Contents

Preamble

This code listing is generated using Matlab's publishing function. The code is in the exact same order as the plaintext .m file. The main bulk of the program (constructing the matrix and solving for the E-field) is contained in a function compute_R_and_T() at the bottom of the file.

```
% Assignment 1
close all
clear all
```

Constants

```
global mu0 c0 e0;
mu0 = 4e-7*pi;
c0 = 299792458;
e0 = 1/(mu0*c0^2);
```

Input params

```
N = 4;
a = 2e-2;       % Glass thickness (=2cm)
omega = 3*10^9;
E0 = 1;         % amplitude of incident field E^i_z
er = 2.5;       % relative permittivity
sigma = 0.02;   % conductivity
grid_select = "G1";
```

Analytical values

```
k0 = omega/c0;
k1ana = sqrt(er*k0^2 - j*omega*mu0*sigma);
Dana = ((k0 + k1ana)^2)*exp(j*4*a*k1ana) - (k0 - k1ana)^2;
Rana = ((k0^2 - k1ana^2)/Dana)*exp(j*2*a*k0)*(exp(j*4*a*k1ana) - 1); %
Tana = ((k0*k1ana)/Dana)*4*exp(j*2*a*(k0 + k1ana));
```

Compute R & T for four values of h

```
Rvec = [];
Tvec = [];
hvec = [];

for idx = 1:4
    % N^idx as an argument ensures a geometric series for h.
    [R, T] = compute_R_and_T(N^idx, a, omega, E0, er, sigma, grid_select);

    Rvec(idx) = R;
    Tvec(idx) = T;
    hvec(idx) = 2*a/N^idx;
end

% Absolute error values
Rerr = Rana - Rvec;
Terr = Tana - Tvec;
```

Order of Convergence

We obtain two values for order of convergence (alpha) using two geometric sequences. This is just to increase confidence in our result.

```
alpha_R_vec = [0,0];
alpha_T_vec = [0,0];

for idx = 1:2
    alpha_R_vec(idx) = log((Rvec(idx) - Rvec(idx+1))./(Rvec(idx+1) - Rvec(idx+2)))./log(hvec(idx)/hvec(idx+1));
    alpha_T_vec(idx) = log((Tvec(idx) - Tvec(idx+1))./(Tvec(idx+1) - Tvec(idx+2)))./log(hvec(idx)/hvec(idx+1));
end

alpha_R = alpha_R_vec(1);
alpha_T = alpha_T_vec(1);
```

Extrapolate to zero cell size

Least-squares approximation

```
% R
A = [ones(length(hvec), 1), (hvec.^(alpha_R)).'];
b = Rvec.';
x = A\b;
extrapolated_R = x(1);

% T
A = [ones(length(hvec), 1), (hvec.^(alpha_T)).'];
b = Tvec.';
x = A\b;
extrapolated_T = x(1);
```

compute_R_and_T

This function is called from the main body of the program above.

```
function [R, T] = compute_R_and_T(N, a, omega, E0, er, sigma, grid_select)
    % Solve E-field and compute R and T for a given grid and resolution
    % Inputs:
    %   N: grid size specifier. Must be even integer >=4
    %   a: glass thickness
    %   omega: wave frequency
    %   E0: complex magnitude of incident wave
    %   er: relative permittivity of glass
    %   sigma: conductivity of glass
    %   grid_select: select grid G1 or G2

    % Initial Parameters
    global mu0 c0 e0;
    k0 = omega/c0;

    % Construct grid
    if (grid_select == "G1")
        % G1 (ie. grid 1) material interfaces fall between grid points
        n = -N-1:1:N;
        dx = 2*a/N;
        xn = (n + 0.5)*dx;
        Ngp = length(xn);
    elseif (grid_select == "G2")
        % G2 - material interfaces fall on grid points
        n = -N:1:N;
        dx = 2*a/N;
        xn = n*dx;
        Ngp = length(xn);
    else
        assert(false, "Use G1 or G2 as grid type")
    end

    % Set up system of equations

    % Set up material properties (e & sigma) for each spatial grid point
    sigma_vec = zeros(Ngp, 1);
    e_vec = e0*ones(Ngp, 1);  % e(x) = er(x)*e0

    glass_indices = find(xn > -a & xn < a);
    sigma_vec(glass_indices) = sigma;
    e_vec(glass_indices) = er * e_vec(glass_indices);

    % Fill the interior
    A = -1/dx^2*spdiags([1, -2, 1],-1:1,Ngp,Ngp); % generic FD tridiagonal
    % Add material-dependent terms to the main diagonal
    for idx = 1:Ngp
        A(idx, idx) = A(idx, idx) + mu0*(j*omega*sigma_vec(idx) - omega^2*e_vec(idx));
    end

    % Left boundary condition (z1-z0)/h - j*k0*(z1+z0)/2 = -2*j*k*Ez((x0+x1)/2)
    A(1, 1:2) = [-1/dx-j*k0/2, 1/dx-j*k0/2];

    % Right boundary condition (zN-zN-1)/h + jk0(zN+zN-1)/2
    A(Ngp, Ngp-1:Ngp) = [-1/dx+j*k0/2, 1/dx+j*k0/2];

    % Right hand side
    b = zeros(Ngp, 1);
    b(1) = -2*j*k0*E0*exp(-j*k0*(0.5*(xn(1) + xn(2)))); % Left bound, evaluated at x1, but really lies on half-grid(??). We obviously don't have a half grid, s

    Etot = A\b;
    Etot = Etot.'; % Nonconjugate transpose!

    % Compute R & T
    % We obtain values for R and T for every grid point.
    % We choose to keep the value that is nearest (but not on) the scatterer.

    % Reflection coeff
    Rnum = ((Etot - E0*exp(-j*k0*xn)).*exp(-j*k0*xn))/E0;
    Rnum = Rnum(find(xn < -a));
    Rnum = Rnum(end);

    % Transmission coeff
    Tnum = (Etot.*exp(j*k0*xn))/E0;
```

```
    Tnum = Tnum(find(xn > a));
    Tnum = Tnum(1);

    R = Rnum;
    T = Tnum;
end
```

Published with MATLAB® R2025b