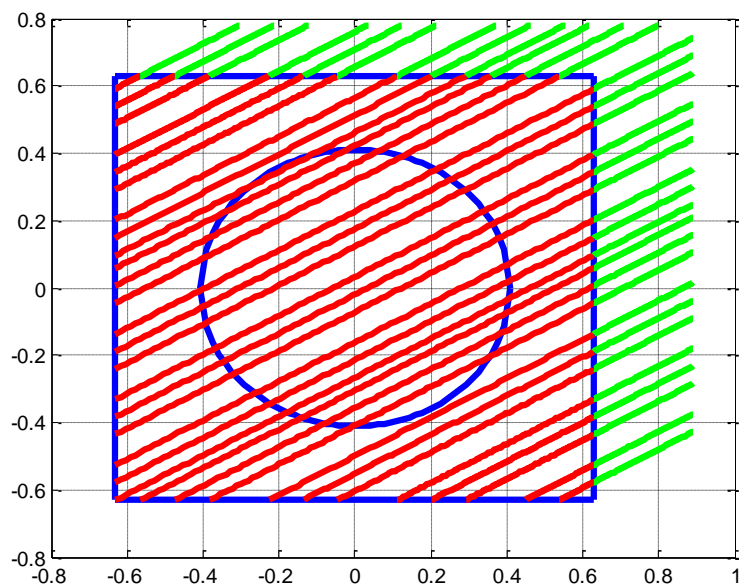
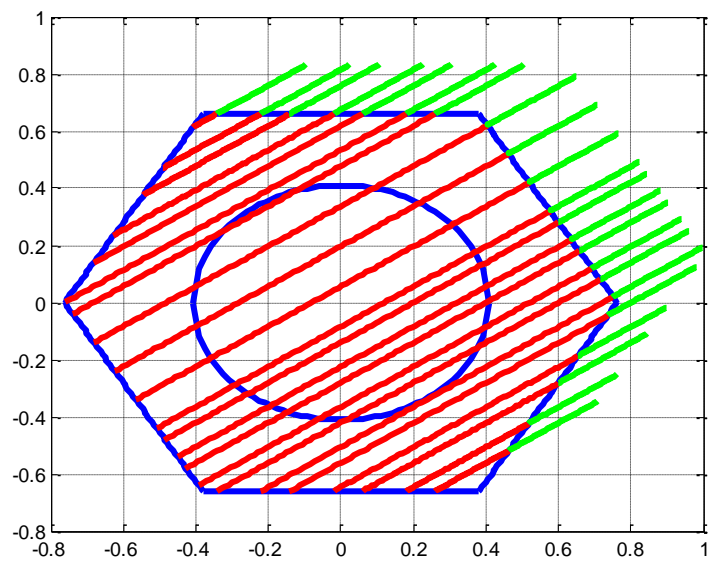


Examples of visual tracing with the program.

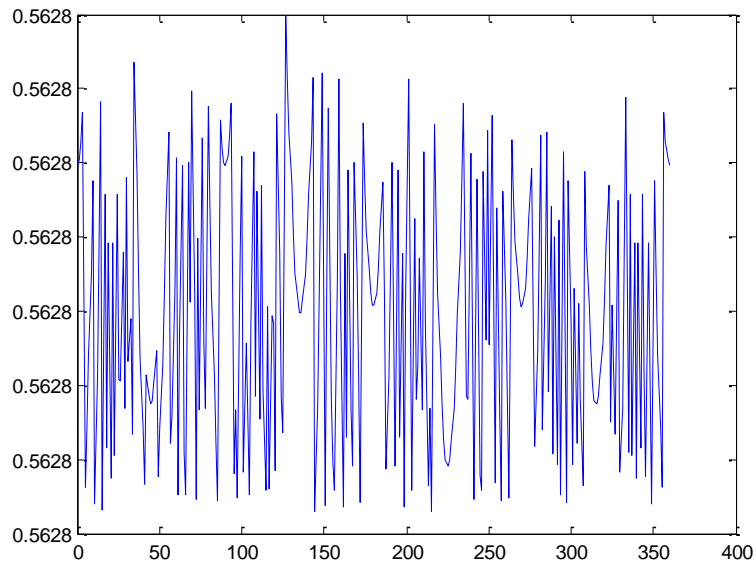


This Figure shows the square lattice with path tracing. The green shows direction of travel.

