff wavenumber for TE_{10} k_{c10}, the wavenumber k and the propagation constant β.

$$k_{c10} = \sqrt{\left(\frac{m\pi}{L_x}\right)^2 + \left(\frac{n\pi}{L_y}\right)^2} \tag{9}$$

$$k = \omega\sqrt{\mu\varepsilon} \tag{10}$$

$$\beta = \sqrt{k^2 - k_{c10}^2} \tag{11}$$

T is then simply

$$T = e^{-j\beta L_z} \tag{12}$$

We see from the graphs in fig. 4 that whereas the original signal (S1in) has a linear phase change over a wide frequency range, starting at a point before the cutoff frequency of the waveguide, the output signal (S2out) has this linear phase shift slightly truncated, starting at the cutoff frequency. This is expected, as lower frequency components should not propagate and can be regarded as noise. S2out is also shifted significantly more, though in a nearly linear fashion, indicating dispersion. Putting this together explains why the phase of the transmission coefficient (S21) rises slightly before the cutoff frequency, and why its phase decreases with frequency after the cutoff frequency, showing the phase shift caused by dispersion. The computed phase follows the theoretical phase precisely apart from the initial increase, since energy in the input signal below the cutoff frequency is not accounted for in the theoretical expression.

Appendix

FdtdLab.m:

```
%%%%%%%%%%%%%%%%%%
% Problem Setup %
%%%%%%%%%%%%%%%%%%
clear
close all

% Physical constants
eps0 = 8.8541878e-12;
mu0  = 4e-7 * pi;
c0 = 299792458;

% Cell size
h = 0.0025;

% Waveguide dimensions
Lx = 0.040;
Ly = 0.0225;
Lz = 0.160;

% Number of cells in each direction
Nx = round(Lx / h);
Ny = round(Ly / h);
Nz = round(Lz / h);

% Length of time steps
Dt = h / (c0 * sqrt(3));

% Insignal data
t_max = 16e-9;
Nt    = ceil(t_max / Dt);
t     = (1:Nt)' * Dt;

f_min = 4e9;
f_max = 7e9;
f_mid = (f_max + f_min) / 2;
BWr   = (f_max - f_mid) / f_mid;
f     = ((0:Nt-1)'-floor(Nt/2)) / Nt / Dt;
[s, sq, se]     = gauspuls(t-0.2e-8, f_mid, BWr, -12);

% % Square wave
% s=square(1e9*pi.*t-0.2e-8)
%  s(1:100) = 0;
%  s(500:end) = 0;
% % Sinc
%  s = zeros(length(t),1);
%  s(1:501) = sinc((-250:250)/(1*2*pi));
% plot(s)


% Allocate field matrices
Ex = zeros(Nx,     Ny + 1, Nz + 1);
Ey = zeros(Nx + 1, Ny,     Nz + 1);
Ez = zeros(Nx + 1, Ny + 1, Nz    );
```

```
Hx = zeros(Nx + 1, Ny,     Nz    );
Hy = zeros(Nx    , Ny + 1, Nz    );
Hz = zeros(Nx    , Ny,     Nz + 1);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initiation of boundary conditions %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp(sprintf('Initiate boundary conditions...'))
NumModesTE = 7; % 01 10 11 20 21 30 31
NumModesTM = 3; % 11 21 31
NumModes   = NumModesTE + NumModesTM;

disp(sprintf('  Compute TE modes'))
[ExTE, EyTE, K2TE] = ComputeTEModes(NumModesTE, Nx, Ny, h, h);
disp(sprintf('  Compute TM modes'))
[ExTM, EyTM, K2TM] = ComputeTMModes(NumModesTM, Nx, Ny, h, h);

ModalEx = [ExTE ExTM];
ModalEy = [EyTE EyTM];
ModalK2 = [K2TE K2TM];
ModalNm = sum(ModalEx.^2) + sum(ModalEy.^2); % Normalizing constants

clear ExTE EyTE ExTM EyTM

% Compute Impulse response for the propagating mode
IR   = zeros(Nt, NumModes);  % Impulse response
s1R  = zeros(Nt, NumModes);  % Reflected signal at z = Dz
s1   = zeros(Nt, NumModes);  % Total signal at z = 0
