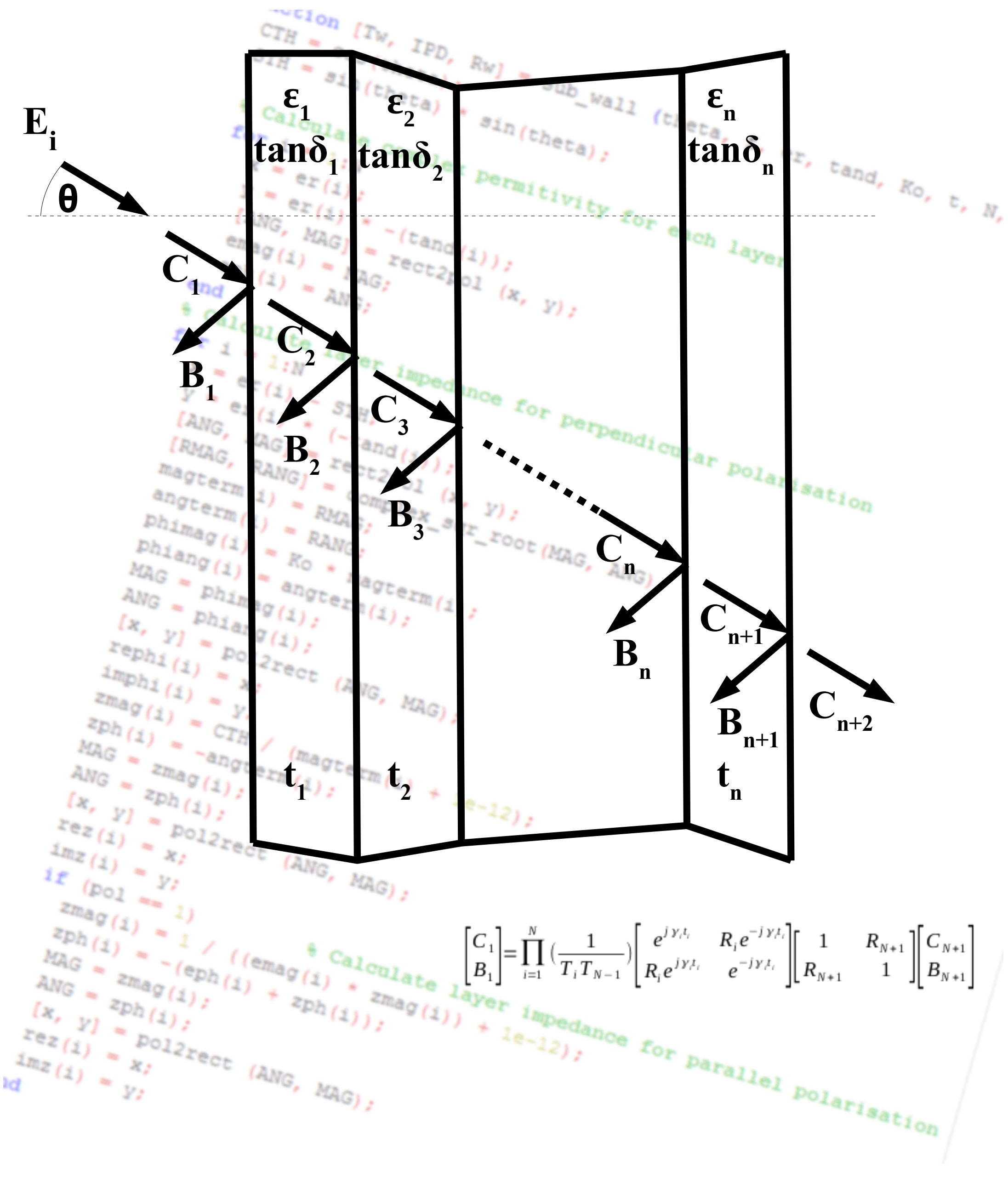
A Software Tool for the Analysis of Multilayer Radomes

Dr Paul Klasmann (2E0PMK)

