Appendix – D MATLAB Code

D.1 Main code including all function to trace a single ray (final_a.m)

```
%% Ray Tracing Algorithm
% the main program
% Different function can be used for tracing of the ray from this main
clc;
clear all;
disp ('Program is on');    % The welcome message
tic % time to calculate result
%% ---Input 1 ----- Geometric input condition
disp ('Enter Geometric data according to following arguments')
TL = 5;%input('Enter Top Surface length in mtr = ');
% TL =length of top surface in mtr (X direction)
TW = 5;%input('Enter Top Surface width in mtr = ');
% TW=width of top surface in mtr (Z direction)
BL = 2;%input('Enter Bottom Surface length in mtr = ');
% BL=length of bottom surface in mtr (X direction)
BW = 2; %input('Enter Top Surface width in mtr = ');
%BW=length of bottom surface in mtr (Z direction)
HE=2;
n = 3; %input('Enter number of nodes on each side = ');
% n=enter number of node on x or z axis
RA = 60; %input('Enter angle made by reflector with horizontal surface = ');
% RA = Reflector angle with horizontal plane
% following comments for the program used for split ref. case
%SA = 10; %input ('Enter angle made by splitter in parking condition = ');
%SA = Split angle
%no = 2; % input ('Enter the number of split = ')
% no= number of split on side reflectors
disp ('Geo data acquired')
```

```
%% ---- Input 2 ---- Solar input condition
I = 500; % taking solar intensity
% while prog executed for given solar data then input the excel file from
% the Solcast city and state wise solar radiation data.
ang = 40; % taking inclination angle with horizontal surface
disp ('solar radiation data acquired')

%% ---- Calculation 1: Getting top and bottom plane
% calculating inclined and vertical height
% IL = Inclined length, HE = Height, cp_T =core point of top surf.
% cp_V=core point of vertical surf.

% step 1 : function for getting data of IL, HE, cp_T, cp_V
[IL cp_T cp_V]=fun_corepoint(TL,TW,BL,BW,n,RA,HE);
% Function for HE, IL, and other data of the TIS core
disp ('top and bottom surafce corepoint data acquired')

% step 2 : getting coordinate of all reflectors
[s1 s2, r1 r2 r3 r4] = fun_coordinate(4,4);
% Function to calculate coordinate of surfaces
disp ('All Reflector line and plane equation is received')
disp ('saving all geo and solar data for continuing of program....')
save all_geodata.mat;
% saved geometric and surface data for above calculation
% data of input 1, input 2 and calculation 1

%% ---- Calculation 2: Ray Tracing for FPSRS
disp ('Ray tracing is started for FPSRS ')
% function to find out tracing of ray
RTA=fun_tracing(RA);
disp ('Ray tracing for Top surface is done')
% Store points for Top surface
TP = lenght(CP_T); % get the total number of rays when entering from TIS
IM_T = [A B C D E F G H TP]; % Total number of ray for ideal model
B_T = [A B C D E F G H TP]; % Total number of ray for base model
SS_T = [A B C D E F G H TP]; % Total number of ray for SS model
```

```
%% --- Calculation 3 : Ray tracing while splitting the reflector
% Details: Vertical plate includes split reflector hence split of it need
% to do. As well as coordinates of split need to find and imaginary
% surface need to be created.
% step 1: getting coordinates of split reflectors for clockwise direction
% of 0:10:60 degree angle.
% AA=fun_split(no,SA)

%% ---- Calculation 4: Ray Tracing for vertical surface
disp ('Ray tracing is started for vertical surface')
RTA=fun_tracing(RA); % function to find out tracing of ray
disp ('Ray tracing for Verticle surface is done')

% Store points for vertical surface
VP = lenght(CP_V); % total number of rays when entering from top surface
IM_T = [A B C D E F G H VP]; % Total number of ray for ideal model
B_T = [A B C D E F G H VP]; % Total number of ray for base model
SS_T = [A B C D E F G H VP]; % Total number of ray for SS model
toc
disp ('Program is over');
```

D.2 Code to find the core points on the TIS (fun_corepoint.m)

```
% Details of the function
% Name : function [IL cp_T cp_V] =fun_corepoint(TL,TW,BL,BW,n,RA,HE)
% Application: to find out the core points on top surfaces and
% vertical surface
% syntax: [IL cp_T cp_V]= fun_corepoint(TL,TW,BL,BW,n,RA,HE)
% Input arguments: TL = top length, TW = top width, BL =bottom length...
% BW = bottom width, n = number of node...
% RA = Reflector angle with horizontal surfaces, HE = height
% Output arguments: IL = Inclined length, cp_T = core point on top surface
% cp_V =  core points on vertical surfaces
function [IL cp_T cp_V] =fun_corepoint(TL,TW,BL,BW,n,RA,HE)
%% --- Part 1 = Inclined and vertical height measurement.
% Step 1: to calculate inclined length (IL)
x_side= (TL-BL)/2; % remaining side after subtracting bottom surface
```

```
IL=x_side/cosd(RA); % find inclination length of reflector
total_node= ((n+1)^2); % Number of centre core point are achieved

% step 2: to calculate height of reflector
% find height of reflector, between top and bottom refl.
