EE6506 - Computational Electromagnetics
Homework - 4

Srivenkat A (EE18B038), Srijan Gupta (EP17B009)

May 7, 2021

1 Question 1

1.1 Introduction

The aim of this exercise is to implement the 1-D Finite Element Method to calculate the fields scattered by a series of layers of a specific material with air gaps in between. We use a material with refractive index 3.5 and have the widths of the material and air gaps following the golden ratio $\tau = (\sqrt{5}+1)/2$. The material is assumed to be arranged in 2 specific sequences. In the first sequence, the material layer alternates with the air layer while in the second sequence, the layers follow a Fibonacci series of arrangement.

1.2 Transfer Matrix Formulation

At every individual interface, the reflection and transmission coefficients can be found out using Fresnel Equations. The problem gets amplified in the multilayer case as the fields start interfering with each other. So, in the multilayer problem, We first aim to analytically solve for the field distribution using the Transfer Matrix Formulation approach. Here, we include the boundary conditions enforced upon the field at the every interface and the phase delay encountered when the field passes through the layer and model it in the form of a cascaded series of matrices.

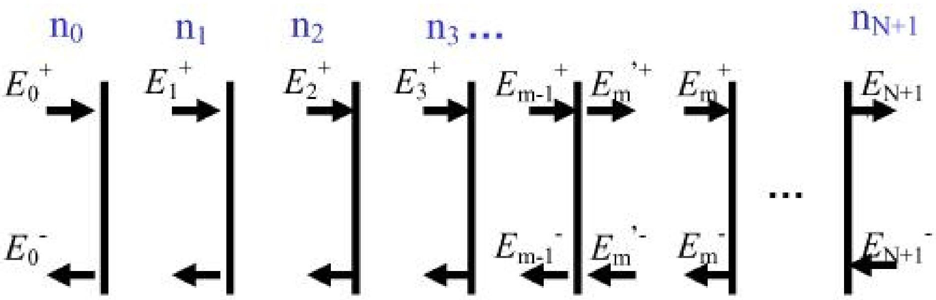


Figure 1: Solving multilayer system using TMM
Source : Reference 1

The amplitudes at the left and right side of an interface are modelled by a Fresnel coefficient matrix as:

$$\begin{pmatrix} E_{m-1}^+ \\ E_{m-1}^- \end{pmatrix} = I_{m-1,m} \begin{pmatrix} E_m^+ \\ E_m^- \end{pmatrix}; \quad I_{m-1,m} = \frac{1}{t_{m-1,m}} \begin{pmatrix} 1 & r_{m-1,m} \\ r_{m-1,m} & 1 \end{pmatrix} \tag{1}$$

Where, $t_{m-1,m}$ and $r_{m-1,m}$ are the Fresnel transmission and reflection coefficients for the interface between the $(m-1)^{th}$ and m^{th} layers.

Modelling the phase delay encountered across a layer, we relate the amplitudes at the start(left) and end(right) of the m^{th} layer as:

$$\begin{pmatrix} E_{m,l}^{+} \\ E_{m,l}^{-} \end{pmatrix} = P_m \begin{pmatrix} E_{m,r}^{+} \\ E_{m,r}^{-} \end{pmatrix}; \quad P_m = \begin{pmatrix} e^{-j\delta m} & 0 \\ 0 & e^{-j\delta m} \end{pmatrix} \tag{2}$$

Where δ is the phase difference and is equal to knd, k is the wave number, n is the refractive index and d is the width of the layer. Extrapolating the idea for all layers, we can write:

$$\begin{pmatrix} E_{0}^{+} \\ E_{0}^{-} \end{pmatrix} = I_{01}P_1 I_{12}P_2 I_{23}P_3....P_N I_{N,N+1} \begin{pmatrix} E_{N+1}^{+} \\ E_{N+1}^{-} \end{pmatrix}; \tag{3}$$

The Cascade can be represented as the net transfer matrix T. In the figure, E_{N+1}^{-} is the incident field from the other end which is 0 in this case. So, the incident field $E_0^+ = T(1,1)$ times the scattered field E_{N+1}^{+}. Similarly, reflected field $E_0^- = T(2,1)$ times the scattered field.

So we first calculate the net transfer matrix and then find the analytical solutions of the reflection and transmission coefficients from the matrix elements.

$$r = \frac{E_0^-}{E_0^+} = \frac{T_{21}}{T_{11}}; \quad t = \frac{E_{N+1}^{+}}{E_0^+} = \frac{1}{T_{11}} \tag{4}$$

We calculate these transmission and reflection coefficients for different wavelengths and try to analyse any interesting phenomenon happening for some specific range of lambda. Since the layers are placed periodically in case 1, we expect the structure to behave like a resonator. In such resonators, resonance occurs at periodic values of k ($=2\pi/\lambda$). So, we also sweep k from 0 to 6π. The transmission and reflection coefficients obtained are:

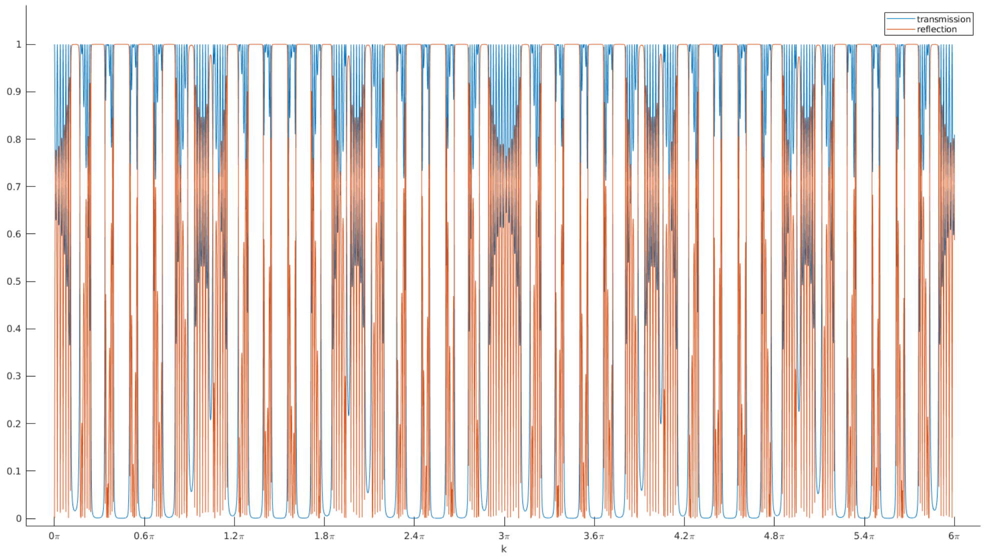


Figure 2: Transmission and Reflection coeffs. for structure 1

Plots for a similar analysis for structure 2:

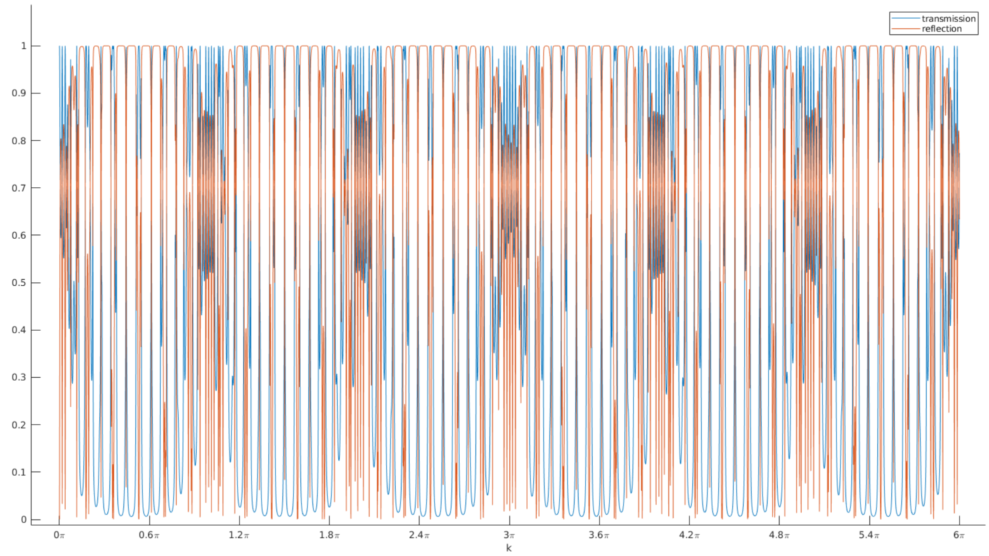


Figure 3: Transmission and Reflection coeffs. for structure 2

As expected, the peaks in transmission coefficient are periodic in k with an approximate period of 0.15π and it was observed to change with the refractive index of the medium.