November 2020

Version 1.1


E_i
θ
ε_1 $\tan\delta_1$
ε_2 $\tan\delta_2$
ε_n $\tan\delta_n$
C_1
B_1
C_2
B_2
C_3
B_3
C_n
B_n
C_{n+1}
B_{n+1}
C_{n+2}
t_1
t_2
t_n
$\begin{bmatrix} C_1 \\ B_1 \end{bmatrix} = \prod_{i=1}^{N} \left(\frac{1}{T_i T_{N-1}}\right) \begin{bmatrix} e^{j\gamma_i t_i} & R_i e^{-j\gamma_i t_i} \\ R_i e^{j\gamma_i t_i} & e^{-j\gamma_i t_i} \end{bmatrix} \begin{bmatrix} 1 & R_{N+1} \\ R_{N+1} & 1 \end{bmatrix} \begin{bmatrix} C_{N+1} \\ B_{N+1} \end{bmatrix}$

Multilayer Radome Analysis Software Instructions

This document describes the usage of the software written as a script for GNU Octave but should also work with Matlab which will predict the transmission, reflection and insertion phase delay performance when a perpendicular or parallel polarized wave is incident upon a single or multilayer radome such as an A or C-sandwich radome, or a dielectric stack-up of any number of layers. It is assumed that the dielectric layers are homogenous and isotropic in nature as shown in Figure 1.

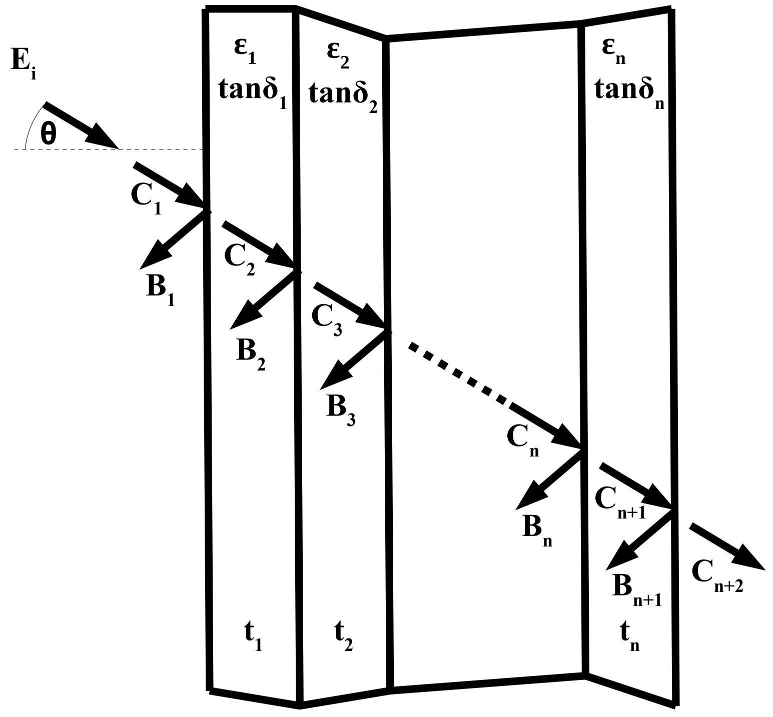


Figure.1

The analytical software is based on the boundary value solution of an N-layer dielectric wall using the Fresnel equations. This is described in the book “Analysis of Radome-Enclosed Antennas”, Dennis Kozakoff (Artech House, 1997), and will be repeated here for completeness. If the forward and reverse propagating waves are C$_i$ and B$_i$ respectively, the solution is:

$$\begin{bmatrix} C_1 \\ B_1 \end{bmatrix} = \prod_{i=1}^{N} \left(\frac{1}{T_i T_{N-1}}\right) \begin{bmatrix} e^{j\gamma_i t_i} & R_i e^{-j\gamma_i t_i} \\ R_i e^{j\gamma_i t_i} & e^{-j\gamma_i t_i} \end{bmatrix} \begin{bmatrix} 1 & R_{N+1} \\ R_{N+1} & 1 \end{bmatrix} \begin{bmatrix} C_{N+1} \\ B_{N+1} \end{bmatrix} \qquad (1)$$

where,

t_i	= Layer thickness of the ith layer.
R$_i$, T$_i$	= Fresnel reflection and transmission coefficients of the ith layer, respectively.
γ_i	= Propagation constant in the ith layer.