HE = HE;%IL*sind(RA);
% Enter the details for getting core points
area_of_unitcell  = ((TL/(n+1))*(TW/(n+1)));
lenghtper_part = (TL/(n+1));

%% --- Part 2= Calculate Top surface core point
% step 1 : Defining x,y,z points for discretised..
% total 5 number is needed, like 3 node + 2 nodes of start and end nodes.
x = linspace(0,TL,(n+2));
y = HE;  % vertical HE
% total 5 number is needed, like 3 node + 2 nodes of start and end nodes.
z = linspace(0,TL,(n+2));
[X,Y,Z] = meshgrid(x,y,z);  % coordinates of single core point
counter = 0;
% step 2 : Calculating all points on Top surface including boundary points also
for k= 1:length(y)
   for j = 1: length(z)
      for i = 1:length(x)
         counter = counter +1;
         % Represents total number of points on surface
         point3d(counter).surf1 = [X(k,i,j) Y(k,i,j) Z(k,i,j)];
         XL1(counter) = X(k,i,j); % X coordinate
          YL1(counter) = Y(k,i,j); % Y coordinate
         ZL1(counter) = Z(k,i,j); % Z coordinate
         % find the core point of first unit area (in X direction)
         d_len_x = (X(1,2,1)-X(1,1,1))/2;
         % find midpoint between first two points
         core_x(counter) = XL1(counter)+d_len_x; % value of ore point.
         if core_x(counter)>=TL  % logical condition
            core_x(counter) =0;  % initiate variable
         end
```

```
        % find the core point of first unit area (in Z direction)
        d_len_z = (Z(1,1,2)-Z(1,1,1))/2;
        core_z(counter) = ZL1(counter)+d_len_z;
        if core_z(counter)>=TW
          core_z(counter) =0;
        end
      end
   end
end

%% step 3 : Removing boundary points
% remove boundary points and create empty cell
for s=1:1:length(core_x)
   if core_x(s)==0 | core_z(s)==0 ; % To remove zero from core point array
   else
     core_point(s).k=[core_x(s) y core_z(s)];
   end
end
%% remove empty cell and output final core point

for s=1:((n+1)^2) % As number of node is n so centre points is only n^2.
% if s<((n+1)^2)
% if node is more that centre point than its end the program
   if isempty(core_point(s).k)==1;
     core_point(s)=[];  % delete the empty row.
   else
   end
   %    end
end

%% --- Part 3 = Calculate Vertical surface core point
% Hint :
%  Initial point start from Point F to B' and from B' to
%  C' so it can be clockwise if looking plane GFB'C' and point are
%  calculated according to FB'C'G.
%  decrement in y direction is done instead on z direction and z direction
% step 1 : Defining x,y,z ppints for discrtised..
```

```
x = linspace(0,TL,(n+2));
% total 5 number is needed, like 3 node + 2 nodes of start and end nodes.
y = linspace(0,HE,(n+2));
% total 5 number is needed, like 3 node + 2 nodes of start and end nodes.
z = TL;
% total 5 number is needed, like 3 node + 2 nodes of start and end nodes.
[X,Y,Z] = meshgrid(x,y,z);
counter = 0;

%% step 2 : Calculating all points on vertical surface
for k= 1:length(z)
   for j = 1: length(y)
     for i = 1:length(x)
        counter = counter +1;

        point3d(counter).surf1 = [X(j,i,k) Y(j,i,k) Z(j,i,k)];
        % Represents total number of points on surface
        XL1(counter) = X(j,i,k); % X coordinate
        YL1(counter) = Y(j,i,k); % Y coordinate
        ZL1(counter) = Z(j,i,k); % Z coordinate

        % find the core point of first unit area (in X direction)
        d_len_x = (X(1,2,1)-X(1,1,1))/2;
        % find mid point between first two points
        core_x(counter) = XL1(counter)+d_len_x;
        % value of ore point.

        if core_x(counter)>=TL
          core_x(counter) =0;
        end
        % find the core point of first unit area (in Z direction)
        d_len_y = (y(1,2,1)-y(1,1,1))/2;
        core_y(counter) = YL1(counter)+d_len_y;
        if core_y(counter)>=HE % Here hight is considering factor
          core_y(counter) =0;
        end
```

```
      end
   end
end

%% step 3 : Removing boundry points
% remove boundry points and ceate emty cell
for s=1:1:length(core_x)
   if core_x(s)==0 | core_y(s)==0 ;
      % To remove zero from core_point array
   else
      core_point_V(s).k=[core_x(s) core_y(s) z];
   end
end

%% remove empty cell and output final core point
for s=1:((n+1)^2)
   %if s<((n+1)^2)
   % if node is more that centre point than its end the program
   if isempty(core_point_V(s).k)==1;
      core_point_V(s)=[];  % delete the empty row
   else
   end
   %end
end

%% Final output of the function
cp_T=core_point;
cp_V=core_point_V;
save corepoint.mat;
end
```

D.3 Code to find the coordinates of the reflectors (fun_coordinates.m)

```
% Details of the function
% Name: function [s1 s2 r1 r2 r3 r4] = fun_coordinate(top_sur, bottom_sur)
% Application of fun : to find coordinate of all reflectors
% syntax : [s1 s2 r1 r2 r3 r4] = fun_coordinate(top_sur, bottom_sur)
% Input arguments: TIS and BRS
% Output arguments: s1 to r4 , including all information
function [s1 s2 r1 r2 r3 r4] = fun_coordinate(top_sur, bottom_sur)
load ('corepoint.mat');  % results from fun_corepoints
format short;
%% For square top and bottom surfaces
if (top_sur==4&bottom_sur==4);
   % Top surface coordinate, origin is at the bottom of 'E' point
   E = [0 HE 0];
   F = [0 HE TW]; % point on z direction
   G = [TL HE TW];
   H = [TL HE 0]; % point on x direction
   % following are the Top and bottom difference which help in finding
   % bottom surface coordinator.