FEM Formulation

1.3 Total Field Formulation

Weak form of FEM for the problem:

$$\int_{\Omega_m} \mathrm{d}x \, \frac{\mathrm{d}W_m(x)}{\mathrm{d}x} \frac{\mathrm{d}U(x)}{\mathrm{d}x} - \int_{\Omega_m} \mathrm{d}x \, k(x)^2 W_m(x) U(x) = \left[W_m(x) \frac{\mathrm{d}U(x)}{\mathrm{d}x} \right]_{\text{end pts.}} \tag{5}$$

We have the linear **basis functions** N_1, N_2, and ${}^iT = {}^{i-1}N_2 + {}^iN_1$.
Field in terms of basis functions:

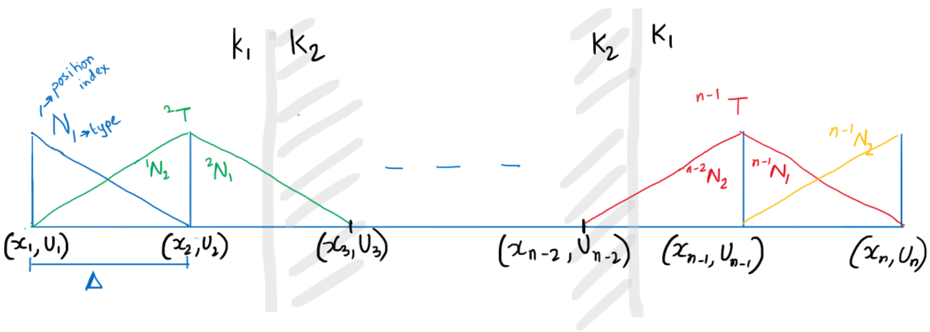
$$U(x) = U_1{}^1N_1 + U_2{}^2T + \cdots + U_{n-1}{}^{n-1}T + U_n{}^{n-1}N_2 \tag{6}$$

Weight/testing functions:

$$W_1 = {}^1N_1; \qquad W_m = {}^mT \quad (m \neq \{1, n\}); \qquad W_n = {}^{n-1}N_2 \tag{7}$$

The **spacing** between successive discretization points (nodes) is Δ. The discretization is done for the whole length uniformly. The interfaces may lie in between the nodes, and care has to be taken while writing the equations.

Radiation boundary condition:
In the 'leftmost' layer the scattered field should be of the form: $U_s(x) = |U_s(x)| \exp(\iota kx)$, i.e., a backward travelling wave (we have $e^{\iota\omega t}$ convention) and in the 'rightmost' layer it should be



forward travelling. Hence on the **left** boundary:

$$\begin{aligned} &\frac{\mathrm{d}U_s}{\mathrm{d}x} = \iota k U_s \\ \implies &\frac{\mathrm{d}}{\mathrm{d}x}(U - U_{in}) = \iota k (U - U_{in}) \\ \implies &\frac{\mathrm{d}U}{\mathrm{d}x} = (\alpha_{in} - \alpha_l) U_{in}(x) + \alpha_l U(x) \end{aligned} \tag{8}$$

where $\alpha_{in} = -\iota k$ (corresponding to a forward travelling wave), and $\alpha_l = \iota k$. Similarly for the right boundary, $\alpha_r = -\iota k$

Hence, the equation becomes:

$$\mathbb{A}\mathbf{U} = \mathbf{b} \tag{9}$$

where $\mathbb{A} =$

$$\begin{bmatrix} \frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^1 N_1{}^1 N_1 + \alpha_l & -\frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^1 N_1{}^1 N_2 & 0 & \cdots & 0 \\ -\frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^1 N_2{}^1 N_1 & \frac{2}{\Delta} - \int \mathrm{d}x\,(k^2)^1 N_2{}^1 N_2 - \int \mathrm{d}x\,(k^2)^2 N_1{}^2 N_1 & -\frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^2 N_1{}^2 N_2 & \cdots & 0 \\ 0 & \ddots & \ddots & \cdots & 0 \\ \vdots & & & & \\ 0 & \cdots & -\frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^{n-1} N_2{}^{n-1} N_1 & \frac{1}{\Delta} - \int \mathrm{d}x\,(k^2)^{n-1} N_2{}^{n-1} N_2 - \alpha_r & \end{bmatrix} \tag{10}$$

$$\mathbf{U} = \begin{bmatrix} U_1 & U_2 & \cdots & U_{n-1} & U_n \end{bmatrix}^T \tag{11}$$

$$\mathbf{b} = \begin{bmatrix} -(\alpha_{in} - \alpha) U_{in}(x_1) & 0 & \cdots & 0 & (\alpha_{in} - \alpha) U_{in}(x_n) \end{bmatrix}^T \tag{12}$$

Note: While calculating the integrals in $\mathbb{A}$, if a boundary is encountered, the integral is split accordingly and the appropriate 'k' values are used.

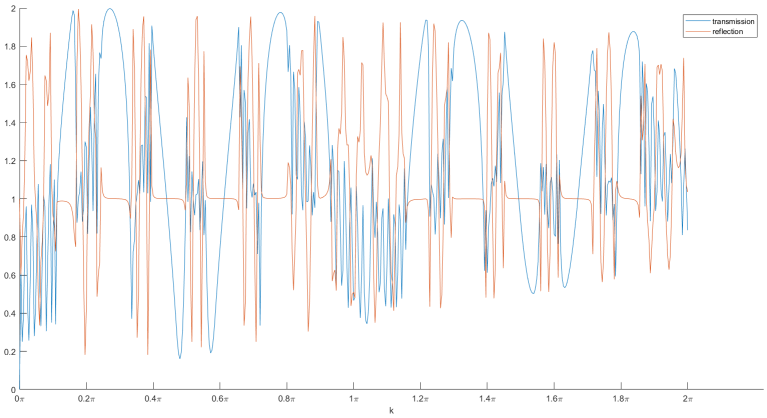


Figure 4: Reflection and transmission coefficients for Structure 1, using total field formulation

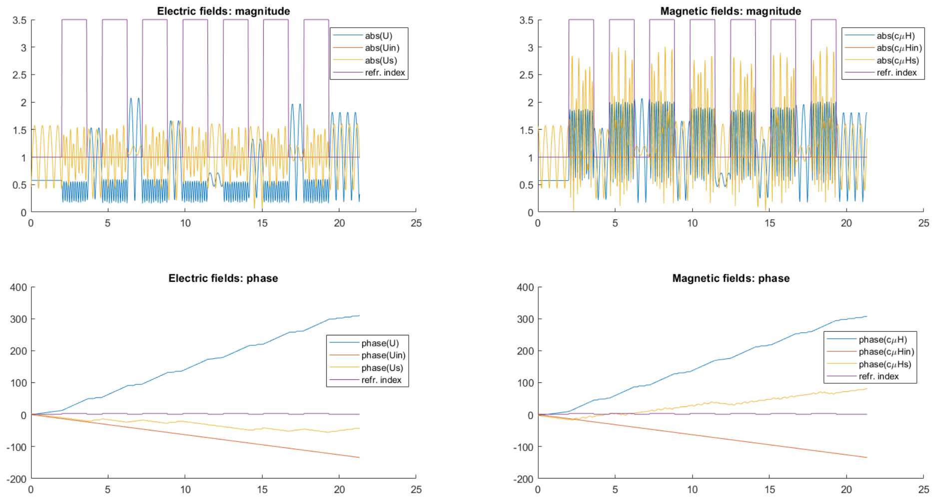


Figure 5: Magnitude and phase of fields w.r.t. position for structure 1 for $k = 2\pi$, using total field formulation.