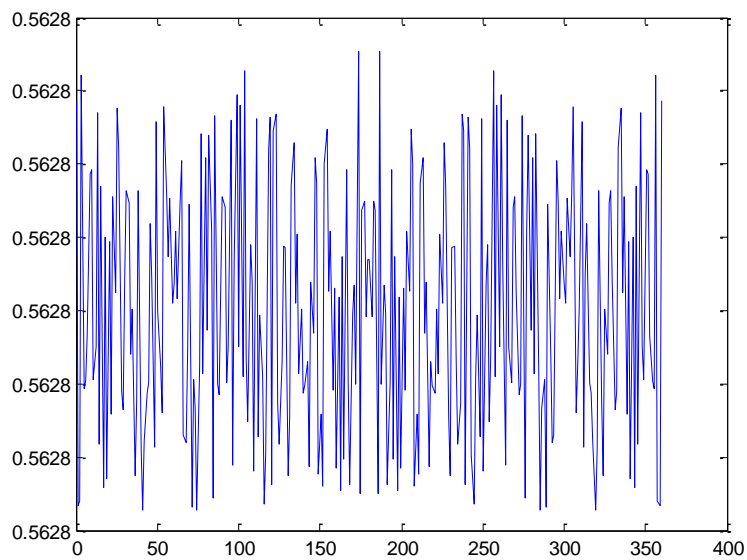


This figure shows the hex lattice.

2. The code was tested by assuming that the source and cross section values were constant throughout both pins. The plots for these cases are shown below.