   DB_L = TL-BL; % Difference of top length
   DB_W = TW-BW; % Difference of bottom width
   % Bottom Sur. coordinate, a point is below E point
   A = [DB_L/2 0 DB_W/2];
   B = [DB_L/2 0 DB_W/2+BW]; % point on z direction
   C = [(DB_L/2+BL) 0 (DB_W/2+BW)];
   D = [(DB_L/2+BL) 0 DB_W/2]; % point on x direction

   %% Coordinate of all surfaces and reflectors
   % In clock wise direction
   s1.co = [E; F; G; H]; % Top Surface = s1
   s2.co = [A; B; C; D]; % Bottom Surface = s2
   r1.co = [F; E; A; B]; % reflecting Surface = r1
   r2.co = [E; H; D; A]; % reflecting Surface = r2
   r3.co = [G; H; D; C]; % reflecting Surface = r3
   r4.co = [F; G; C; B]; % reflecting Surface = r4
```

```
%% Plane equation of Top Surface (s1)
syms x y z;
P = [x,y,z];
s1.nor = cross(H-E,F-E);
s1.planefunction = dot(s1.nor, P-E);
s1.p_eq = (s1.planefunction==0);
% following code is to find the angle with the normal
sv=s1.nor.*s1.nor;
svv=sqrt(sum(sv));
%angle with axes to the normal
s1.dir=[acosd(s1.nor(1)/svv) acosd(s1.nor(2)/svv) acosd(s1.nor(3)/svv)];

%% Plane equation of Bottom Surface (s2)
s2.nor = cross(D-A,B-A);
s2.planefunction = dot(s2.nor, P-A);
s2.p_eq = (s2.planefunction==0);
% following code is to find the angle with the normal
sv=s2.nor.*s2.nor;
svv=sqrt(sum(sv));
%angle with axes to the normal
s2.dir=[acosd(s2.nor(1)/svv) acosd(-s2.nor(2)/svv) acosd(s2.nor(3)/svv)];

%% Plane equation of reflecting Surface (r1)
r1.nor = cross(F-E,A-E);
r1.planefunction = dot(r1.nor, P-E);
r1.p_eq = (r1.planefunction==0);
%  following code is to find the angle with the normal
sv=r1.nor.*r1.nor;
svv=sqrt(sum(sv));
r1.dir=[acosd(r1.nor(1)/svv) acosd(r1.nor(2)/svv) acosd(r1.nor(3)/svv)];
%angle with axes to the normal

%% Plane equation of reflecting Surface (r2)
r2.nor = cross(E-H,D-H);
r2.planefunction = dot(r2.nor, P-H);
r2.p_eq = (r2.planefunction==0);
%  following code is to find the angle with the normal
```

```
    sv=r2.nor.*r2.nor;
    svv=sqrt(sum(sv));
   r2.dir=[acosd(r2.nor(1)/svv) acosd(r2.nor(2)/svv) acosd(r2.nor(3)/svv)];
    %angle with axes to the normal

    %% Plane equation of reflecting Surface (r3)
    r3.nor = cross(G-H,D-H);
    r3.planefunction = dot(r3.nor, P-H);
    r3.p_eq = (r3.planefunction==0);
    %  following code is to find the angle with the normal
    sv=r3.nor.*r3.nor;
    svv=sqrt(sum(sv));
   r3.dir=[acosd(-r3.nor(1)/svv) acosd(-r3.nor(2)/svv) acosd(r3.nor(3)/svv)];
    %angle with axes to the normal

    %% Plane equation of reflecting Surface (r4)
    r4.nor = cross(G-F,B-F);
    r4.planefunction = dot(r4.nor, P-F);
    r4.p_eq = (r4.planefunction==0);
    %  following code is to find the angle with the normal
    sv=r4.nor.*r4.nor;
    svv=sqrt(sum(sv));
    %angle with axes to the normal
   r4.dir=[acosd(r4.nor(1)/svv) acosd(r4.nor(2)/svv) acosd(r4.nor(3)/svv)];
    all_plane_eq = [s1.p_eq; s2.p_eq; r1.p_eq; r2.p_eq; r3.p_eq; r4.p_eq;];
    angle_dir.pk=[s1.dir; s2.dir; r1.dir; r2.dir; r3.dir; r4.dir;];

    %% Hexagonal TIS
elseif (top_sur==4&bottom_sur==6);
    fprintf('Your Top and Bottom selected shape of surface is square and Hexagonal, respectively')
    TL = 4%input('enter value for length of top surface in mtr = ');
    TW = 4%input('enter value for width of top surface in mtr = ');
    BL = 2%input('enter value for a side of Hexagonal BRS in mtr = ');
    BW = BL;
    s1.area_top = TL*TW;
    s2.area_bottom = ((3*sqrt(3))/2)*BL^2;
    disp ('**still coordinate for all 6 surface are still remaining to find out; will update soon')
```

```
    r1.pl_eq = 'still remaining';
    r2.pl_eq = 'still remaining';
    r3.pl_eq = 'still remaining';
    r4.pl_eq = 'still remaining';

    %% Octagonal
elseif (top_sur==4&bottom_sur==8);
    fprintf('Your Top and Bottom selected shape of surface is square and Octagonal, respectively')

    TL = 4%input('enter value for length of top surface in mtr = ');
    TW = 4%input('enter value for width of top surface in mtr = ');