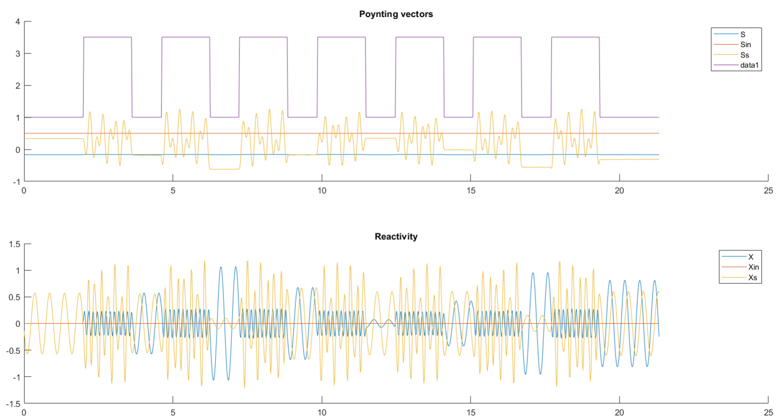


Figure 6: Power of the field for $k = 2\pi$, for total field formulation

1.4 Scattered Field Formulation

In the scattered field formulation, we use the scattered field U_s where the object is not present (i.e. where U_{in} satisfies the maxwell's equations implying so does U_s) and the total field U where it is. Hence, the discretization is done individually for the layers, so that a node always lies on an interface.

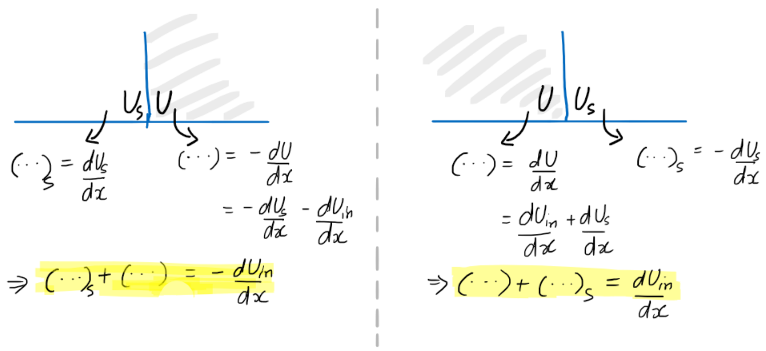
At the first and last boundary: The radiation boundary condition is simply,

$$\frac{\mathrm{d}U_s}{\mathrm{d}x} = \alpha_{l/r} U_s \tag{13}$$

since the variable used is the scattered field. Hence, in the matrix formulation, the L.H.S. remains same as in the total field formulation case, and the R.H.S. is zero.

At intermediate points (no interface): The equations look exact same as in the total formulation case. Though, there is a simplification w.r.t. the implementation. Since the interface case is formulated separately, the integral terms correspond to just one type of region, and hence we need not split it.

At the interface: In the total field formulation, if there had been a node at the interface,



there would have just been one variable and we would have had one equation from the corresponding T testing function. In the scattered field formulation, we have 2 variables at the interface node, U_s and U. The two equations corresponding to them are as follows: (say the interface is at x_i)

$$U(x_i) - U_s(x_i) = U_{in}(x_i) \tag{14}$$

and,

$$\begin{cases} \text{for air} \to \text{medium interface:} \\ \int \mathrm{d}x \, \frac{\mathrm{d}({}^{i-1}N_2)}{\mathrm{d}x} \frac{\mathrm{d}U_s}{\mathrm{d}x} - k^2 \int \mathrm{d}x \, {}^{i-1}N_2 U_s + \int \mathrm{d}x \, \frac{\mathrm{d}({}^{i}N_1)}{\mathrm{d}x} \frac{\mathrm{d}U}{\mathrm{d}x} - (kn)^2 \int \mathrm{d}x \, {}^{i}N_1 U = -\alpha_{in} U_{in}(x_i) \\ \\ \text{for medium} \to \text{air interface:} \\ \int \mathrm{d}x \, \frac{\mathrm{d}({}^{i-1}N_2)}{\mathrm{d}x} \frac{\mathrm{d}U}{\mathrm{d}x} - (kn)^2 \int \mathrm{d}x \, {}^{i-1}N_2 U + \int \mathrm{d}x \, \frac{\mathrm{d}({}^{i}N_1)}{\mathrm{d}x} \frac{\mathrm{d}U_s}{\mathrm{d}x} - k^2 \int \mathrm{d}x \, {}^{i}N_1 U = \alpha_{in} U_{in}(x_i) \end{cases} \tag{15}$$

where k is the wavevector for air, and n is he refractive index of the medium.

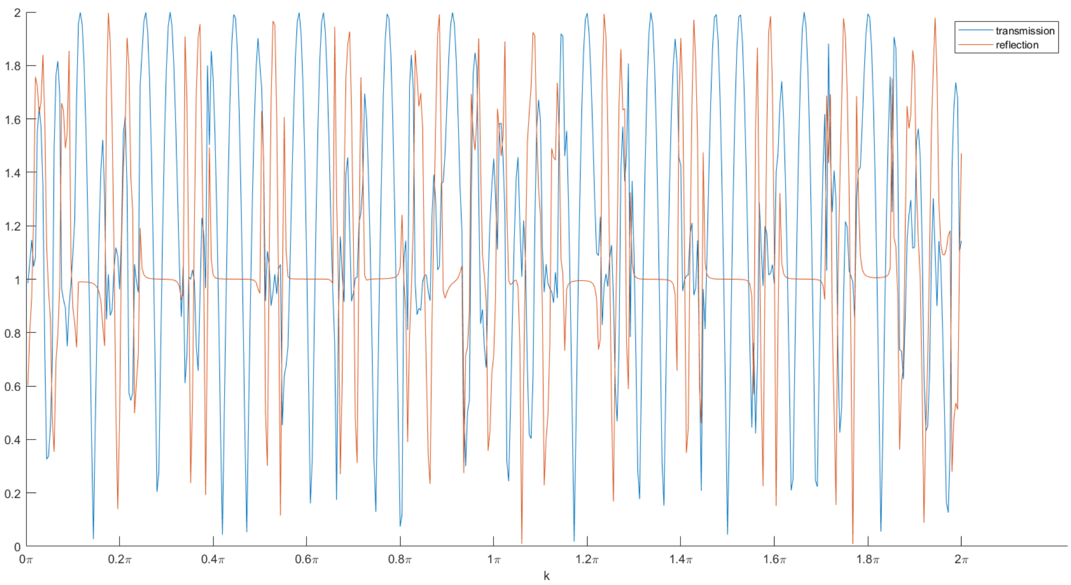


Figure 7: Reflection and transmission coefficients for Structure 1, using scattered field formulation