Square Lattice

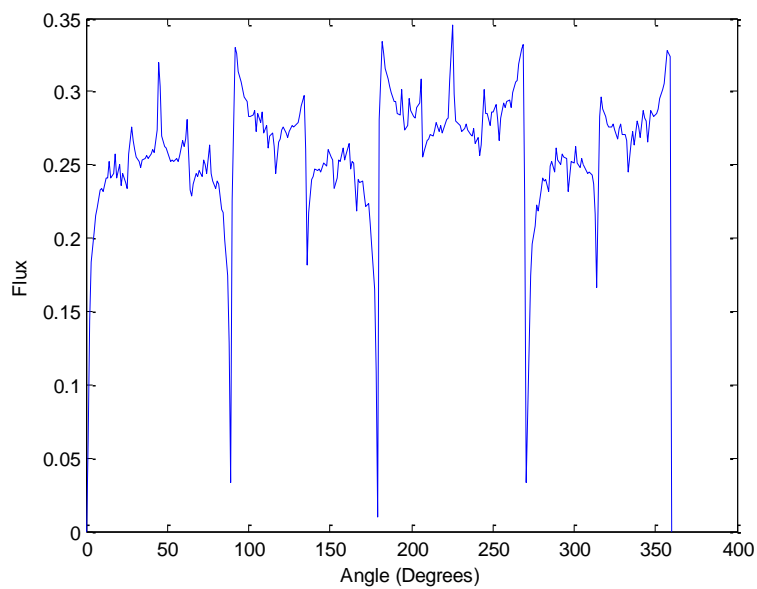


Hexagonal Lattice

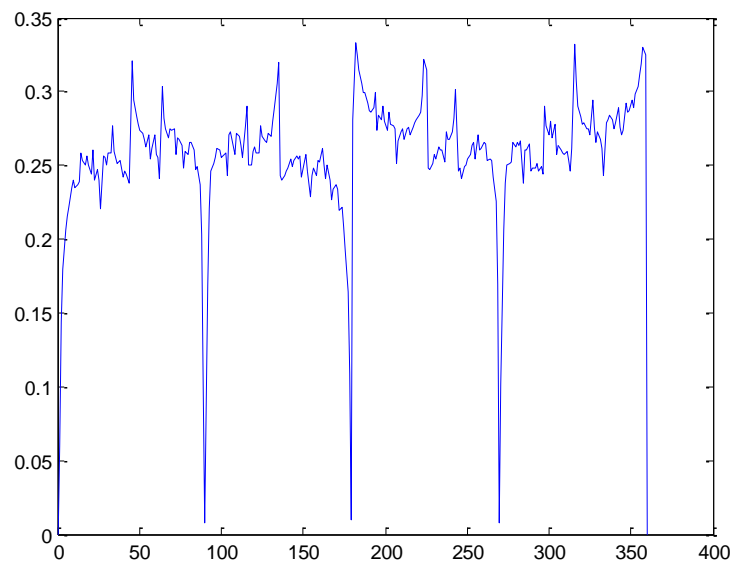
These plots show a flat distribution.

### 3. Plots at different points:

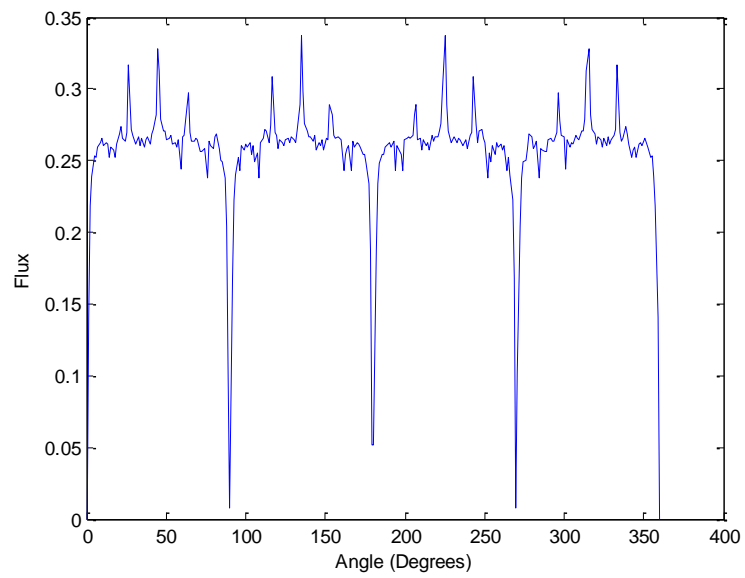
Square Lattice [0.41,0.41]



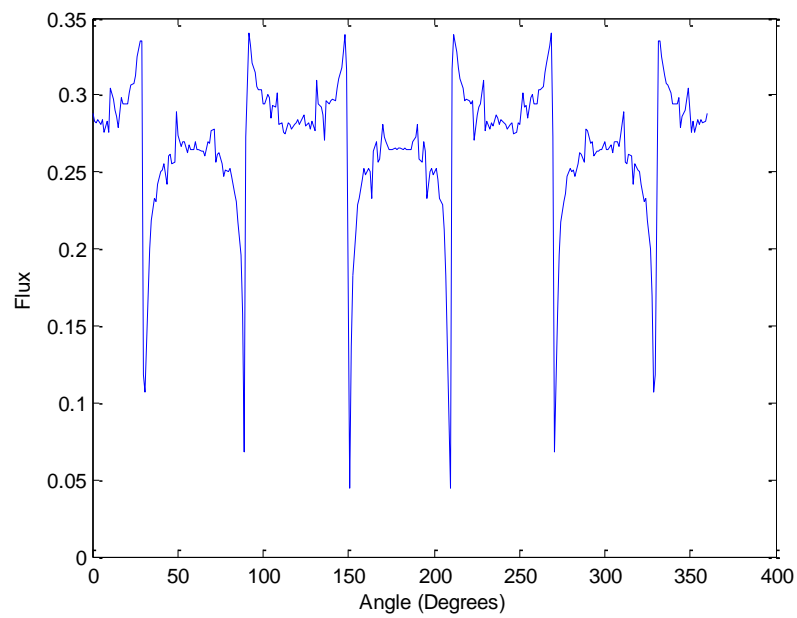
Square Lattice [0.63,0.41]