s2T  = zeros(Nt, NumModes);  % Transmitted signal at z = Lz - Dz
s2   = zeros(Nt, NumModes);  % Total signal at z = Lz

for k = 1:NumModes
  disp(sprintf('  Computing Impulse response for Mode %d', k))
  IR(:,k) = ComputeIR(Dt, h, Nt, ModalK2(k));
end

%%%%%%%%%%%%%%%
% Main Loop. %
%%%%%%%%%%%%%%%
disp(sprintf('Start time stepping...'))
% Set initial source boundary conditions
sEy = Ey(2:Nx, :, 2);
sEx = Ex(:, 2:Ny, 2);
sEy(:) = s(1) * ModalEy(:,1);
sEx(:) = s(1) * ModalEx(:,1);
Ey(2:Nx, :, 1) = sEy;
Ex(:, 2:Ny, 1) = sEx;
s1(1,1) = s(1);

CH = Dt / (h * mu0);
CE = Dt / (h * eps0);

ks = 400;
for k = 2:Nt
```

```
  if k > 250 && k < 600
    figure(99)
    mesh(0:Nz, 0:Nx, squeeze(0.5*Ey(:,6,:))), axis equal, axis([0 Nz 0 Nx -6 6])
    xlabel('z');ylabel('x');zlabel('y');
    caxis([-6 6]), view(145,30)
    drawnow
  end

% % Export pretty image.
% if (k == 450)
%     figure
%     mesh(0:Nz, 0:Nx, squeeze(0.5*Ey(:,6,:))), axis equal, axis([0 Nz 0 Nx -6 6])
%     xlabel('z');ylabel('x');zlabel('y');
%     caxis([-6 6]), view(145,30)
%     export_fig('TE10-Ey.pdf', '-painters', '-transparent')
%     pause()
% end

  %==================%
  % FDTD update loops %
  %==================%

  % ------------------------------------------------------------
  % | ADD FDTD UPDATE LOOPS                                    |
  % ------------------------------------------------------------  |
  %                                                            |  |
  %                                                           \    /
  %                                                            \  /
  %                                                             \/
  R = c0*Dt/h; % Numerical Dispersion parameter ~0.58 by default. 1 is magic. >1 diverges.

%     Ex = zeros(Nx,     Ny + 1, Nz + 1);
%     Ey = zeros(Nx + 1, Ny,     Nz + 1);
%     Ez = zeros(Nx + 1, Ny + 1, Nz    );
%
%     Hx = zeros(Nx + 1, Ny,     Nz    );
%     Hy = zeros(Nx    , Ny + 1, Nz    );
%     Hz = zeros(Nx    , Ny,     Nz + 1);


% %% Update Hx
%   for i = 1:Nx+1
%     for j = 1:Ny
%         for k = 1:Nz
%             Hx(i,j,k) = Hx(i,j,k) + CH*...
%                 ((Ey(i,j,k+1)-Ey(i,j,k)) - (Ez(i,j+1,k)-Ez(i,j,k)));
%         end
%     end
%   end
%
%     %% Update Hy
%   for i = 1:Nx
%     for j = 1:Ny+1
%         for k = 1:Nz
%             Hy(i,j,k) = Hy(i,j,k) + CH*...
%                 ((Ez(i+1,j,k)-Ez(i,j,k)) - (Ex(i,j,k+1)-Ex(i,j,k)));
```

```
%         end
%     end
%   end
%
%     %% Update Hz
%   for i = 1:Nx
%     for j = 1:Ny
%         for k = 1:Nz+1
%             Hz(i,j,k) = Hz(i,j,k) + CH*...
%                 (Ex(i,j+1,k)-Ex(i,j,k)) - (Ey(i+1,j,k)-Ey(i,j,k));
%         end
%     end
%   end
%
%
%   %% E-field
%   %% Update Ex except on boundary
%   for i = 1:Nx
%       for j = 2:Ny
%           for k = 2:Nz
%             Ex(i,j,k) = Ex(i,j,k) + CE*...
%                 (Hz(i,j,k)-Hz(i,j-1,k)) - (Hy(i,j,k)-Hy(i,j,k-1));
%           end
%       end
%   end
%
%   %% Update Ey
%     for i = 2:Nx
%       for j = 1:Ny
%           for k = 2:Nz
%             Ey(i,j,k) = Ey(i,j,k) + CE*...
%                 (Hx(i,j,k)-Hx(i,j,k-1)) - (Hz(i,j,k)-Hz(i-1,j,k));
%           end
%       end
%     end
%
%     %% Update Ez