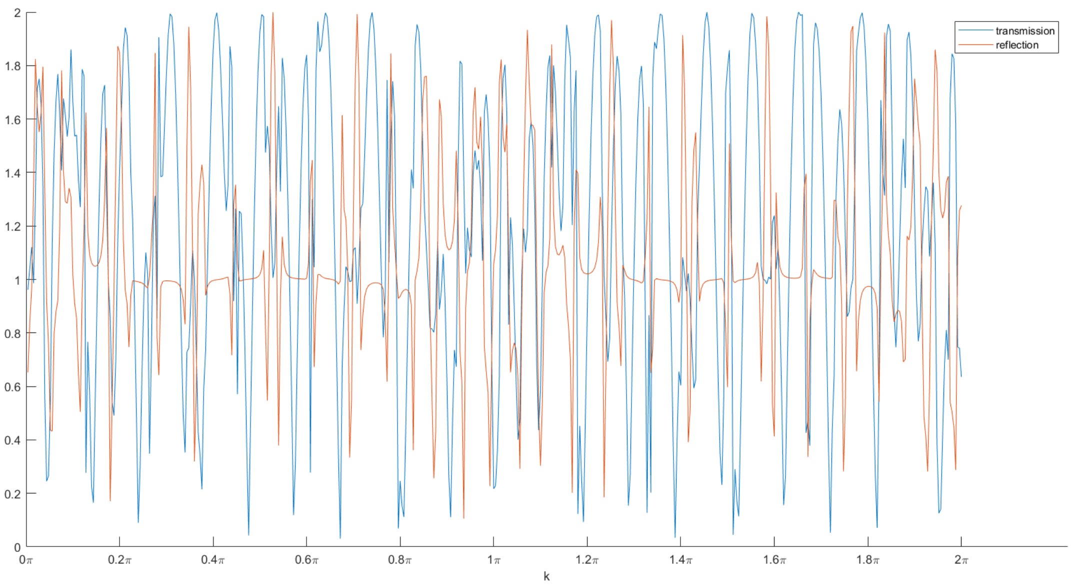


Figure 8: Reflection and transmission coefficients for Structure 1, using scattered field formulation

Comments on Fig. 9 & 10

- The field U (blue dots) represents the scattered field when the refractive index is 1, and the total field when it is not.
- **Phase**

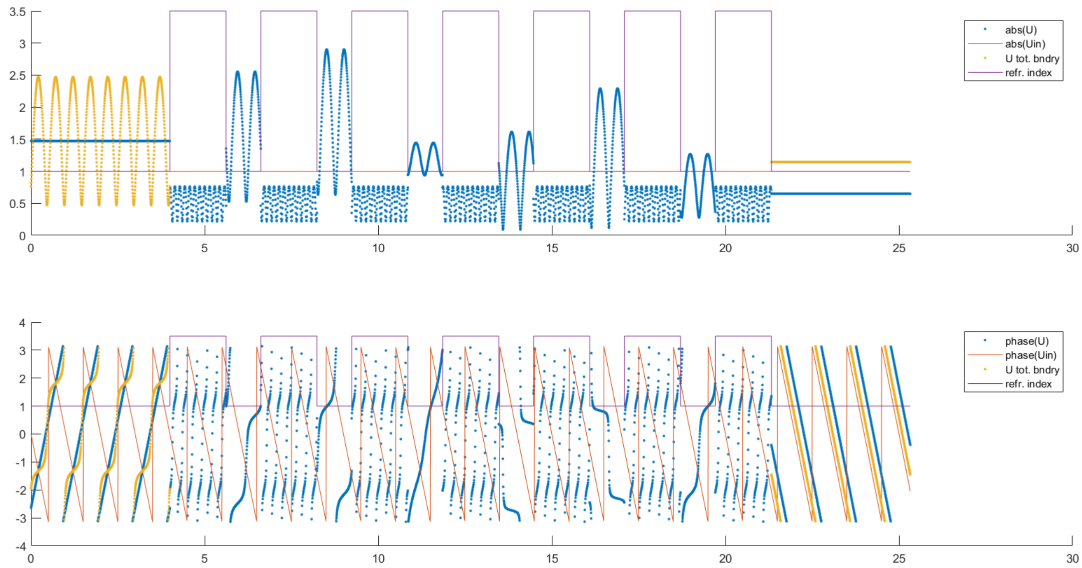


Figure 9: Magnitude and phase of fields w.r.t. position for structure 1 for $k = 2\pi$, using scattered field formulation.

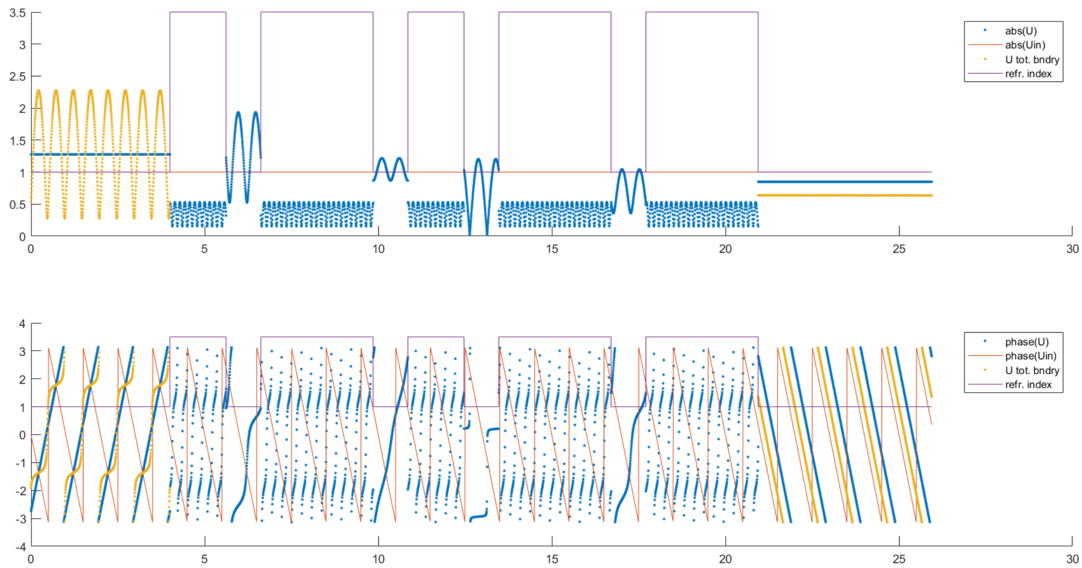


Figure 10: Magnitude and phase of fields w.r.t. position for structure 2 for $k = 2\pi$, using scattered field formulation.

 – The phase of the incident wave decreases with position (as per the definition $(exp(-\imath kx))$).
 – The phase of the scattered field increases in the first layer (hence backward travelling wave) and decreases in the last layer (hence forward travelling wave) as it should be.

- Outside the object (i.e. in the first and last layer), the magnitude of the scattered field is constant and the phase varies linearly, hence it is a sinusoidal wave.

- The magnitude of the total field (yellow dots) in the first layer (air) is oscillating, implying standing waves, as expected.
- No such standing waves in the total field in the last layer. It's magnitude is constant, and phase decreases linearly. Hence it is a forward travelling wave.

2 Question 2

2.1 Introduction

The aim is to generate 2D scalar (node based) basis functions for FEM. When discretising the region into small triangles, these basis functions are used to express the unknown variable (electric field in CEM) as a linear combination of the values at each vertex of the discretisation unit. So, given a triangular discretisation unit, we need to generate 3 linear functions, one for each vertex such that it equals 1 at the corresponding node and 0 at the remaining nodes.

So, given Δ123, generate linear functions L1, L2 and L3 such that the unknown field U at any point inside the triangle can be expressed as U1*L1 + U2*L2 + U3*L3. At node 1, L2 ad L3 = 0. So, U(node 1) = U1. Same condition applies to nodes 2 and 3 also.