Angle of incidence θ is relative to the normal of the radome surface.

$$\gamma_i = k_0 \sqrt{\varepsilon_{ri} - \sin^2(\theta)} \tag{2}$$

The permittivity of the ith layer is given by ε_i and k_0 is the wave number in a vacuum. Permittivity is given by:

$$\varepsilon_{ri} = \varepsilon'_{ri}(1 - j\tan(\delta_i)) \tag{3}$$

The Fresnel transmission and reflection coefficients are calculated from the following:

$$R_i = \frac{Z_i - Z_{i-1}}{Z_i + Z_{i-1}}; T_i = 1 - R_i \tag{4}$$

The wave impedance calculated depends on the polarization of the incident wave. For perpendicular polarization we have:

$$Z_i = \frac{\cos(\theta)}{\sqrt{\varepsilon_{ri} - \sin^2(\theta)}} \tag{5}$$

For parallel polarization we have:

$$Z_i = \frac{\sqrt{\varepsilon_{ri} - \sin^2(\theta)}}{\varepsilon_{ri}\cos(\theta)} \tag{6}$$

When the matrix (1) is multiplied out, the following is obtained:

$$\begin{bmatrix} C_1 \\ B_1 \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} C_{N+2} \\ B_{N+2} \end{bmatrix} \tag{7}$$

The voltage reflection coefficient from the radome’s front surface is:

$$R_w = \frac{B_1}{C_1} | (B_{N+2} = 0) = \frac{A_{21}}{A_{11}} \tag{8}$$

The voltage transmission coefficient is:

$$T_w = \frac{C_{N+2}}{C_1} | (B_{N+2} = 0) = \frac{1}{A_{11}} \tag{9}$$

The reflection and transmission coefficients are both complex numbers, therefore it can be expressed as a magnitude and insertion phase delay (IPD), this is an angle. The radome transmission efficiency is:

$$|T_w|^2 \tag{10}$$

These equations are shown for completeness so that the program can be more easily followed for those who do not have access to "Analysis of Radome-Enclosed Antennas". Buy the book if it's available, I thoroughly recommend it!

The program is written in GNU Octave which should be compatible with Matlab. The master file calls a number of functions which are saved in separate files and must be present in the same folder/directory as the main function or any other function that calls them. An exhaustive description of the code will not be given but the complete listing of all functions is given at the end of the document and important sections are commented.

It is necessary to specify the problem with only changing these lines:

```
% Enter user variables here...
N = 5;                                    % Enter number of layers
fl = 17;                                  % Enter lower frequency in GHz
fh = 32;                                  % Enter upper frequency in GHz
no_of_freq_points = 301;                  % Enter No. of frequency points
no_inc_ang = 3;                           % No. of incidence angles
ang_inc_step = 15;                        % Increment angle of incidence
% Enter the dielectric contatant/permitivity for each layer
er1 = 4.0;  er2 = 1.1;  er3 = 4.0;  er4 = 1.1;  er5 = 4.0;
% Enter the loss tangent/tand for each layer
tand1 = 0.003;  tand2 = 0.001;  tand3 = 0.003;  tand4 = 0.001;  tand5 = 0.003;
% Enter the layer thickness in mm for each layer
t1 = 0.24;  t2 = 2.1;  t3 = 0.48;  t4 = 2.1;  t5 = 0.24;

er = [er1, er2, er3, er4, er5]                % Enter dielectric constants
tand = [tand1, tand2, tand3, tand4, tand5]    % Enter tand values (loss tangent)
t = [t1, t2, t3, t4, t5]                      % Enter layer thickness in mm
% End of user variables
```

Values are entered as a vector (one dimensional array) in the order of dielectric layers. Add or remove layers as necessary. The above code is for a five layer stack-up.

If you require more or less plots for differing angles of incidence, set the "no_inc_ang", increments can be set with "ang_inc_step".

Note that this version of the program is written in a traditional program flow with for loops defined where necessary. It would be far more efficient to perform calculations using native complex number notation and matrix operations instead of converting numbers from Cartesian to polar form and vice versa and to use matrix operations. This may be done at a future date.

When the program is run, six plots will be generated. These are the reflection, transmission and insertion phase delay for both perpendicular and parallel polarization of the incident E-field. The code can be modified by the user to generate only what plots are necessary.

Comparisons between Analytic Software and CST FEM Solver

To confirm that the analytical software is giving the correct result, two benchmarks are presented below with equivalent simulations using CST Microwave Studio and its FEM solver with unit cell boundaries. Although unit cell FEM simulations may take a couple of minutes, the analytical method takes just seconds to complete and generate plots with the option for several angle of incidences. The results below are for 0° incidence.