%     for i = 2:Nx
%       for j = 2:Ny
%           for k = 1:Nz
%             Ez(i,j,k) = Ez(i,j,k) + CE*...
%                 (Hy(i, j, k)-Hy(i-1 ,j, k)) - (Hx(i, j, k)-Hx(i, j-1, k));
%           end
%       end
%   end

  %% Based on Rylander pg.76.
  Dx = h;
  Dy = h;
  Dz = h;

  Hx = Hx + (Dt/mu0) * (diff(Ey,1,3)/Dz - diff(Ez,1,2)/Dy);
  Hy = Hy + (Dt/mu0) * (diff(Ez,1,1)/Dx - diff(Ex,1,3)/Dz);
  Hz = Hz + (Dt/mu0) * (diff(Ex,1,2)/Dy - diff(Ey,1,1)/Dx);

  Ex(: ,2:Ny, 2:Nz) = Ex(: ,2:Ny, 2:Nz) + (Dt/eps0) * (diff(Hz(:,:,2:Nz),1,2)/Dy - diff(Hy(:,2:Ny,:)
```

```
  Ey(2:Nx ,: ,2:Nz) = Ey(2:Nx ,: ,2:Nz) + (Dt/eps0) * (diff(Hx(2:Nx,:,:),1,3)/Dz - diff(Hz(:,:,2:Nz)
  Ez(2:Nx, 2:Ny, :) = Ez(2:Nx, 2:Ny, :) + (Dt/eps0) * (diff(Hy(:,2:Ny,:),1,1)/Dx - diff(Hx(2:Nx,:,:)


  %                                                          /\
  %                                                         /  \
  %                                                        /    \
  %                                                         |  |
  % ---------------------------------------------------------  |
  % |                                                          |
  % ------------------------------------------------------------


  %=========================%
  % Metal Boundary Conditions %
  %=========================%

  % ------------------------------------------------------------
  % | ADD METAL OBJECTS FOR FILTERING                           |
  % ---------------------------------------------------------  |
  %                                                         |  |
  %                                                        \    /
  %                                                         \  /
  %                                                          \/
  %% Narrow midsection

  % ------------    -------------
  %            |----|
  %            |----|              x
  % ------------    -------------  |
  %                                 -z

%   % Set E and H to 0 within PEC walls.
%   Ex(1:2, :, 25:39) = zeros(2,10,15);
%  Ex(end-1:end, :, 25:39) = zeros(2,10,15);
%
%   Ey(1:2, :, 25:39) = zeros(2,9,15);
%  Ey(end-1:end, :, 25:39) = zeros(2,9,15);
%
%  Ez(1:2, :, 25:39) = zeros(2,10,15);
%  Ez(end-1:end, :, 25:39) = zeros(2,10,15);
%
%    Hx(1:2, :, 25:39) = zeros(2,9,15);
%  Hx(end-1:end, :, 25:39) = zeros(2,9,15);
%
%   Hy(1:2, :, 25:39) = zeros(2,10,15);
%  Hy(end-1:end, :, 25:39) = zeros(2,10,15);
%
%  Hz(1:2, :, 25:39) = zeros(2,9,15);
%  Hz(end-1:end, :, 25:39) = zeros(2,9,15);


  %                                                          /\
  %                                                         /  \
  %                                                        /    \
```

```
%                                                          |  |
% ---------------------------------------------------------  |
% |                                                          |
% ------------------------------------------------------------
```

```
%==============================%
% Boundary Conditions at Z = 0 %
%==============================%

% Extract transverse fields at z = dz %
sEy = Ey(2:Nx, :, 2);
sEx = Ex(:, 2:Ny, 2);

% Compute modal voltages %
s1R(k,:) = (sEy(:)' * ModalEy + sEx(:)' * ModalEx) ./ ModalNm;