2.2 Analytical Expression of L and ∇L

We assume the triangle Δ123 to have the coordinates of its vertices as pt1(x1, y1), pt2(x2, y2) and pt3(x3, y3) and P to be an arbitrary point inside the triangle. Intuitively, we can define the expression for L1 as: Area of Δ123 / Area of $\Delta P23$. So, when P = pt1, the areas will be same and L1 = 1. When P = pt2 or P = pt3, the numerator will be the area of a line segment, evaluating to 0.

Algebraically simplifying the expression for L1 and extrapolating for L2 and L3, we get:

$$L_1^e(x,y) = \frac{(x_2y_3 - x_3y_2) - x(y_3 - y_2) + y(x_3 - x_2)}{(x_2y_3 - x_3y_2) - x_1(y_3 - y_2) + y_1(x_3 - x_2)} \tag{16a}$$

$$L_2^e(x,y) = \frac{(x_3y_1 - x_1y_3) - x(y_1 - y_3) + y(x_1 - x_3)}{(x_3y_1 - x_1y_3) - x_2(y_1 - y_3) + y_2(x_1 - x_3)} \tag{16b}$$

$$L_3^e(x,y) = \frac{(x_1y_2 - x_2y_1) - x(y_2 - y_1) + y(x_2 - x_1)}{(x_1y_2 - x_2y_1) - x_3(y_2 - y_1) + y_3(x_2 - x_1)} \tag{16c}$$

It can be easily verified that these 3 functions satisfy the condition $L_i(pti) = 1$ and $L_i(ptj) = 0$ and $L_i(ptk) = 0$ for i,j,k $\in \{1, 2, 3\}$

When implementing scalar 2D FEM, the matrices used will have elements of the form $\iint_e L_i^e(r)L_j^e(r)dxdy$ and $\iint_e \nabla L_i^e(r) \cdot \nabla L_j^e(r)dxdy$, where e is the arbitrary triangle chosen as the discretisation unit.

2.3 Coordinate Transformation

Numerically integrating a function over an arbitrary 2D shape will be complicated. So, we perform a linear transformation between the given triangle and an unit right triangle with the vertices (0,0), (0,1) and (1,0).

The transformation between (x,y) coordinates in given coordinate syste and (u,v) coordinates in the system with unit right triangle is as follows:

$$x = x_1 + (x_2 - x_1)u + (x_3 - x_1)v; \quad y = y_1 + (y_2 - y_1)u + (y_3 - y_1)v; \tag{17}$$

Here, the vertex mapping from (u,v) to (x,y) is as follows:
$(0,0) \longrightarrow (x1, y1)$; $(1,0) \longrightarrow (x2, y2)$; $(0,1) \longrightarrow (x3, y3)$

So, when we attempt to integrate the function $L_i^e(r)L_j^e(r)$, we use change of variables to get the following equation:

$$\iint_e L_i^e(r)L_j^e(r)dxdy = \iint_\Delta L_i^e(r')L_j^e(r')dudv \cdot J \tag{18a}$$

Similarly,

$$\iint_e \nabla L_i^e(r) \cdot \nabla L_j^e(r)dxdy = \iint_\Delta \nabla L_i^e(r') \cdot \nabla L_j^e(r')dudv \cdot J \tag{18b}$$

where, J is the determinant of the Jacobian matrix arising from the transformation. The Jacobian for the following transformation is:

$$J = det \begin{bmatrix} \partial x/\partial u & \partial x/\partial v \\ \partial y/\partial u & \partial y/\partial v \end{bmatrix} = \begin{vmatrix} (x_2 - x_1) & (x_3 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) \end{vmatrix} \tag{19}$$

2.4 Validating the Basis Functions

To check the validity of the scalar basis functions generated, we first check the values of the basis functions at the the 3 vertices and ensure that each basis function has unit magnitude only at the allocated vertex and is 0 at the other 2 vertices.

To test the code for the 2nd case, i.e., $\iint_e \nabla L_i^e(r) \cdot \nabla L_j^e(r)dxdy$, we need to check the coordinate transformation done by the code. Since L_i^e is a linear function, its gradient will be a constant. So, we are essentially integrating a constant over an arbitrary triangle. We manually calculate the area of the arbitrary triangle chosen and compare it with the area obtained from the integral in the code. They were found to match perfectly.

To test the code for 1st case, i.e., $\iint_e L_i^e(r)L_j^e(r)dxdy$, we take a simple case where the input triangle is chosen to be the unit right triangle. So, L1(x,y) will be of the form $L1(x, y) = 1-x-y$. So, the self multiplicative terms (i==j) will have $\iint_e L_i^2(x, y)dxdy$. This was manually worked out to be :

$$\iint_e L_i^2(x, y)dxdy = \iint_e (1 - x - y)^2dxdy \iint_e (1 + x^2 + y^2 - 2x - 2y + 2xy)dxdy = 1/12 \tag{20}$$

The diagonal elements in the A matrix generated by the code also have the value $0.0833 = 1/12$.

We also plot the basis functions generated for this case below: As expected, the planes generated each have unit magnitude at the corresponding vertices of the unit triangle and are 0 at the other vertices.

So, we separately verify the bias functions generated, the integration and the coordinate transformation in the code.

3 Appendix 1

3.1 get_multilayer_eps.m

```
function [eps_arr] = get_multilayer_eps(seq, n, eps_n)
        %seq=1: alternate array, seq=2: fibonocci
        %n: number of terms in the sequence
        %assumes only 2 materials: material N and Air

        eps_arr = [];
        if seq == 1    %alternate arrangement
                for i = 1:n-1
                        eps_arr(end+1:end+2) = [eps_n 1];
```

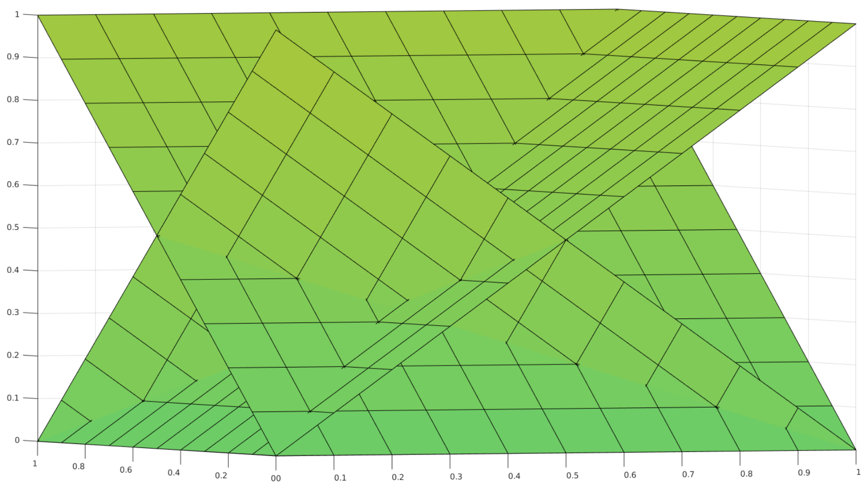


Figure 11: Scalar basis functions for a unit triangle

```
                end
                eps_arr(end+1) = eps_n;
        else    %fibonocci arrangement
                f1 = [1];
                f2 = [eps_n];
                if n == 1
                        eps_arr = f1;
                elseif n == 2
                        eps_arr = f2;
                else
                        for i = 3:n
                                eps_arr = cat(2, f2, f1);
                                f1 = f2;
                                f2 = eps_arr;
                        end
                end
         end
```

3.2 get_width.m

```
function [wid_arr] = get_width(eps_arr,wid, ratio)
        %generates array of widths of each layer

         wid_a = wid;
         wid_n = wid * ratio;

         wid_arr = zeros(size(eps_arr));
         for i = 1 : size(eps_arr')
                 if eps_arr(i) == 1
                         wid_arr(i) = wid_a;     %set width of air gap
                 else
                         wid_arr(i) = wid_n;     %set width of material layer
                 end
         end
```