    BL = 2%input('enter value for a side of Octagonal BRS in mtr = ');
    BW = BL;
    s1.area_top = TL*TW;
    s2.area_bottom = 2*(1+sqrt(2))*BL^2;
    r1.pl_eq = 'still remaining';
    r2.pl_eq = 'still remaining';
    r3.pl_eq = 'still remaining';
    r4.pl_eq = 'still remaining';
    disp ('**still coordinate for all 8 surface are still remaining to find out; will update soon')
else
    disp ('you have enter wrong entry; please read instruction careful')
    s1.pl_eq = 'Not assign, read instruction carefully';
    s2.pl_eq = 'Not assign, read instruction carefully';
    r1.pl_eq = 'Not assign, read instruction carefully';
    r2.pl_eq = 'Not assign, read instruction carefully';
    r3.pl_eq = 'Not assign, read instruction carefully';
    r4.pl_eq = 'Not assign, read instruction carefully';
end
save coordinate.mat;
end
```

D.4 Code to trace a single ray (fun_tracing.m)

```
% ray tracing function
% Application: used for first reflection and Find out coordinates and angle with the reflector
```

```
% syntax : RTA=fun_tracing(RA)
% Input arguments: RA =reflector angle
% Output arguments: RTA = ray tracing algorithm; actually .mat file is
% saved for getting more data
function RTA=fun_tracing(RA)
load all_geodata.mat; % getting neccesary input data for tracing algorithm
for nn=1:length(cp_T)
    % Input Data for ray (x,y,z,ang)
    % X Y Z coordinate of input ray
    x=cp_T(nn).k(1);
    y=cp_T(nn).k(2);
    z=cp_T(nn).k(3);

    % Following is the function to find the value of 't' which is used know
    % the inresection point between line and plane
    % in put variable is x,y,z,ang.
    t_final_z = fun_LP_intersect(x,y,z,ang);
    % gives 6 value of t for incoming rays
    load('data_ray_1.mat');
    % getting value of intersection point and angle of that intersection.
    var_inter_a(nn).intersec = inter_final;
    % Intersection points for all reflector
    var_angle_a(nn).angle = ang_final; % Angle with all reflector
    %-----------------------------------------------------------------
    % Create empty points and angle which is used after each intersection
    % Points for 6 surfaces
    ps1(nn).pt=0;   pr1(nn).pt=0;   pr2(nn).pt=0;
    pr3(nn).pt=0;   pr4(nn).pt=0;   ps2(nn).pt=0;
    % Angles for 6 surfaces
    as1(nn).pt=0;   ar1(nn).pt=0;   ar2(nn).pt=0;
    ar3(nn).pt=0;   ar4(nn).pt=0;   as2(nn).pt=0;
    %-----------------------------------------------------------------
    % following is the intensity with respect to surface
    II_s1(nn).pt=I; II_r1(nn).pt=0; II_r2(nn).pt=0;
    II_r3(nn).pt=0; II_r4(nn).pt=0; II_s2(nn).pt=0;
    % ----------------------
    % following is the intensity with respect to Different cases including
```

```
% A,B,C,D
case_A(nn).pt=0; case_B(nn).pt=0; case_C(nn).pt=0;
case_D(nn).pt=0; case_E(nn).pt=0; case_F(nn).pt=0;
%% sub Prog for t_final Value
if t_final_z==5
  t_val_1(nn).all=t_final_z;
else
  t_val_1(nn).all= t_final_z;
end
tt = t_val_1(nn).all;
%% Ray tracing by use of t value
if tt==5  % this will happen when angle is 0 or 180 degree only
  % disp('Ray is parallel to "TOP Horizontal Surf');
  ps1(nn).pt=var_inter_a(nn).intersec (1,:);
  % point on top horizontal plane
  as1(nn).pt=var_angle_a(nn).angle (1);
  % angle making the ray with top plane
else if tt ~=5
      tt_max = min(tt(tt>0 & tt <1.0));
      %----------------------------------------
      % like first intersection is done on the 0 side of the line
      % when "tt_max is empty then tt_max consider as zero"
      % "tt_max come empty when there are 1 and 0 value for it
      % so '0' is considered as final requirement.
      %----------------------------------------
      if isempty(tt_max)==1 % when ray reach to bottom
        ps2(nn).pt=var_inter_a(nn).intersec(6,:);
        as2(nn).pt=var_angle_a(nn).angle(6);
        %ang_a=90-ang; % as incident ray is directly reach to the
        %bottom so no decrement in solar intensity.