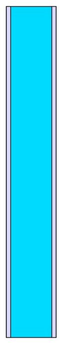
A-Sandwich Radome (3-layer stackup)

A-Sandwich Sample

Outer Layers: plain weave GFRP, thickness 0.24mm, $\varepsilon_r = 4.0$.

Core Layer: Millifoam RHC71, thickness 2.5mm, $\varepsilon_r = 1.1$.

Values of tanδ are estimated to be 0.003 for GFRP, and 0.001 for the Millifoam, and these values were used for the analytical and numerical simulations for a like for like comparison.



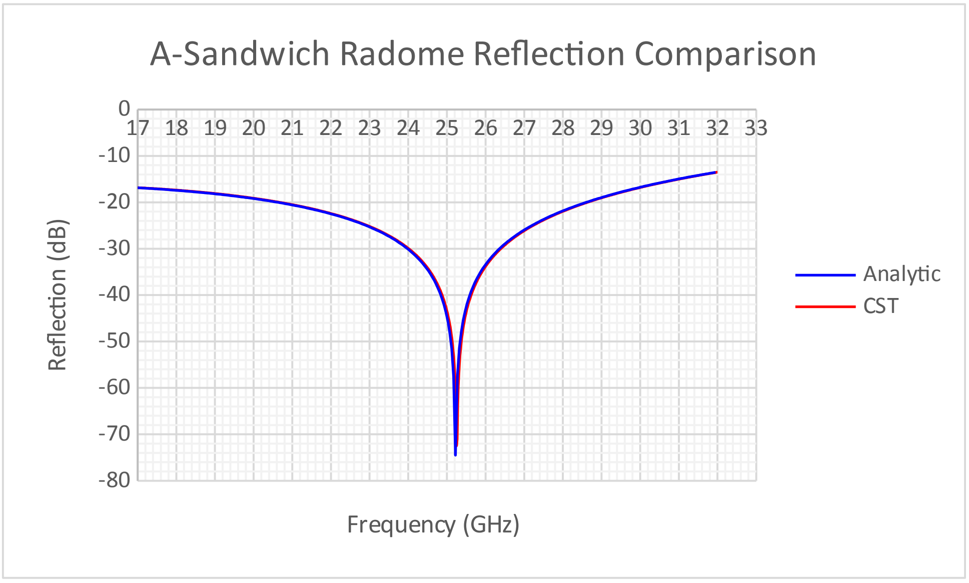


Figure. 2

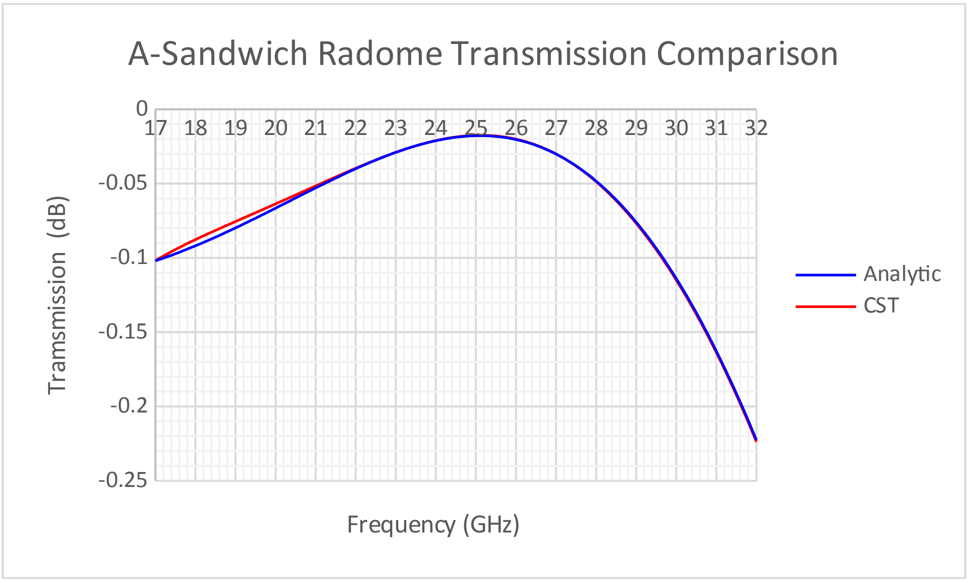


Figure. 3

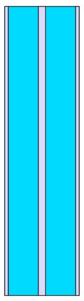
C-Sandwich Radome (5-layer stackup)

C-Sandwich Sample

Outer Layers: plain weave GFRP, thickness 0.24mm, $\varepsilon_r = 4.0$.