% The reflected modal amplitude at z = Dz is the difference between the
% total modal amplitude and that of the incoming wave. The amplitude of
% the incoming wave at z = Dz is a convolution of the insignal at z = 0
% and the impulse response of the wave guide.
s1R(k,1) = s1R(k,1) - s(1:k-1)' * IR(k-1:-1:1,1);

% Port Amplitude: sum of insignal and convolution of the reflected
% signal at z = Dz with the impulse response
for l = 1:NumModes
  s1(k,l) = s1R(1:k-1, l)' * IR(k-1:-1:1, l);
end
s1(k,1) = s1(k,1) + s(k);

% Set port 1  boundary conditions %
sEy(:) = ModalEy * s1(k,:)';
sEx(:) = ModalEx * s1(k,:)';
Ey(2:Nx,:,1) = sEy;
Ex(:,2:Ny,1) = sEx;

%==============================%
% Boundary Conditions at Z = Lz %
%==============================%

% Extract transverse fields at z = Lz - Dz %
sEy = Ey(2:Nx, :, Nz);
sEx = Ex(:, 2:Ny, Nz);

% Compute modal voltage at z = Dz
s2T(k,:) = (sEy(:)' * ModalEy + sEx(:)' * ModalEx) ./ ModalNm;

% Port Amplitude
for l = 1:NumModes
  s2(k,l) = s2T(1:k-1, l)' * IR(k-1:-1:1, l);
end

% Set port 2  boundary conditions %
sEy(:) = ModalEy * s2(k,:)';
sEx(:) = ModalEx * s2(k,:)';
```

```
  Ey(2:Nx,:,Nz+1) = sEy;
  Ex(:,2:Ny,Nz+1) = sEx;

  if(mod(k,100) == 0)
    disp(sprintf('  step %5d of %5d', k, Nt))
  end

end;

% remove the incoming sigland from the total signal at port 1.
s1(:,1) = s1(:,1) - s; % s1out?

figure(6)
T = t*1e9;
plot(T,s1(:,1),T,s2(:,1))
legend('s_1(t)','s_2(t)')
xlabel('t [ns]');

% ------------------------------------------------------------
% | ADD SCATTERING PARAMETERS AS FUNCTION OF FREQUENCY        |
% ------------------------------------------------------------  |
%                                                            |  |
%                                                           \    /
%                                                            \  /
f = f(f>=0);

S1in   = one_sided_scaled_fft(s);
S1out  = one_sided_scaled_fft(s1R(:,1));
S2out  = one_sided_scaled_fft(s2T(:,1));

S11 = S1out./S1in;
S21 = S2out./S1in;

% Plot transmission coefficient T(w)
figure
plot(f, S21)
grid on
title('T(\omega)')
xlim([3e9 10e9]);
ylim([0 2]);
xlabel('f');
ylabel('|T(\omega)|');
hold on

% Plot S2out frequency domain
figure
plot(f, unwrap(angle(S1in)))
hold on
plot(f, unwrap(angle(S2out)))
legend('S1in', 'S2out')
xlim([0 10e9])
grid on


% Plot Analytical Transmission coefficient T(w).
omega = 2*pi.*f;
```

```
k = omega.*sqrt(mu0*eps0);
kc = sqrt((1*pi/Lx)^2 + (0*pi/Ly)^2);
beta = sqrt(k.^2 - kc.^2);
Tanalytical = exp(-1i*beta*Lz);
plot(f, abs(Tanalytical))
hold on
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0)); % Cutoff frequency
plot([fcTE10 fcTE10], [-2 10], '--')
ylim([-3 10])
xlim([0 10e9])
legend('T(\omega) Numerical', 'T(\omega) Analytical')
grid on

%% Subplots
figure(2)
% Input signal: s
subplot(421)
plot(f,abs(S1in))
title('S1in')
xlim([0 10e9])
xlabel('f (Hz)')
ylabel('|S(f)|')
legend('S')
grid on
hold on

% S1in phase
subplot(422)
plot(f,unwrap(angle(S1in)))
title('angle S1in')
xlabel('f (Hz)')
xlim([0 10e9])
ylim([-70 10])
grid on
hold on
% TE10 cutoff frequency (Cheng. eq:10-163)
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0))
plot([fcTE10 fcTE10], [0 150], '--r')