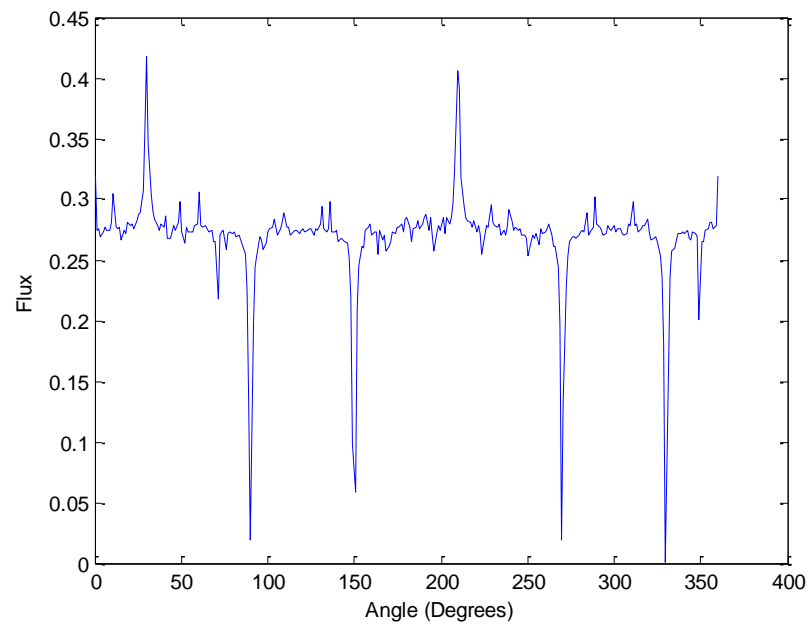
Square Lattice [0.63,0.63]



Hexagonal Lattice [0.381051,0.66]:



Hexagonal Lattice [0.572,0.33]:



These plots depict peaks and valleys, accurately representing my emotions in the completion of this homework.

Programs:

## Loop\_Check.m

```
function [Flux]=Loop_Check(n,mfp,r,CHOICE)
theta=linspace(0,360,n);
Flux=zeros(1,n);

%You can pick which problem to work on
for i=1:n
    %Flux(1,i)=Problem_3(theta(i),[0.63,0.63],1,mfp);
    %Flux(1,i)=Problem_3(theta(i),[0.381051,0.66],2,mfp);
    %Flux(1,i)=Problem_3(theta(i),[0.41,0.41],1,mfp);
    %Flux(1,i)=Problem_3(theta(i),[0.63,0.41],1,mfp);
    %Flux(1,i)=Problem_3(theta(i),[0.57157676649,0.33],2,mfp);
    Flux(1,i)=Problem_3(theta(i),r,CHOICE,mfp);
end

hold off
plot(theta,Flux);
xlabel 'Angle (Degrees)'
ylabel 'Flux'
```