Core Layers: Millifoam RHC71, thickness 2.1mm, $\varepsilon_r = 1.1$.

Middle Layer: plain weave GFRP, thickness 0.48mm, $\varepsilon_r = 4.0$.



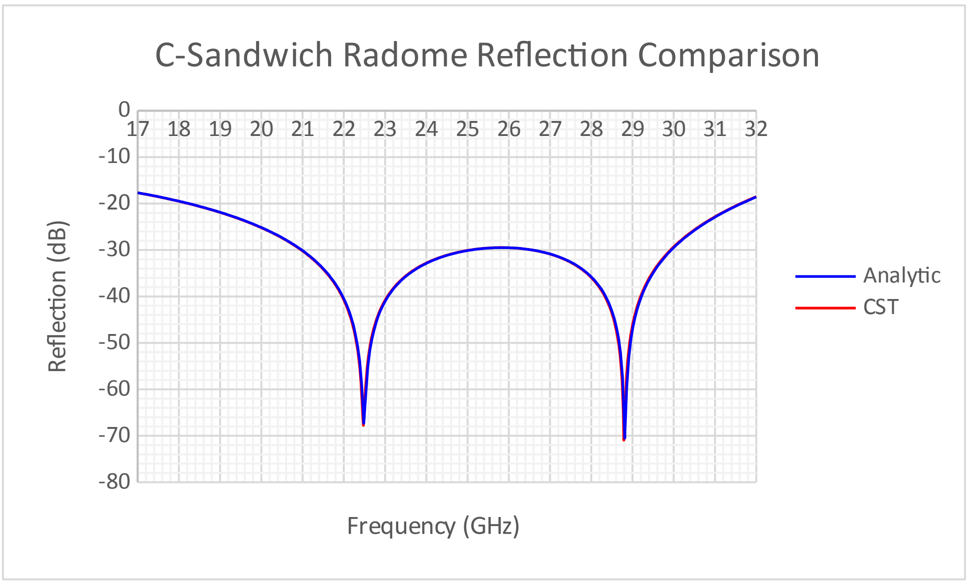


Figure. 4

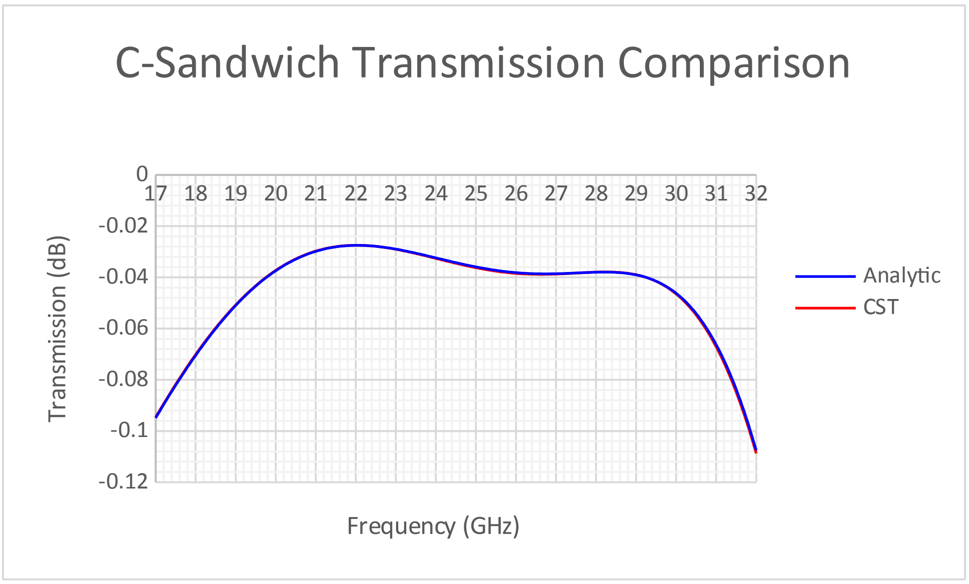


Figure. 5

GNU Octave Code

```
% Program to calculate Transmission Loss, Insertion Phase Delay and Reflection
% for a multilayer radome.
% This program is a translation from 'Analysis of Radome-Enclosed Antennas'
% by Dennis J. Kozakoff, 1997, Artech House.
%
% Author: Dr Paul Klasmann
% Date: 05/11/2020      Version 1.1
%
% 1) Set N to equal the mumber of layers.
% 2) Set fl and fh to the lower and upper analysis frequency in GHz.
% 3) Set no_of_freq_points to a sensible value for the frequency range.
% 4) Setup the array values for the relative dielectric constant (er).
% 5) Setup the array values for the loss tangents (tand).
% 6) Setup the array values for the radome layer thicknesses (t) in mm.

close all
clear
clc
% Enter user variables here...
N = 5;                                    % Enter number of layers
fl = 17;                                  % Enter lower frequency in GHz
fh = 32;                                  % Enter upper frequency in GHz
no_of_freq_points = 301;                  % Enter No. of frequency points
no_inc_ang = 3;                           % No. of incidence angles
ang_inc_step = 15;                        % Increment angle of incidence
% Enter the dielectric contatant/permitivity for each layer
er1 = 4.0;  er2 = 1.1;  er3 = 4.0;  er4 = 1.1;  er5 = 4.0;
% Enter the loss tangent/tand for each layer
tand1 = 0.003;  tand2 = 0.001;  tand3 = 0.003;  tand4 = 0.001;  tand5 = 0.003;
% Enter the layer thickness in mm for each layer
t1 = 0.24;  t2 = 2.1;  t3 = 0.48;  t4 = 2.1;  t5 = 0.24;

er = [er1, er2, er3, er4, er5]                % Enter dielectric constants
tand = [tand1, tand2, tand3, tand4, tand5]    % Enter tand values (loss tangent)
t = [t1, t2, t3, t4, t5]                      % Enter layer thickness in mm
% End of user variables

finc = (fh-fl)/no_of_freq_points;      % Frequency increment
eo = 8.854e-12;                        % Permativity of free space (F/m)
RAD = pi/180;                          % For degree to radians conversion

% Begin Calculation
for pol = 0:1              % Pol=0 for Perpendicular Polarization
  array_index = 0;         % Pol=1 for parallel polarization
  for f = fl:finc:fh
    array_index = array_index + 1;
    LAM = 300/f;                   % Lambda is wavelength in mm
    Ko = (2*pi/LAM);               % Ko is wavenumber in mm^(-1)
    ANGLE = 0;                     % Initialize first incident angle to calculate
        for angle_index = 1:no_inc_ang;
