Table of Contents


```
function Abbas_HW_11_Parabolic_equation_modifications_final()

% Hasan Tahir Abbas
% ECEN 637
% Homework 11: Study on Wave Propagation in the Troposphere
% 12/12/2015
%
%
%  This code is a translation of the Fortran code provided in class
%  simulating:
% %  Dockery, G. D., “Modeling Electromagnetic Wave Propagation in the
% Troposphere Using the Parabolic Equation,” IEEE Trans. Antennas
 Propag.
% 36(10), 1464–1470 (1988).
%
%
% Taylor-pattern source
% FFT of the signal
% Far-field profile
% Utilization of Toeplitz nature of the impedance matrix (Only one row
% needs to be computed)
%
% PARAMETERS AND VARIABLES LIST
% -----------------------------
% N = Spectral Length
% Nx Spatial Length
% p = p_space axis
```

```
% z = z_space axis
% pat_p = Pattern in p-space
% f_z = Profile in z-space
% Picture = Field profile in the environment
% x = Spatial iteration
% a = radius of the earth
% m = dielectric of the space
% epsi_o = Free-space permittiivty
% index = Refractive index of the medium
% epsi_c = Complex dielectric of the medium
% dNdz = Graded duct gradient
% zmax = medium change locations in space
% xx = Horizontal x-axis profile positions
% choice = choice of medium
% p_max = maximum value in p-space
% z_max = maximum value in z-space
% dp = Delta increment in p-direction
% dz = Delta increment in z-direction
% pe = beam tilt angle
% za = height of the source
% dx = spatial intervla
% a_e = effective radius of the earth
% pattern = p_space pattern
% fz = source signal
% temp = temporary variable
% freq = source frequency
% lambda = wavelength of the source
% c = speed of light
% kappa = propagation constant
% theta_max = Beam direction of the source
% beamwidth = beamwidth of the Taylor-patterened source
% Length = length of the source
% epsi_r = relative permittivity of sea water
% sigma = conductivity
% omega = angular frequency
%
% FUNCTIONS LIST
% -----------------------------
% DATA() = initializes the variables and arrays to be used in the code
% REFR_INDEX_N() = models the refractive index of the different cases
 of
%                  atmosphere
% GEN_SOURCE() = Taylor-pattern based source
% P_SPACE() = Pattern in the p-space by FFT
% DRAW_DATA() = Arrange pattern in p-space
% Z_SPACE() = Profile in the z-space by IFFT
% REPLACE() = Choose signal for FFT
% PRE_PROCESS() = Take care of marching to the next step
% POST_PROCES() =  Refractive Index difference corretion and Hanning
 window
% MAX_FZ() = Find maximum value of field distribution
% PLOTS() = Plots results

% -----------------------------
```

```
close all;clear all; clc
% ************************
```

Parameters

****

Global variables are used to span across all the functions in this code According to MATLAB's documentation, a better and safer option will be persistent type variables

```
global Nx
global f_z Picture x
% ************************
% ************************
%###################################
```

Main Program

*********************** *********************** ***********************

```
data();
gen_source();
for x = 1 : Nx
        refr_index_n();
        p_space();
        z_space();
        Picture(:,x) = f_z;
% plots();
end
plots();
save var1.mat

end
%###################################
```

End Main Program

*********************** *********************** ***********************

Initialize

*********************** ***********************

```
function data()
% Set all the variables to zero

global N Nx
global p_max z_max dp dz
global pe za dx a a_e
global p z pat_p f_z
global pattern fz temp m
global freq lambda c kappa  theta_max
```

```
global beamwidth Length han
global epsi_o epsi_r sigma omega epsi_c Picture
global xx dxx

% Data to be used
% ************************
N = 16384;
Nx = 800;
% Nx = 1024;
freq = 3e9 ; % Antenna operation frequency

c = 3e8;
epsi_o = 8.854e-12;
epsi_r = 69;
sigma = 6.5;
omega = 2*pi*freq;
epsi_c = epsi_o * epsi_r + 1i* sigma/omega;

lambda = c/freq; % One wavelength
kappa = 2*pi/lambda;
Length = 38*lambda; % Length of aperture in wavelengths

za = 31;
dx = 200;
a = 6.37e6; % % Radius of the earth
a_e = a*(4/3); % effective radius of the earth

theta_max = 13;
beamwidth = 22.2718567e-3; % Half-power beamwidth of Taylor pattern
p_max = kappa*sind(theta_max); % Maximum propagation angle %% sind()
 for degree based arguments
dp = 2*p_max/(N-1);
dz = 1/(p_max/pi); % Delta increment in z-direction
z_max = (N-1)/2*dz; % Physical area analyzed
pe = kappa*sin(beamwidth/2); % Beam tilt angle

z = dz*(0:N-1) - z_max;
p = dp*(0:N-1) - p_max;
xx = (1:Nx)*dx; % Horizontal x-axis profile positions

pattern = zeros(1,N);
fz = zeros(1,N);
temp = zeros(1,N);
m = zeros(1,N);
pat_p = zeros (1,N);
f_z = zeros (1,N);
Picture = zeros(N,Nx);
han = zeros (1,N);
dxx = (0:Nx-1)*dx;

end
% ************************
% ************************
%
```