3.3 HW4_1a.m

```
tic

n = 7;    %number of layers
seq =1;  %choose structure 1 or 2
```

```
eps_n = 3.5^2; %relative permitivitty
air_thickness = 1;
ratio = ((sqrt(5) + 1)/2);    %ratio of thicknesses of material layer and air gap
k_vec = 0:2*pi/1000:2*pi;  %sweep k instead of lambda to check for periodicity

len_vec = length(k_vec);
tou_arr = zeros(1,len_vec);
ref_arr = zeros(1,len_vec);

eps_arr = get_multilayer_eps(seq, n, eps_n);
wid_arr = get_width(eps_arr, air_thickness, ratio);
len = size(eps_arr');

tou = @(eps1, eps2) 2*sqrt(eps1) ./ (sqrt(eps1) + sqrt(eps2));     % transmission coeff. from fresnel eqn
ref = @(eps1, eps2)  (sqrt(eps1) - sqrt(eps2)) ./ (sqrt(eps1) + sqrt(eps2));     %reflection coeff. from fresnel eqn

I_mat  =@(t,r) (1/t) .* [1 r; r 1;];    %imposes constraints at every interface
delta = @(eps,wid, k) k*sqrt(eps)*wid;
P_mat = @(delta) [exp(1i*delta) 0; 0 exp(-1i*delta)];    %includes phase difference across a layer

eps_arr(end+1) = 1;

for k_id = 1:len_vec
        %get net transfer matrix
        T_mat = I_mat( tou(1, eps_arr(1)), ref(1, eps_arr(1)) );
        for i = 1:len
              T_mat = T_mat * P_mat( delta(eps_arr(i), wid_arr(i), k_vec(k_id)) ) * I_mat( tou(eps_arr(i), eps_arr(i+1)), ref(eps_arr(i), eps_arr(i+1)) );
        end

        %get tou, ref from T matrix
        net_tou = 1 / abs(T_mat(1,1));
        net_ref = abs( T_mat(2,1) / T_mat(1,1) );

        tou_arr(k_id) = net_tou;
        ref_arr(k_id) = net_ref;

end

toc

figure;
hold on;
plot(k_vec, tou_arr);
plot(k_vec, ref_arr);
xticks(k_vec(1:fix(len_vec/10):len_vec))
xticklabels(strcat(string(k_vec(1:fix(len_vec/10):len_vec)./pi), '\pi'))
xlabel('k')
hold off;
legend("transmission","reflection");
```

3.4 HW4_1b_SF.m

Scattered field formulation.

```
tic

% Defining the parameters
n_layers = 7;
seq =2;
```

```
eps_r = 3.5^2; %relative permitivitty
n = sqrt(eps_r); %refractive index
air_thickness = 1;
ratio = ((sqrt(5) + 1)/2);
k_max = 2*pi;
k_min = 0;
num_pts_per_lyr = 201;

% Discretization lengths for air and medium
DL1 = air_thickness/(num_pts_per_lyr-1);
DL2 = ratio*air_thickness/(num_pts_per_lyr-1);

% The vector with k values
k_vec = k_min:(k_max-k_min)/500:k_max;
%k_vec = 2*pi;
len_vec = length(k_vec);

% Initializing arrays to store ref. and trans. coefficients
tau_arr = zeros(1,len_vec);
ref_arr = zeros(1,len_vec);

% Defining the ref. inde array and the corresponding widths
n_obj_arr = [1 1 1 1 (get_multilayer_eps(seq, n_layers, eps_r)).^0.5 1 1 1 1];
wid_arr = get_width(n_obj_arr, air_thickness, ratio);
wid_arr([1 2 3 4 (end-3) (end-2) (end-1) end]) = air_thickness;

% Combining consecutive objects with the same ref. index (i.e. treating as one,
    and updating width)
num_objs = length(n_obj_arr);
n_obj_eff_arr = [n_obj_arr(1)];
wid_eff_arr = [wid_arr(1)];
for i = 2:num_objs
    if n_obj_arr(i) == n_obj_arr(i-1)
        wid_eff_arr(end) = wid_eff_arr(end) + wid_arr(i);
    else
        n_obj_eff_arr(end+1) = n_obj_arr(i);
        wid_eff_arr(end+1) = wid_arr(i);
    end
end
num_objs_eff = length(n_obj_eff_arr);

% Effective number of points in each layer, array with position values,
% array with corresponding n values
num_pts_eff_each_lyr = zeros(1,num_objs_eff);
z_arr = 0:DL1:wid_eff_arr(1);
n_arr = ones(1,length(z_arr));
for i = 1:num_objs_eff
    if n_obj_eff_arr(i) == 1
        num_pts_eff_each_lyr(i) = fix(wid_eff_arr(i)/DL1) + 1;
        if i>1
            z_append = sum(wid_eff_arr(1:i-1)):DL1:sum(wid_eff_arr(1:i));
            z_arr = [z_arr, z_append];
            n_arr = [n_arr, ones(1, length(z_append))];
        end
    else
        num_pts_eff_each_lyr(i) = fix(wid_eff_arr(i)/DL2) + 1;
        z_append = sum(wid_eff_arr(1:i-1)):DL2:sum(wid_eff_arr(1:i));
        z_arr = [z_arr, z_append];
        n_arr = [n_arr, n_obj_eff_arr(i)*ones(1, length(z_append))];
    end
end

% Number of equations
```

```
num_eq = sum(num_pts_eff_each_lyr);

% Precomputing a few quantities
intN1N1_1 = DL1/3; %integral(N1*N1) for \Delta = DL1
intN1N2_1 = DL1/6;
intN2N2_1 = DL1/3;

intN1N1_2 = DL2/3;
intN1N2_2 = DL2/6;
intN2N2_2 = DL2/3;

% Running the loop over k
for k_id = 1:len_vec

        k = k_vec(k_id);

        % Initialising
        A = zeros(num_eq);
        b = zeros(num_eq, 1);

        % The incident field as a function of position
        Uin = @(z) exp(-1j*k.*z);

        % alpha values
        alpha_in = -1j*k;
        alpha_s_l  = -1j*k;
        alpha_s_r = 1j*k;

        off_diag_1 = -1/DL1 - k^2*intN1N2_1;
        diag_1 = 2/DL1 - k^2*(intN1N1_1 + intN2N2_1);

        off_diag_2 = -1/DL2 - (k*n)^2*intN1N2_2;
        diag_2 = 2/DL2 - (k*n)^2*(intN1N1_2 + intN2N2_2);

        % Defining equations
        id_eq =1;
        A(1,[1 2]) = [(diag_1/2+alpha_s_l) off_diag_1];
        for i = 2:(num_pts_eff_each_lyr(1)-1)
            id_eq = id_eq + 1;
            A(i,[i-1 i i+1]) = [off_diag_1 diag_1 off_diag_1];
        end
        id_eq = id_eq+1;
        A(id_eq, [(id_eq-1) (id_eq) (id_eq+1) (id_eq+2)]) = [off_diag_1 diag_1/2 diag_2/2 off_diag_2];
        b(id_eq) = -alpha_in*Uin(wid_eff_arr(1));

        for id_obj = 2:num_objs_eff
            if n_obj_eff_arr(id_obj) ~= 1 % medium
                id_eq = id_eq+1;
                A(id_eq, [id_eq-1 id_eq]) = [-1 1];
                b(id_eq) = Uin(sum(wid_eff_arr(1:id_obj-1)));
                for i = 2:num_pts_eff_each_lyr(id_obj)-1
                    id_eq = id_eq + 1;
                    A(id_eq,[id_eq-1 id_eq id_eq+1]) = [off_diag_2 diag_2 off_diag_2];
                end
                id_eq = id_eq+1;
                if id_obj ~= num_objs_eff
                    A(id_eq, [(id_eq-1) (id_eq) (id_eq+1) (id_eq+2)]) = [off_diag_2 diag_2/2 diag_1/2 off_diag_1];
                    b(id_eq) = alpha_in*Uin(sum(wid_eff_arr(1:id_obj)));
                else % last layer
```

```
                    A(id_eq, [(id_eq-1) (id_eq)]) = [off_diag_2 (diag_2/2-alpha_s)];
                end

            else % air
                id_eq = id_eq+1;
                A(id_eq, [id_eq-1 id_eq]) = [1 -1];
                b(id_eq) = Uin(sum(wid_eff_arr(1:id_obj-1)));
                for i = 2:num_pts_eff_each_lyr(id_obj)-1
                    id_eq = id_eq + 1;
                    A(id_eq,[id_eq-1 id_eq id_eq+1]) = [off_diag_1 diag_1 off_diag_1];
                end
                id_eq = id_eq+1;
                if id_obj ~= num_objs_eff
                    A(id_eq, [(id_eq-1) (id_eq) (id_eq+1) (id_eq+2)]) = [off_diag_1 diag_1/2 diag_2/2 off_diag_2];
                    b(id_eq) = -alpha_in*Uin(sum(wid_eff_arr(1:id_obj)));
                else % last layer
                    A(id_eq, [(id_eq-1) (id_eq)]) = [off_diag_1 (diag_1/2-alpha_s_r)];
                end
            end
        end

        % Solving the equations
        U = (A\b)';

        % Calculating Uin
        Uin_arr = Uin(z_arr);

        % Reflection and transmission coefficients
        ref_arr(k_id) = sum(abs( U(1:10) ))/10;
        tau_arr(k_id) = sum(abs( U(end-9:end)+Uin_arr(end-9:end) ))/10;

end

% U total in the first and last layers
U_tot_first_lyr = U(1:num_pts_eff_each_lyr(1)) + Uin_arr(1:num_pts_eff_each_lyr(1));
U_tot_last_lyr  = U(end-num_pts_eff_each_lyr(end)+1:end) + Uin_arr(end-num_pts_eff_each_lyr(end)+1:end);
U_tot_bndry_lyrs = [U_tot_first_lyr U_tot_last_lyr];
z_arr_bndry_lyrs = [z_arr(1:num_pts_eff_each_lyr(1)) z_arr(end-num_pts_eff_each_lyr(end)+1:end)];

% Plotting ref. and trans. coefficients
figure;
hold on;
plot(k_vec, tau_arr);
plot(k_vec, ref_arr);
xticks(k_vec(1:fix(len_vec/10):len_vec))
xticklabels(strcat(string(k_vec(1:fix(len_vec/10):len_vec)./pi), '\pi'))
xlabel('k')
hold off;
legend("transmission","reflection");

% Plotting fields
figure
% Magnitude
subplot(2,1,1)
hold on
plot(z_arr, abs(U),'.')
```

```
plot(z_arr, abs(Uin_arr))
plot(z_arr_bndry_lyrs, abs(U_tot_bndry_lyrs),'.');
plot(z_arr, n_arr)
legend('abs(U)','abs(Uin)', 'U tot. bndry', 'refr. index')

% Phase
subplot(2,1,2)
hold on
plot(z_arr, (angle(U)),'.');
plot(z_arr, (angle(Uin_arr)));
plot(z_arr_bndry_lyrs, (angle(U_tot_bndry_lyrs)),'.');
plot(z_arr, n_arr)
legend('phase(U)','phase(Uin)', 'U tot. bndry', 'refr. index')

toc
```