        % II_s2(nn).pt=I*cosd(90-ang);
        % Shows final intensity of incident ray
        II_s2(nn).pt=I;
        % considering no reduction in eff. and directly reach to bottom
        case_A(nn).pt=II_s2(nn).pt;
        % following is for checking further interaction of ray
      elseif tt_max~=0
```

```
save data_neigh.mat;
% this file is saved to input data in t value calculation....
ad = min(find(tt==tt_max));
switch ad
  %% reflector r1 ad=2
  case 2
    % Following are the coordinates and angle after first intersection to the reflector
    pr1(nn).pt= var_inter_a(nn).intersec(ad,:); % Point on reflector
    ar1(nn).pt=var_angle_a(nn).angle(ad); % Angle with reflector normal
    % for simplified things
    px=pr1(nn).pt(1); py=pr1(nn).pt(2); pz=pr1(nn).pt(3); % Coordinates
    xang=ar1(nn).pt; % Angle with normal of the reflector to the incoming ray
    II_r1(nn).pt=I*cosd(xang)*0.9124; % incident ray intensity
    % Intial checking for F case with following condition
    angx=(180-H_ang*2)+H_ang;
% this angle for calculate if incident ray angle is more than 90 degree
    if xang<=H_ang | ang>=angx % condition for F case
      % ray is passing from the top ...
      xang=abs(H_ang-xang); % angle with horizontal for further reflection
      t_final_z_out = fun_LP_intersect_outray(px,py,pz,xang);
      if t_final_z_out(1)<=0
% .. final verification ... ray is going outside from TOP surface
        case_F(nn).pt=nn;
      else
        number(nn).pt=nn
      end
    else  % For B,C,D,E cases
      ang_hor=xang-H_ang; % effective angle with horizontal
      ang_hor=180+ang_hor; % this formula is for reflector case 2 only
      %ang_hor is considering angle for next reflection
      % tracing after 1st reflection--
      dd=fun_next_ray(px,py,pz,ang_hor,1);
      px=dd(2); py=dd(3);pz=dd(4); % coordinates for next reflection
      xang=dd(5); % angle for next reflection
      II_r1(nn).pt=II_r1(nn).pt*cosd(xang)*0.9124;
% Shows final intensity of incident ray
      if dd(1)==3 | dd(1)==5; % Case D neighbour reflector
```

```
    if xang>=H_ang % confirmation of D case
      ang_hor=xang-H_ang;
      dd=fun_next_ray(px,py,pz,ang_hor,1);
      xang=dd(5);
      if dd(1)==0
        case_D(nn).pt=II_r1(nn).pt*cosd(xang)*0.9124;
      else
        number(nn).pt=nn
      end
    end
  elseif dd(1)==2 % C or E case
    if xang>=H_ang % confirmation of C case
      ang_hor=xang-H_ang;
      dd=fun_next_ray(px,py,pz,ang_hor,1);
      xang=dd(5);
      if dd(1)==0
        case_C(nn).pt=II_r1(nn).pt*cosd(xang)*0.9124;
      else
        number(nn).pt=nn
      end
    else
      % for case E
      ang_hor=180-(xang+H_ang);
      dd=fun_next_ray(px,py,pz,ang_hor,1);
      xang=dd(5);
      if dd(1)==0
        case_E(nn).pt=nn;
      else
        number(nn).pt=nn
      end
    end
  elseif dd(1)==0 % Case_B single reflection
    case_B(nn).pt=II_r1(nn).pt; % intensity after reaching at base;
  else
    number(nn).pt=nn
  end
end
```

```
            %% reflector r2 when ad=3
          case 3
            % Following are the coordinates and angle after first intersection to the reflector
            pr2(nn).pt= var_inter_a(nn).intersec(ad,:); % Point on reflector
            ar2(nn).pt=var_angle_a(nn).angle(ad); % Angle with reflector normal
            % for simplified things
            px=pr2(nn).pt(1); py=pr2(nn).pt(2); pz=pr2(nn).pt(3); % Coordinates
            xang=ar2(nn).pt; % Angle with normal of the reflector to the incoming ray
            II_r2(nn).pt=I*cosd(xang)*0.9124; % incident ray intensity
            I=II_r2(nn).pt;
            % Now doing for next reflection process
            % by considering plane from which ray is passing
            %% new method to find line and plane intersection
            % first passing plane and then line intersection
            for er=1:5
              if er==5 % terminating statement.. Consider er=5 as F case
                case_F(nn).pt=0; % number of time interaction , terminate problem
                term_point(nn).pt=nn; %this ray is neglected due to more number of reflection
                break ;
              else
                dd=fun_next_ray_n(px,py,pz,xang,I,nn);
% special function for neighbour reflector
                % for intersection..