        theta = ANGLE * RAD;       % Theta is angle of incidence in radians
        % Call function sub_wall to perform main calculations
        [Tw, IPD, Rw] = sub_wall(theta, f, er, tand, Ko, t, N, pol);
        TwdB(array_index, angle_index) = 20 * log10(Tw);
        IPDdeg(array_index, angle_index) = -IPD / RAD;
        RwdB(array_index, angle_index) = 20 * log10(Rw);
        if(IPDdeg(array_index, angle_index) == -360)
          IPDdeg(array_index, angle_index) = 0;
        end
        ANGLE = ANGLE + ang_inc_step;        % Increment angle of incidence
```

```matlab
      end  % END ANGLE LOOP
  end      % END f LOOP

f = fl:finc:fh;
% Transmission Loss for perpendicular polarisation
if (pol == 0)
  % Plot TL for perpendicular polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, TwdB(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Transmission Loss for Perpendicular Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Transmission Loss (dB)','FontSize', 14);
  drawnow

  % Plot IPD for perpendicular polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, IPDdeg(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Insertion Phase Delay for Perpendicular Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Insertion Phase Delay (degrees)','FontSize', 14);
  drawnow

  % Plot Reflection for perpendicular polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, RwdB(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Reflection for Perpendicular Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Return Loss (dB)','FontSize', 14);
  drawnow

% Transmission Loss for Parallel Polarisation
elseif pol == 1
  % Plot TL for parallel polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, TwdB(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Transmission Loss for Parallel Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Transmission Loss (dB)','FontSize', 14);
  drawnow

  % Plot IPD for parallel polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, IPDdeg(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Insertion Phase Delay for Parallel Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Insertion Phase Delay (degrees)','FontSize', 14);
  drawnow

  % Plot Reflection for parallel polarisation
  figure
  for angle_index = 1:no_inc_ang;
  plot(f, RwdB(1:end,angle_index))
  hold on;
  end
  grid on
  set(gca,'linewidth',2, 'fontsize', 14, 'XTick', fl:1:fh)
  legendtext = ['0 degrees'; '15 degrees'; '30 degrees'; '45 degrees'; '60 degrees';
'75 degrees'];
  legend( legendtext, 'location', 'southeast' );
  title('Reflection for Parallel Polarization','FontSize', 16);
  xlabel('Frequency (GHz)','FontSize', 14);
  ylabel('Return Loss (dB)','FontSize', 14);
  drawnow

end
end         % END pol LOOP
```