Get the refractive index for further calculation

*********************** ***********************

```
function refr_index_n()

global N a z m

global epsi_o index epsi_c
global dNdz zmax
global xx x choice

if x == 1
    disp('        Choose the media           ');
    disp('                                   ');
    disp('                                   ');
    disp('1. Free space');
    disp('2. for Standard Atmosphere');
    disp('3. for Surface Duct');
    disp('4. for Graded Duct');
    prompt = 'Enter option from 1-4:        ';
    choice = input(prompt);
end

% Set the environment

switch (choice)
```

Free-space with flat-earth

```
    case(1)

        if x == 1
            disp('           Choice = 1                  ');
        end
        for i = 1 : N
            if z(i) < 0 % Sea Water
                m(i) = (epsi_c)/(epsi_o) - 1;
            else
                index = 1; %  Flat earth condition (2*z/a = 0.)
                m(i) = (index^2 - 1);
            end
        end
```

Troposphere condition

```
    case(2)

        if x == 1
            disp('           Choice = 2                  ');
        end

        for  i = 1 : N
```

```
            if z(i) < 0 % Sea Water
                m(i) = (epsi_c)/(epsi_o) - 1;
            else
                index = 1 + (300 - 0.0394 * z(i)) * 1e-6; % !Standard
atmosphere condition
                m(i) = index^2 - 1 + 2*z(i)/a;
            end
        end
```

Surface Duct condition

```
    case(3)

        if x == 1
            disp('           Choice = 3                  ');
        end
        for i = 1 : N
            if (z(i) < 0) % Sea water
                m(i) = epsi_c/epsi_o - 1;
            elseif (z(i) > 0 && z(i) <= 37) % Surface Duct
                index = 1 + (300 - .5*z(i))*1e-6;
                m(i) = index^2 - 1 + 2*z(i)/a;
                k = i;
            else
                index = 1 + (300 - .5*z(k) - .0394*( z(i) -
z(k)))*1e-6; % Standard atmosphere
                m(i) = index^2 - 1 + 2*z(i)/a;
            end
        end
```

Graded Surface Duct Condition

```
    case(4)
        if x == 1
            disp('           Choice = 4                  ');
        end
        if ( xx(x) <= 40000)
            dNdz = (.5 - .167) / 40000 * xx(x) - .5;
            zmax = (150 - 37) / 40000 * xx(x) + 37;
        end
        for i = 1 : N
            if (z(i) < 0) % Sea water
                m(i) = epsi_c/epsi_o - 1;
            elseif (z(i) > 0 && z(i) <= zmax) % Graded surface Duct
                index = 1 + (300 + dNdz * z(i)) * 1e-6;
                m(i) = index^2 - 1 + 2 * z(i) / a;
                k = i;
            else
                index = 1 + (300 + dNdz * z(k) - .0394 * (z(i) -
z(k))) * 1e-6; % Standard atmosphere
                m(i) = index^2 - 1 + 2 * z(i) / a;
            end
        end
```

```
    otherwise

        disp('%%%%%ERROR%%%%%%');
        disp('Wrong choice entered');
        return;
end

end
% ************************
% ************************
%
% ************************
```

Generate the source

*********************** ***********************

```
function gen_source()

% The aperture distribution and its far field pattern

global N z Length
global pe za fz f_z zz


% global pattern pat_p  p
% global Picture x
```

Sinc Source

x = 0; for i = 1 : N pp = p(i) - pe; if pp == 0 pattern(i) = Length; else pattern(i) = 2*sin(pp*Length/2)/pp*exp(-1i*pp*za); end pat_p(i) = abs(pattern(i)); end

Create Taylor Line pattern

SLL = -20 dB n_bar = 10; ! Taylor pattern SLL=-20dB, n_bar=10

```
for  k = 1 : N
    if  z(k) >= (za - Length/2) && z(k) <= (za + Length/2)
            zz = z(k) - za;
            fz(k) = exp(1i*pe*z(k))...
            *( 1.0 ...
            + 2.0*( 0.0977020907)*cos(2*pi*1.0*zz/Length)...
            + 2.0*( 0.045938932)*cos(2*pi*2.0*zz/Length)...
            + 2.0*(-0.0567750651)*cos(2*pi*3.0*zz/Length)...
            + 2.0*( 0.0543140899)*cos(2*pi*4.0*zz/Length)...
            + 2.0*(-0.0472939433)*cos(2*pi*5.0*zz/Length)...
            + 2.0*( 0.0381149255)*cos(2*pi*6.0*zz/Length)...
            + 2.0*(-0.0279769765)*cos(2*pi*7.0*zz/Length)...
            + 2.0*( 0.0177444933)*cos(2*pi*8.0*zz/Length)...
            + 2.0*(-0.0081655839)*cos(2*pi*9.0*zz/Length))*(1.0/
Length);
```

```
    else
        fz(k) = 0.0;
    end

end
f_z = abs(fz(k));

end
% ************************
% ************************
%
```