                % dd output will be;  dd= [ad xf yf zf angf N_Int];
                if dd==6 %  after one reflection reach to bottom case_B
                  case_B(nn).pt=dd(6);
% intensity value store in the 6th number of value of dd array
                elseif dd==1 % ray leave from the top surface
                  case_F(nn).pt=0; % Ray leaving outside
                else
% otherwise considered as neighbour intersection or may be (CASE of C,D,E)
                  % ----> in the following section after
                  % second reflection find out whether ray
                  % is reach to bottom or not.
                   save nx.mat px py pz; % getting the information of point after second reflection
                   dd=fun_next_ray_nn(dd(2),dd(3),dd(4),dd(5),dd(6),nn);
                   % output of dd=[ad xf yf zf angf N_Int];
```

```
                if dd(1)==6
                  case_D(nn).pt=dd(6); % Case d is considered
                break;
                end
              end
            end
          end
          %% following is the alternative methods
          %
          %           syms a b c X Y Z;
          %           P = [a,b,c,X,Y,Z];
          %           eq_ex=vpa((a*(P(3)-p1(1))+b*(P(4)-p1(2))+c*(P(5)-p1(3))==0),3);
          %           eq_a=vpa((a*(p4(1)-p1(1))+b*(p4(2)-p1(2))+c*(p4(3)-p1(3))==0),3);
          %           eq_b=vpa((a*r2.nor(1)+b*r2.nor(2)+c*r2.nor(3)==0),3);
          %           %eq_ans=vpa((solve([eq_a,eq_b],[a,b])),3);
          %           eq_sol=solve([eq_a,eq_b],[a,b]);
          %           % now find c value
          %           val_c=subs(eq_a,[a,b],[eq_sol.a,eq_sol.b])
          %           % now find the a,b,c values .....
          %           val_a=eq_sol.a;
          %           val_b=subs(eq_b,c,val_c)
          %           val_c=solve(val_c);
          %           % now put solution of eq_sol in the eq_ex names
          %           % equation....
% eq_ex=vpa((val_a*(P(3)-p1(1))+val_b*(P(4)-p1(2))+val_c*(P(5)-p1(3))==0),3);
          %           vec_line1=[xx-px yy-py zz-pz];
          %           vec_line2=[p5(1)-px p5(2)-py p5(3)-pz];
          %           re=cross(vec_line1,vec_line2)
          %           % intersection of line and plane
          %           newfunction_s1 = subs(p_eq, P, line_eq);
% intersecting line and plane function
          %           t0_s1 = solve(newfunction_s1);
% Solving line and plane function and find't' equation
          %
          %% Old method to find the plane and line intersection
          angx=(180-H_ang*2)+H_ang;
% this angle for calculate if incident ray angle is more than 90 degree
```

```
            if xang<=H_ang | ang>=angx % condition for F case
              % following code to confirmation of F case when
              % ray is passing from the top ...
              xang=H_ang-xang;
              t_final_z_out = fun_LP_intersect_outray(px,py,pz,xang);
              if t_final_z_out(1)<=0
% .. final verification ... ray is going outside from TOP surface
                case_F(nn).pt=nn;
              end
            else   % for A to E cases...
              ang_hor=abs(H_ang-xang);
              if ang_hor<=H_ang % checking case E condition
                % following code is for confirmation of E
                % case
                t_final_z_out = fun_LP_intersect_outray(px,py,pz,ang_hor);
                if t_final_z_out(1)<=0
% .. final verification ... ray is going outside from TOP surface
                  case_E(nn).pt=nn;
                end
              else % For case D...
                %For Particular D case
                dd=fun_next_ray(px,py,pz,ang_hor,1); % function for next reflection
                px=dd(2); py=dd(3);pz=dd(4); % coordinates after next reflection
                xang=dd(5); % angle after next reflection
                II_r2(nn).pt=II_r2(nn).pt*cosd(xang)*0.9124; % Intensity after next reflection
 %% Find difficult while its get intersect to the ref. 3 and then ref. 4 and no trace that where ray will
go.
                xang=abs(xang-H_ang);
                dd=fun_next_ray(px,py,pz,xang,1);
                px=dd(2); py=dd(3);pz=dd(4); % coordinates for next reflection
                xang=dd(5);
                case_D(nn).pt=II_r2(nn).pt;
              end
            end
            %% reflector r3 ad=4
          case 4
            % Following are the coordinates and angle after first intersection to the reflector
```

```
            pr3(nn).pt= var_inter_a(nn).intersec(ad,:); % Point on reflector
            ar3(nn).pt=var_angle_a(nn).angle(ad); % Angle with reflector normal
            % for simplified things
            px=pr3(nn).pt(1); py=pr3(nn).pt(2); pz=pr3(nn).pt(3); % Coordinates
            xang=ar3(nn).pt; % Angle with normal of the reflector to the incoming ray
            II_r3(nn).pt=I*cosd(xang)*0.9124; % incident ray intensity
            % Intial checking for F case with following condition
            angx=(180-H_ang*2)+H_ang;
% this angle for calculate if incident ray angle is more than 90 degree
            if xang<=H_ang | ang>=angx % condition for F case
              % following code to confirmation of F case when
              % ray is passing from the top ...
              xang=abs(H_ang-xang); % angle with horizontal for further reflection
              t_final_z_out = fun_LP_intersect_outray(px,py,pz,xang);
              if t_final_z_out(1)<=0
% .. final verification ... ray is going out side from TOP surface
                case_F(nn).pt=nn;
              else
                number(nn).pt=nn
              end
            else  % For B,C,D,E cases
              ang_hor=xang-H_ang; % effective angle with horizontal
              ang_hor=360-ang_hor; % Angle for further process when intersect with reflector 4
              % Tracing after 1st reflection--
              dd=fun_next_ray_in(px,py,pz,ang_hor,1);
              px=dd(2); py=dd(3);pz=dd(4); % coordinates for next reflection
              xang=dd(5); % angle for next reflection
              II_r3(nn).pt=II_r3(nn).pt*cosd(xang)*0.9124;
% Shows final intensity of incident ray
              %% now second reflection from here.....