Function sub_wall

```
function [Tw, IPD, Rw] = sub_wall (theta, f, er, tand, Ko, t, N, pol)
  CTH = cos(theta);
  STH = sin(theta) * sin(theta);

  % Calculate complex permitivity for each layer
  for i = 1:N
    x = er(i);
    y = er(i) * -(tand(i));
    [ANG, MAG] = rect2pol (x, y);
    emag(i) = MAG;
    eph(i) = ANG;
  end
  % Calculate layer impedance for perpendicular polarisation
  for i = 1:N
    x = er(i) - STH;
    y = er(i) * (-tand(i));
    [ANG, MAG] = rect2pol (x, y);
    [RMAG, RANG] = complex_sqr_root(MAG, ANG);
    magterm(i) = RMAG;
    angterm(i) = RANG;
    phimag(i) = Ko * magterm(i);
    phiang(i) = angterm(i);
    MAG = phimag(i);
    ANG = phiang(i);
    [x, y] = pol2rect (ANG, MAG);
    rephi(i) = x;
    imphi(i) = y;
    zmag(i) = CTH / (magterm(i) + 1e-12);
    zph(i) = -angterm(i);
    MAG = zmag(i);
    ANG = zph(i);
    [x, y] = pol2rect (ANG, MAG);
    rez(i) = x;
    imz(i) = y;
    if (pol == 1)          % Calculate layer impedance for parallel polarisation
      zmag(i) = 1 / ((emag(i) * zmag(i)) + 1e-12);
      zph(i) = -(eph(i) + zph(i));
      MAG = zmag(i);
      ANG = zph(i);
      [x, y] = pol2rect (ANG, MAG);
      rez(i) = x;
      imz(i) = y;
    end
  end
% End of layer impedance calculation
  zmag(N+1) = 1;
  zph(N+1) = 0;
  rez(N+1) = 1;
  imz(N+1) = 0;
% Do next loop i = 1 serperately from the loop to include z for space before radome
% because indexing starts at 1 and not 0. Then proceed with loop starting i=2
  i = 1;
  x = rez(i) - 1;
  y = imz(i) - 0;
  [ANG, MAG] = rect2pol (x, y);
  NUMMAG = MAG;
  NUMANG = ANG;
  x = rez(i) + 1;
  y = imz(i) + 0;
  [ANG, MAG] = rect2pol (x, y);
  DENMAG = MAG;
  DENANG = ANG;
  RMAG(i) = NUMMAG / DENMAG;
  Rph(i) = NUMANG - DENANG;
  MAG = RMAG(i);
  ANG = Rph(i);
  [x, y] = pol2rect (ANG, MAG);
  reR(i) = x;
  imR(i) = y;
  reT(i) = 1 + reR(i);
  imT(i) = imR(i);
  x = reT(i);
  y = imT(i);
  [ANG, MAG] = rect2pol (x, y);
  Tmag(i) = MAG;
  Tph(i) = ANG;

% Continue from i=2
  for i = 2:(N+1)
    x = rez(i) - rez(i-1);
    y = imz(i) - imz(i-1);
    [ANG, MAG] = rect2pol (x, y);
    NUMMAG = MAG;
    NUMANG = ANG;
    x = rez(i) + rez(i-1);
    y = imz(i) + imz(i-1);
    [ANG, MAG] = rect2pol (x, y);
    DENMAG = MAG;
    DENANG = ANG;
    RMAG(i) = NUMMAG / DENMAG;
    Rph(i) = NUMANG - DENANG;
    MAG = RMAG(i);
    ANG = Rph(i);
    [x, y] = pol2rect (ANG, MAG);
    reR(i) = x;
    imR(i) = y;
    reT(i) = 1 + reR(i);
    imT(i) = imR(i);
    x = reT(i);
    y = imT(i);
    [ANG, MAG] = rect2pol (x, y);
    Tmag(i) = MAG;
    Tph(i) = ANG;
  end

% Matrix multiplications begin here
Amag(1) = exp(-imphi(1) * t(1));
Amag(4) = 1 / Amag(1);
Amag(2) = RMAG(1) * Amag(4);
Amag(3) = RMAG(1) * Amag(1);
Aph(1) = rephi(1) * t(1);
Aph(2) = Rph(1) - Aph(1);
Aph(3) = Rph(1) + Aph(1);
Aph(4) = -Aph(1);

for K = 2:N
  Bmag(1) = exp(-imphi(K) * t(K));
  Bmag(4) = 1 / Bmag(1);
  Bmag(2) = RMAG(K) * Bmag(4);
  Bmag(3) = RMAG(K) * Bmag(1);
  Bph(1) = rephi(K) * t(K);
  Bph(2) = Rph(K) - Bph(1);
  Bph(3) = Rph(K) + Bph(1);
  Bph(4) = -Bph(1);
  [Amag, Aph] = cmult(Amag, Aph, Bmag, Bph);
end

Bmag(1) = 1;
Bmag(4) = 1;
Bmag(2) = RMAG(N + 1);
Bmag(3) = Bmag(2);
Bph(1) = 0;
Bph(2) = Rph(N + 1);
Bph(3) = Bph(2);
Bph(4) = 0;
[Amag, Aph] = cmult(Amag, Aph, Bmag, Bph);
tranmag = 1;
tranph = 0;

for j = 1:N +1
  tranmag = tranmag * Tmag(j);
  tranph = tranph + Tph(j);
end

tranmag = 1 / (tranmag + 1e-12);

while (tranph > (2 * pi))
    tranph = tranph - (2 * pi);
endwhile

tranph = -tranph;
Tw = 1 / (tranmag * Amag(1));
Ttotph = -(tranph + Aph(1));
Rw = Amag(3) / Amag(1);
SUM = 0;
for j = 1:N
  SUM = SUM + t(j);
end

% Get IPD into correct quadrant
do
  if(Aph(1) < (CTH * Ko * SUM))
    Aph(1) = Aph(1) + (2 * pi);
  end
until (Aph(1) >= (CTH * Ko * SUM))

IPD = Aph(1) - (CTH * Ko * SUM);
endfunction
```