Fourier Transform of known aperture distribution

```
*********************** ***********************

function p_space()


global N  dz temp
global pattern  ii

ii = -1;
replace(); % Store data in temporary array for calculation
temp = fft(temp,N);
pattern = temp*dz;
draw_data();
% save pattern.mat
end

% ************************
% ************************
%
```

Store the data for Plotting in p-space

```
*********************** ***********************

function draw_data()

global N pattern pat_p

for  i = 1 : N
    if ( i <= floor(N/2+1))
        k = i + floor(N/2) - 1;
        pat_p(k) = abs(pattern(i));
    else
        k = i - floor(N/2) - 1;
        pat_p(k) = abs(pattern(i));
    end
end
```

```
end
% ************************
% ************************
%
```

Inverse Fourier Transform

*********************** ***********************

```
function z_space()
% Take inverse fourier transform of the previous pattern at the next
 step

global N dz temp
global f_z fz ii fmax
global Picture x
ii = 1;
replace();  % Store data in temporary array
pre_process();  % Take care of marching
temp = N*ifft(temp,N);
post_process(); % Take care of index difference in z-direction
max_fz();

fz = temp/(N*dz);
f_z = abs(fz)/fmax;
Picture(:,x) = f_z;
% save z_space.mat
end
```

Store the data for calculation

*********************** ***********************

```
function replace()

global pattern fz ii temp

if ii == 1
    temp = pattern;
else
    temp = fz;
end % end if

% save replace.mat
end % end function
% ************************
% ************************
%
```

Marching

*********************** ***********************

```
function pre_process()
% Take care of marching to the next step
global N dp dx pp kappa temp

for i = 1 : N
    if i <= (N/2 + 1)
        pp = dp*(i-1);
    else
        pp = dp*(i-N-1);
    end
    temp(i) = temp(i) * exp(-1i*pp^2 * dx/(2 * kappa));
end
end % end function
% *************************
% *************************
%
```

Refractive Index difference corretion and Hanning window

```
*********************** ***********************

function post_process()
% Take care of refractive index difference in z-direction
global z_max temp
global za dx z m kappa han

han = .5 + .5 * cos( 2*pi*(z - za)/(2*z_max));
temp = temp.* exp(1i* kappa * m * dx/2) .* han;

end % end function
% *************************
% *************************
%
```

Maximum of field distribution

```
*********************** ***********************

function max_fz()
%
% Find maximum value of field distribution
global N dz fmax temp

fmax = 0;
for i = 1 : N
    ftemp = abs(temp(i))/(N*dz);
    if ftemp >= fmax
        fmax = ftemp;
%     else
%         fmax = fmax;
    end
```

```
end
end % end function
% ************************
% ************************
%
```

Plotting routine

```
*********************** ***********************

function plots()

global Picture choice
global p pat_p z f_z dxx


% Plot Input Admittance
figure(1);
H = plot(pat_p, p);
H.Color = 'black';
H.LineWidth = 1.4;
title('Plot in p-space','Interpreter','latex')
set(gcf,'Color','white'); % Set background color to white
set(gca,'FontName','times new roman') % Set axes fonts to Times New
 Roman
xlabel('Magnitude ','Interpreter','latex'); % X-axis label
ylabel('$p = k \times sin(\theta) |\theta|<=13^\circ
$','Interpreter','latex'); % y-axis label
grid on
% cleanfigure();
% matlab2tikz('filename',sprintf('ECEN637_HW11_p_space_choice_%d.tex',
 choice));
%
%
%
figure(2)
H = plot(20*log10(f_z),z);
H.Color = 'black';
H.LineWidth = 1.4;
ylim([0 500]);
title('Plot in z-space','Interpreter','latex')
set(gcf,'Color','white'); % Set background color to white
set(gca,'FontName','times new roman') % Set axes fonts to Times New
 Roman
xlabel('Normalized Power $(dB)$','Interpreter','latex'); % X-axis
 label
ylabel('Altitude $(m)$','Interpreter','latex'); % y-axis label
grid on
% cleanfigure();
% matlab2tikz('filename',sprintf('ECEN637_HW11_z_space_plot_choice_
%d.tex', choice));
%
```

```
%
%
figure(3)
surf(dxx,z,Picture);
shading interp;
ylim([-50 300]);
view([0 90]);
colormap jet
title('Spatial Field Distribution','Interpreter','latex')
set(gcf,'Color','white'); % Set background color to white
set(gca,'FontName','times new roman') % Set axes fonts to Times New
 Roman
xlabel('Distance $(m)$','Interpreter','latex'); % X-axis label
ylabel('Altitude $(m)$','Interpreter','latex'); % y-axis label
% cleanfigure();
% matlab2tikz('filename',sprintf('ECEN637_HW11_E_space_plot_choice_
%d.tex', choice));
% %
%
end
```

Published with MATLAB® R2015b