              %--- as here no D case
              if dd(1)==2 % C or E case
%
%                 if xang>=H_ang % confirmation of D case
%                   ang_hor=xang-H_ang;
%                   dd=fun_next_ray(px,py,pz,ang_hor,1);
%                   xang=dd(5);
```

```
%                       if dd(1)==0
%                          case_D(nn).pt=II_r3(nn).pt*cosd(xang)*0.9124;
%                       else
%                          number(nn).pt=nn
%                       end
%                     end
                    if xang>=H_ang % confirmation of C case
                      ang_hor=xang-H_ang;
                      dd=fun_next_ray(px,py,pz,ang_hor,1);
                      xang=dd(5);
                      if dd(1)==0
                        case_C(nn).pt=II_r3(nn).pt*cosd(xang)*0.9124;
                      else
                        number(nn).pt=nn
                      end
                    else
                      % for case E
                      ang_hor=180-(xang+H_ang); %Effective angle from horizontal surface
                      t_final_z_out = fun_LP_intersect_outray(px,py,pz,ang_hor);
                      if t_final_z_out(1)<=0
% .. final verification ... ray is going out side from TOP surface
                        case_E(nn).pt=nn;
                      else
                        number(nn).pt=nn
                      end
                    end
                  elseif dd(1)==0 % Case_B single reflecion
                    case_B(nn).pt=II_r3(nn).pt; % intensity after reaching at base;
                  else
                    number(nn).pt=nn
                  end
                end


                %% reflector r4 ad=5
              case  5
                % Following are the coordinates and angle after first intersection to the reflector
```

```
                pr4(nn).pt= var_inter_a(nn).intersec(ad,:); % Point on reflector
                ar4(nn).pt=var_angle_a(nn).angle(ad); % Angle with reflector normal
                % for simplified things
                px=pr4(nn).pt(1); py=pr4(nn).pt(2); pz=pr4(nn).pt(3); % Coordinates
                xang=ar4(nn).pt; % Angle with normal of the reflector to the incoming ray
                II_r4(nn).pt=I*cosd(xang)*0.9124; % incident ray intensity
                I=II_r4(nn).pt;
                % Now doing for next reflection process
                % Now doing for next reflection process
                % by considering plane from which ray is passing
                %% new method to find line and plane intersection
                % first passing plane and then line intersection
                for er=1:5
                  if er==5 % terminating statement.. Consider er=5 as F case
                    case_F(nn).pt=0; % number of time interaction , terminate problem
                    term_point(nn).pt=nn;
                    break ;
                  else
                    dd=fun_next_ray_n(px,py,pz,xang,I,nn);
% special function for neighbour reflector
                    % for intersection..
                    % dd output will be;  dd= [ad xf yf zf angf N_Int];
                    if dd==6 %  after one reflection reach to bottom case_B
                      case_B(nn).pt=dd(6);
% intensity value store in the 6th number of value of dd array
                    elseif dd==1 % ray leave from the top surface
                      case_F(nn).pt=0; % Ray leaving outside
                    else % otherwise considered as neighbour intersection
                      dd=fun_next_ray_n(dd(2),dd(3),dd(4),dd(5),dd(6),nn);
                      % output of dd=[ad xf yf zf angf N_Int];
                      if dd(1)==6
                        case_D(nn).pt=dd(6); % Case d is considered
                      break;
                      end
                    end
                  end
                end
```

```
            % First check for F case with following condition
            angx=(180-H_ang*2)+H_ang;
% this angle for calculate if incident ray angle is more than 90 degree
            if xang<=H_ang | ang>=angx % condition for F case
              % following code to confirmation of F case when
              % ray is passing from the top ...
              xang=H_ang-xang;
              t_final_z_out = fun_LP_intersect_outray(px,py,pz,xang);
              if t_final_z_out(1)<=0
% .. final verification ... ray is going out side from TOP surface
                case_F(nn).pt=nn;
              end
            else   % for A to E cases...
              ang_hor=abs(H_ang-xang);
              if ang_hor<=H_ang % cheaking case E condition
                % following code is for confirmaton of E
                % case
                t_final_z_out = fun_LP_intersect_outray(px,py,pz,ang_hor);
                if t_final_z_out(1)<=0
% .. final verification ... ray is going out side from TOP surface
                  case_E(nn).pt=nn;
                end
              else % For case B,C,D...