Function cmult

```
function [Amag, Aph] = cmult(Amag, Aph, Bmag, Bph)
  for i = 1:2:3
    for j = 1:2
      R1 = Amag(i) * Bmag(j);
      R2 = Amag(i + 1) * Bmag(j + 2);
      E1 = Aph(i) + Bph(j);
      E2 = Aph(i + 1) + Bph(j + 2);
      R3 = (R1 * cos(E1)) + (R2 * cos(E2));
      E3 = (R1 * sin(E1)) + (R2 * sin(E2));
      Cmag(j + i - 1) = sqrt((R3*R3) + (E3*E3));
      if((R3 > 0) && (E3 > 0))
        PA = 0;
      elseif(R3 < 0)
        PA = pi;
      elseif((R3 > 0) && (E3 < 0))
        PA = 2 * pi;
      end
      Cph(j + i - 1) = atan(E3 / (R3 + 1e-12)) + PA;
    end
  end

  for i = 1:4
    Amag(i) = Cmag(i);
    Aph(i) = Cph(i);
  end

endfunction
```

Function complex_sqr_root

```
function [RMAG, RANG] = complex_sqr_root(MAG, ANG)
  RMAG = sqrt(MAG);
  RANG = ANG / 2;
endfunction
```

Function pol2rect

```
function [x, y] = pol2rect (ANG, MAG)

  x = MAG * cos(ANG);
  y = MAG * sin(ANG);

endfunction
```

Function rect2pol

```
function [ANG, MAG] = rect2pol (x, y)

  MAG = sqrt((x^2) + (y^2));

  if ((x>0) && (y>0))
    corr = 0;
  elseif ((x<0) && (y<0))
    corr = pi;
  elseif ((x>0) && (y<0))
    corr = 0;
  elseif (x<0) && (y>0)
    corr = pi;
  elseif (x>0) && (y==0)
```

```
    corr = 0;
  elseif (x<0) && (y==0)
    corr = pi;
  elseif (x==0) && (y!=0)
    corr = 0;
  elseif (x==0) && (y==0)
    ANG = 0;
    return;
  end

  ANG = atan(y / (x + 1e-12)) + corr;

endfunction
```

END OF DOCUMENT