3.5 HW4_1b_TF.m

Total field formulation.

```
tic

n_layers = 7;
seq =1;
eps_r = 3.5^2; %relative permitivitty
n = eps_r^0.5; %refractive index
air_thickness = 1;
ratio = ((sqrt(5) + 1)/2);
k_max = 2*pi;
k_min = 0;

num_pts = 1000;

k_vec = k_min:(k_max-k_min)/500:k_max;
%k_vec = 0.6*pi;
len_vec = length(k_vec);
tau_arr = zeros(1,len_vec);
ref_arr = zeros(1,len_vec);

global wid_arr DL

n_obj_arr = [1 (get_multilayer_eps(seq, n_layers, eps_r)).^0.5 1];
wid_arr = get_width(n_obj_arr, air_thickness, ratio);
wid_arr([1 end]) = 2;

DL = sum(wid_arr)/(num_pts-1);
n_node_arr = zeros(1,num_pts);

id_obj = 1;
for i = 1:num_pts
    if (i-1)*DL > sum(wid_arr(1:id_obj))
        id_obj = id_obj+1;
    end
    n_node_arr(i) = n_obj_arr(id_obj);
end

DLsq = DL^2;
intN1N1 = @(x1, x2) (x2^3-x1^3)/(3*DLsq) - (x2^2-x1^2)/DL + (x2-x1);
intN1N2 = @(x1, x2) -(x2^3-x1^3)/(3*DLsq) + (x2^2-x1^2)/(2*DL);
intN2N2 = @(x1, x2) (x2^3-x1^3)/(3*DLsq);

for k_id = 1:len_vec