                %For Particular D case
                xang=xang-H_ang;
% as system taking horizontal surface as ref. thus this line give the angle from the hor. surface
                dd=fun_next_ray(px,py,pz,xang,1); % function for next reflection
                px=dd(2); py=dd(3);pz=dd(4); % coordinates after next reflection
                xang=dd(5); % angle after next reflection
                II_r4(nn).pt=II_r4(nn).pt*cosd(xang)*0.9124; % Intensity after next reflection
                case_D(nn).pt=II_r4(nn).pt;
%               px=dd(2); py=dd(3);pz=dd(4); % coordinates for next reflection
%               xang=dd(5);
%               xang=abs(xang-H_ang);
%             dd=fun_next_ray(px,py,pz,xang,1);
%
              end
```

```
                end
            end
          end
        end
    end
    nn
end
RTA=1;
save tracing.mat;
end
```

D.5 Code to find the intersection points of ray and reflector (fun_LP_intersect.m)

```
%Function for finding t value when ray comes from top to bottom
function t_final = fun_LP_intersect(x_L,y_L,z_L,angg)
load ('coordinate.mat'); % Result of fun_coordinate();
%% to fins Vector Equation of line
% line equation ((x-x1)/(p2.x2-x1)=(y-y1)/(p2.y2-y1)=(z-z1)/(p2.z2-z1)=t)
sintheta_degree = sind(angg);
costheta_degree = cosd(angg);
ang = @(u,v) acosd(dot(u,v)/(norm(u)*norm(v)));
% syntax of the function to find angle between line and plane
p4 = [x_L y_L z_L]; % Incident point on top surface
slnatside_L1 = y_L/sintheta_degree; % slant side of line means incident radiation
% Condition of slant side (slantside_l1=Inf at 0 and 180 degree of angle)
if slnatside_L1==Inf % when angle is zero than slant side is zero because sin 0=0
    t_final = [5 5 5 5 5 5];
% t value give information about intersection point. if t is higher means its first intersect at that point.
    inter_final = [6 6 6; 6 6 6; 6 6 6; 6 6 6; 6 6 6; 6 6 6]; % Intersection point
    ang_final = [ 7 7 7 7 7 7]; % angle at intersection
else
    % Coordinate on horizontal plane at y = 0;
    x2_L1 = ((costheta_degree*slnatside_L1)+x_L);
    y2_L1 = 0; % at bottom plate its taken by default as zero
    z2_L1 = z_L ; % considering z coordinate not changed
    p5 = [x2_L1 y2_L1 z2_L1]; % point on horizontal plane
    syms t; % symbolic t for getting line and its intersection equation
```

```
% Calculating line equation
line_eq = vpa(p4 + t*(p5-p4),3);
%% Intersection of line with top surface (plane_1)
newfunction_s1 = vpa((subs(s1.p_eq, P, line_eq)),3); % intersecting line and plane function
t0_s1 = solve(newfunction_s1); % Solving line and plane function and find 't' equation
t0_s1=round((double(t0_s1)),3);
point_s1 = double(subs(line_eq, t, t0_s1)); % finding point on plane when line is intersect with it
u_s1 = s1.nor; % Coordinate of normal vector to plane
ang_s1=ang(u_s1,(p5-p4)); % this an angle make with the normal of plane
if ang_s1>=90;  ang_s1=180-ang_s1;  end;
%% Intersection of line with reflector 1 (plane_2))
newfunction_r1 = subs(r1.p_eq, P, line_eq);
t0_r1 = solve(newfunction_r1);
t0_r1=round((double(t0_r1)),3);
point_r1 = double(subs(line_eq, t, t0_r1));
u_r1 = r1.nor;
ang_r1=ang(u_r1,(p5-p4));
if ang_r1>=90;  ang_r1=180-ang_r1;  end;
%% Intersection of line with reflector 2 (plane_3)
newfunction_r2 = subs(r2.p_eq, P, line_eq);
t0_r2 = solve(newfunction_r2);
t0_r2=round((double(t0_r2)),3);
point_r2 = double(subs(line_eq, t, t0_r2));
u_r2 = r2.nor;
ang_r2=ang(u_r2,(p5-p4));
if ang_r2>=90;  ang_r2=180-ang_r2;  end;
%% Intersection of line with reflector 3 (plane_4)
newfunction_r3 = subs(r3.p_eq, P, line_eq);
t0_r3 = solve(newfunction_r3);
t0_r3=round((double(t0_r3)),3);
point_r3 = double(subs(line_eq, t, t0_r3));
u_r3 = r3.nor;
ang_r3=ang(u_r3,(p5-p4));
if ang_r3>=90;  ang_r3=180-ang_r3;  end;
%% Intersection of line with reflector 4 (plane_5)
newfunction_r4 = subs(r4.p_eq, P, line_eq);
t0_r4 = solve(newfunction_r4);
```

```
    t0_r4=round((double(t0_r4)),3);
    point_r4 = double(subs(line_eq, t, t0_r4));
    u_r4 = r4.nor;
    ang_r4=ang(u_r4,(p5-p4));
    if ang_r4>=90;  ang_r4=180-ang_r4;  end;
    %% Intersection of line with bottom surface (plane_6)
    newfunction_s2 = subs(s2.p_eq, P, line_eq);
    t0_s2 = solve(newfunction_s2);
    t0_s2=round((double(t0_s2)),3);
    point_s2 = double(subs(line_eq, t, t0_s2));
    u_s2 = s2.nor;
    ang_s2=ang(u_s2,(p5-p4));
    if ang_s2>=90;  ang_s2=180-ang_s2;  end;
    %%
    % with for loop and with use of array we can do that , break
    t_final =[t0_s1 t0_r1 t0_r2 t0_r3 t0_r4 t0_s2];
    inter_final =[ point_s1; point_r1; point_r2; point_r3; point_r4; point_s2];
    ang_final = [ang_s1 ang_r1 ang_r2 ang_r3 ang_r4 ang_s2];
end
save data_ray_1.mat inter_final ang_final;
end
```