legend('S1in', 'f_c')

% Transmitted signal: s2T
subplot(423)
plot(f,abs(S2out))
title('S2out')
xlabel('f (Hz)')
xlim([0 10e9])
legend('S2T')
grid on
hold on

% S1in phase
subplot(424)
plot(f,unwrap(angle(S2out)))
title('angle S2T')
xlabel('f (Hz)')
xlim([0 10e9])
```

```
ylim([-100 10])
grid on
hold on
% TE10 cutoff frequency (Cheng. eq:10-163)
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0))
plot([fcTE10 fcTE10], [0 150], '--r')
legend('S2T', 'f_c')

% S11
subplot(425)
plot(f,abs(S11))
title('S11')
xlabel('f (Hz)')
legend('S11 (R)')
xlim([0 10e9])
ylim([0 2])
grid on
hold on

% S11 phase
subplot(426)
plot(f,unwrap(angle(S11)))
title('angle S11')
xlabel('f (Hz)')
xlim([0 10e9])
ylim([-20 20])
grid on
hold on
% TE10 cutoff frequency (Cheng. eq:10-163)
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0))
plot([fcTE10 fcTE10], [0 150], '--r')
legend('S11 (T)', 'f_c')


% S21 mag
subplot(427)
plot(f,abs(S21))
title('S21')
xlabel('f (Hz)')
xlim([0 10e9])
ylim([0 2])

grid on
hold on
% TE10 cutoff frequency (Cheng. eq:10-163)
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0))
plot([fcTE10 fcTE10], [0 150], '--r')
plot(f,abs(Tanalytical))
legend('S21 (T)', 'f_c','mag Tanalytical')

% S21 phase
subplot(428)
plot(f,unwrap(angle(S21)))
title('angle S21')
xlabel('f (Hz)')
xlim([0 10e9])
```

```
%ylim([-20 20])
grid on
hold on
% TE10 cutoff frequency (Cheng. eq:10-163)
fcTE10 = 1/(2*Lx*sqrt(mu0*eps0));
plot([fcTE10 fcTE10], [0 150], '--r')
plot(f,unwrap(angle(Tanalytical)))
legend('S21 (T)', 'f_c','phase Tanalytical')

% figure
% plot(f,unwrap(angle(Tanalytical))-345)
% xlim([0 10e9])

%% Misc
Modalfc = sqrt(ModalK2)./(2*pi*sqrt(mu0*eps0));


% for blepz = 1:6
% % Analytical resonant frequency (Rylander eq5.21)
% %f_TE10 = c0/2*( (1/Lx)^2 + (0/Ly)^2 + (blepz/Lz)^2 )^(1/2)
% plot([Modalfc(blepz) Modalfc(blepz)], [0 max(P1)], '--')
% end

%% Plot K vs omega
figure(4)
plot((fcTE10:1000:8e9), sqrt((2*pi*(fcTE10:1000:8e9)).^2*mu0*eps0 - (pi/Lx)^2))
hold on
plot((fcTE10:1000:8e9), sqrt((2*pi*(fcTE10:1000:8e9))./c0))
xlabel('f');ylabel('k_z');
grid on

R = c0*Dt/Dz;

figure
hold on
grid on
k = omega./c0
for R = [1 0.8 c0*Dt/Dz 0.1]
    x = k.*Dz;
    y = R.*sin(x./2);

    plot( x,2*asin(y)./Dt.*Dz/c0)
    ylim([0 pi])
    xlim([0 pi])
end

%                                                          /\
%                                                         /  \
%                                                        /    \
%                                                        |  |
% ----------------------------------------------------   |
% |                                                      |
% -------------------------------------------------------
```

one_side_scaled_fft.m:

```
function X = one_sided_scaled_fft(x)
% FFT with correct amplitude and only positive frequency range.

Y = fft(x);
L = length(x);
P2 = (Y./L);      % Normalize amplitude by sample length.
%P1 = fftshift(P2,1); % fft(x) returns vector with *positive* freq components first, then negative c
P1 = P2(1:L/2+1);         % Extract positive frequency components.
P1(2:end-1) = 2*P1(2:end-1); % Normalize one-sided spectrum.
X = P1(1:end-1); % Truncate last element in order to be consistent with f vector.
```