```

```
    id_obj_arr = zeros(1, num_pts);

    k = k_vec(k_id);
    Uin = @(x) exp(-1j*k.*x);

    alpha_in = -1j*k;
    alpha_left = -1j*k;
    alpha_right = 1j*k;

    A = zeros(num_pts);
    b = zeros(num_pts, 1);

    %assuming DL<wid_arr(1)
    id_obj = 1;
    id_obj_arr(1) = id_obj;

    n = n_node_arr(1);
    A(1,1) = 1/DL - (k*n)^2*intN1N1(0,DL) + alpha_left;
    A(1,2) = -1/DL - (k*n)^2*intN1N2(0,DL);
    b(1) = -(alpha_in - alpha_left);

    f_arr = zeros(1,num_pts);
    f = 1;
    f_arr(1) = f;
    for i = 2:(num_pts-1)
        n_im1 = n_node_arr(i-1);
        n_i   = n_node_arr(i);
        n_ip1 = n_node_arr(i+1);

        if n_i ~= n_im1
            id_obj = id_obj+1;
        end
        id_obj_arr(i) = id_obj;

        %A(i,i-1)
        A(i,i-1) = -1/DL - (k*n_im1)^2*intN1N2(0,f*DL) - (k*n_i)^2*intN1N2(f*DL,DL);

        %A(i,i):
        int1 = -(k*n_im1)^2*intN2N2(0,f*DL) - (k*n_i)^2*intN2N2(f*DL,DL);

        f = k1k2split(i, n_i, n_ip1, id_obj);
        f_arr(i) = f;

        int2 = -(k*n_i)^2*intN1N1(0,f*DL) - (k*n_ip1)^2*intN2N2(f*DL,DL);
        A(i,i) = 2/DL + int1 + int2;

        %A(i,i+1)
        A(i,i+1) = -1/DL - (k*n_i)^2*intN1N2(0,f*DL) - (k*n_ip1)^2*intN1N2(f*DL,DL);

    end

    n = n_node_arr(end);
    A(num_pts,num_pts-1) = -1/DL - (k*n)^2*intN1N2(0,DL);
    A(num_pts,num_pts) = 1/DL - (k*n)^2*intN2N2(0,DL) - alpha_right;
    b(end) = (alpha_in - alpha_right)*Uin(sum(wid_arr));

    id_obj_arr(end) = id_obj;

    U = (A\b)';

    ref_arr(k_id) = sum(abs(U(1:10)-Uin(0:DL:9*DL)))/10;
```

```
    tau_arr(k_id) = sum(abs(U(end-9:end)))/10;
end

toc

figure;
hold on;
plot(k_vec, tau_arr);
plot(k_vec, ref_arr);
xticks(k_vec(1:fix(len_vec/10):len_vec))
xticklabels(strcat(string(k_vec(1:fix(len_vec/10):len_vec)./pi), '\pi'))
xlabel('k')
hold off;
legend("transmission","reflection");

z_arr = 0:DL:sum(wid_arr);
Uin_arr = Uin(z_arr);
Us = U-Uin_arr;

cmu_Hin_arr = Uin_arr;
cmu_H_arr = zeros(1,num_pts);
cmu_H_arr(1) = (1j/k)*(U(2) - U(1))/DL; % c/omega = k
cmu_H_arr(2:end-1) = (1j/k)*(U(3:end) - U(1:end-2))/(2*DL);
cmu_H_arr(end) = (1j/k)*(U(end) - U(end-1))/DL;
cmu_Hs_arr = cmu_H_arr - cmu_Hin_arr;

Sin_arr = 0.5*real(Uin_arr.*conj(cmu_Hin_arr));
S_arr = 0.5*real(U.*conj(cmu_H_arr));
Ss_arr = 0.5*real(Us.*conj(cmu_Hs_arr));

Xin_arr = 0.5*imag(Uin_arr.*conj(cmu_Hin_arr));
X_arr = 0.5*imag(U.*conj(cmu_H_arr));
Xs_arr = 0.5*imag(Us.*conj(cmu_Hs_arr));

% figure
% hold on
% plot(z_arr, n_node_arr, '.')
% plot(z_arr, id_obj_arr,'.')
% plot(z_arr, 10*f_arr,'.')
% legend('n','id_obj','10*f')

figure
subplot(2,2,1)
hold on
plot(z_arr, abs(U))
plot(z_arr, abs(Uin_arr))
plot(z_arr, abs(Us))
plot(z_arr, n_node_arr)
legend('abs(U)','abs(Uin)','abs(Us)', 'refr. index')
title('Electric fields: magnitude')

subplot(2,2,3)
hold on
plot(z_arr, unwrap(angle(U)));
plot(z_arr, unwrap(angle(Uin_arr)));
plot(z_arr, unwrap(angle(Us)));
plot(z_arr, n_node_arr)
legend('phase(U)','phase(Uin)','phase(Us)', 'refr. index')
title('Electric fields: phase')

subplot(2,2,2)
hold on
plot(z_arr, abs(cmu_H_arr))
```

```
plot(z_arr, abs(cmu_Hin_arr))
plot(z_arr, abs(cmu_Hs_arr))
plot(z_arr, n_node_arr)
legend('abs(c\muH)','abs(c\muHin)','abs(c\muHs)', 'refr. index')
title('Magnetic fields: magnitude')

subplot(2,2,4)
hold on
plot(z_arr, unwrap(angle(cmu_H_arr)));
plot(z_arr, unwrap(angle(cmu_Hin_arr)));
plot(z_arr, unwrap(angle(cmu_Hs_arr)));
plot(z_arr, n_node_arr)
legend('phase(c\muH)','phase(c\muHin)','phase(c\muHs)', 'refr. index')
title('Magnetic fields: phase')

figure
subplot(2,1,1)
hold on
plot(z_arr, S_arr);
plot(z_arr, Sin_arr);
plot(z_arr, Ss_arr);
legend('S','Sin','Ss');
plot(z_arr, n_node_arr);
title('Poynting vectors')

subplot(2,1,2)
hold on
plot(z_arr, X_arr);
plot(z_arr, Xin_arr);
plot(z_arr, Xs_arr);
legend('X','Xin','Xs')
title('Reactivity')

function frac_k1 = k1k2split(i, n1, n2, id_obj)
    global wid_arr DL
    if n1 == n2
        frac_k1 = 1;
    else
        wid_1 = sum(wid_arr(1:id_obj));
        wid_2 = DL*((i-1) + 1);
        excess = wid_2 - wid_1;
        if excess>DL
            disp("excess width is more than permissible")
        else
            frac_k1 = 1 - excess/DL;
        end
    end
end
```

3.6 scalar_basis.m

```
function L_coeff = scalar_basis(pt_arr)
        %L_{i}(x) is linear fn of (xi,yi), i.e.,  (ax + by + c)/k
        %this returns [a,b,c,k]

        a = pt_arr(2,2) - pt_arr(3,2);
        b = pt_arr(3,1) - pt_arr(2,1);
        c = ( pt_arr(2,1) * pt_arr(3,2) ) - ( pt_arr(3,1) * pt_arr(2,2) );
        k = a*pt_arr(1,1) + b*pt_arr(1,2) + c;

        L_coeff = [a, b, c, k];     %express basis fn as array of coeff
```

3.7 HW4_2.m

```
%set chosen points for arbitrary triangle
pt1 = [0,0];
pt2 = [1,0];
pt3 = [0,1];
pt_arr = [pt1; pt2; pt3];

%define scalar basis fn coefficient arrays
L1 = scalar_basis(pt_arr);
L2 = scalar_basis( circshift(pt_arr,2) );
L3 = scalar_basis( circshift(pt_arr,1) );

L_xy = @(x, y, L) (L(1).*x + L(2).*y + L(3)) ./ L(4); %for testing scalar basis functions
grad_L = @(L) [L(1)./L(4), L(2)./L(4), 0];    %to get gradient of L

grad_L1 = grad_L(L1);
grad_L2 = grad_L(L2);
grad_L3 = grad_L(L3);

L_arr = [L1; L2; L3];
grad_L_arr = [grad_L1; grad_L2; grad_L3];

%convention followed in transformation: pt1 -> (0,0), pt2 -> (1,0), pt3 -> (0,1)
%transformation: x = x1 + (x2-x1)u + (x3-x1)v, y = y1 + (y2-y1)u + (y3-y1)v
% (x2-x1) = L3(2), (x3-x1) = -L2(2), (y2-y1) = -L3(1), (y3-y1) = L2(1)

J = [pt2(1) - pt1(1), pt3(1) - pt1(1); pt2(2) - pt1(2), pt3(2) - pt1(2)];    % Jacobian matrix

%coordinate transformation from (x,y) to (u,v)
x_uv = @(u, v) pt_arr(1,1) + ( pt_arr(2,1) - pt_arr(1,1) ) .* u +  ( pt_arr(3,1) - pt_arr(1,1) ) .* v;
y_uv = @(u, v) pt_arr(1,2) + ( pt_arr(2,2) - pt_arr(1,2) ) .* u +  ( pt_arr(3,2) - pt_arr(1,2) ) .* v;
L_uv = @(u,v,L) L_xy( x_uv(u,v), y_uv(u,v), L );

A = zeros(3);
B = zeros(3);

y_max = @(x) 1 - x;
for i=1:3
        for j=1:3
                A(i,j) = integral2( @(u,v) ( L_uv(u,v, L_arr(i,:)) .* L_uv(u,v, L_arr(j,:)) ), 0,1, 0,y_max);
                B(i,j) = dot( grad_L_arr(i,:), grad_L_arr(j,:) ) .* 0.5;
        end
end

A = A .* det(J)
B = B .* det(J)

%validating the basis functions
x = [0:0.1:6];
y = [0:0.1:6];
[X,Y] = meshgrid(x,y);
Z1 = L_xy(X,Y,L1);
surf(X,Y,Z1);    %plot L1
hold on;
Z2 = L_xy(X,Y,L2);
surf(X,Y,Z2);    %plot L2
```

```
Z3 = L_xy(X,Y,L3);
surf(X,Y,Z3);     %plot L3
xlim([0 max(pt_arr(:,1))]);
ylim([0 max(pt_arr(:,2))]);
zlim([0 1]);
```

References

[1] M. Claudia Troparevsky, Adrian S. Sabau, Andrew R. Lupini, and Zhenyu Zhang, "Transfer-matrix formalism for the calculation of optical response in multilayer systems: from coherent to incoherent interference," Opt. Express 18, 24715-24721 (2010)