## Problem\_3.m

```
function [Flux]=Problem_3(theta,r,CHOICE,mfp)
%This program attempts to solve problem 3 of homework 1
%of NUEN 629 Fall 2015, for details refer to the problem statement
format short
format compact
%Units cm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Input %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Lattice (Note: Hexigons should have flat faces with constant y values)
%Note Make sure x1 and y1 values are centered
%Form of input: [y1,x1,m,sin(b),cos(b)]

%Central Radius
R=0.41;
%Hexagon 1/2 pitch
RH=0.66;

%CHOICE=2;
if CHOICE==1
    %Square Lattice:
    Lines(1,1:5)=[0,0.63,nan,1,0]; %Positive X Line
    Lines(2,1:5)=[0,-0.63,nan,1,0]; %Negative X line
    Lines(3,1:5)=[0.63,0,0,0,1]; %Positive Y line
    Lines(4,1:5)=[-0.63,0,0,0,1]; %Negative Y line
else %Hex Lattice:
    Lines(1,1:5)=[0.66,0,0,0,1]; %Positive Y line
    Lines(2,1:5)=[-0.66,0,0,0,1]; %Negative Y line
```

```

Lines(3,1:5)=[sind(30)*RH,cosd(30)*RH,tand(120),sind(120),cosd(120)]; %Right Side Negative
Slope
Lines(4,1:5)=[-sind(30)*RH,cosd(30)*RH,tand(60),sind(60),cosd(60)]; %Right Side Positive
Slope
Lines(5,1:5)=[-sind(30)*RH,-cosd(30)*RH,tand(120),sind(120),cosd(120)]; %Left Side Negative
Slope
Lines(6,1:5)=[sind(30)*RH,-cosd(30)*RH,tand(60),sind(60),cosd(60)]; %Left Side Positive
Slope
end

Em=0.08;Ef=0.1414;I=1/(4*pi*Ef); %Values for Flux Determination
Q_CHECK=0;
%In Order to Check if my solution is working, need a flat profile
%Q_CHECK=1;Em=0.1414;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Find Length of Edges
if (any(Lines(:,3)))
    Lengths=(RH*2)/(3^0.5);
    Lengthx=Lengths*cosd(60);
    Pitch=2*RH;
    Min_dis=(Pitch/2)/sind(60)-R;
else
    Lengths=max(Lines(:,2))-min(Lines(:,2));
    Pitch=Lengths;
    Lengthx=1;
    Min_dis=(Pitch/(2^0.5))-R;
end

Rows=size(Lines,1); %Used for Looping
limits=zeros(Rows,4); %Limits for lines [ymin,ymax,xmin,xmax]

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Plot Boundaries to visually observe %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xplot=zeros(1,100);
yplot=xplot;

for j=1:Rows %Loop over all Boundary Lines
    slope=1;
    if(Lines(j,3)<0) %Check for negative Slopes
        slope=-1;
    end
    if(Lines(j,3)==0) %m=0, sin(b)=0, cos(b)=1
        yplot(1,:)=Lines(j,1); %y is constant
        xplot(1,:)=linspace(Lines(j,2)-Lengths/2,Lines(j,2)+Lengths/2);
        limits(j,1)=Lines(j,1);
        limits(j,2)=Lines(j,1);
        limits(j,3)=Lines(j,2)-Lengths/2;
        limits(j,4)=Lines(j,2)+Lengths/2;
    elseif(isnan(Lines(j,3))) %m=INF, sin(b)=1, cos(b)=0
        xplot(1,:)=Lines(j,2); %x is constant
        yplot(1,:)=linspace(Lines(j,1)-Lengths/2,Lines(j,1)+Lengths/2);
        limits(j,1)=Lines(j,1)-Lengths/2;
        limits(j,2)=Lines(j,1)+Lengths/2;
        limits(j,3)=Lines(j,2);
        limits(j,4)=Lines(j,2);
    else %m is nonzero and non inf
        xplot(1,:)=linspace(Lines(j,2)-Lengthx/2,Lines(j,2)+Lengthx/2);
        yplot(1,:)=Lines(j,3).*(xplot-Lines(j,2))+Lines(j,1);
        limits(j,1)=slope.*Lines(j,3).*(xplot(1,1)-Lines(j,2))+Lines(j,1);
        limits(j,2)=slope.*Lines(j,3).*(xplot(end)-Lines(j,2))+Lines(j,1);
        limits(j,3)=Lines(j,2)-Lengthx/2;
        limits(j,4)=Lines(j,2)+Lengthx/2;
    end
    plot(xplot,yplot,'b','LineWidth',3);
    hold on
end

```

```

%Plot The Circle
xplot=linspace(-R,R,50);
yplot=(R.^2-xplot.^2).^0.5;
plot(xplot,-yplot,'b','LineWidth',3);
hold on
plot(xplot,yplot,'b','LineWidth',3);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Calculations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Direction and Magnitude This variable holds our location and direction
Vector(1,1:5)=[r(2),r(1),tand(theta),sind(theta),cosd(theta)];
Vector(Vector==inf)=nan;
Vector(Vector==-inf)=nan;

P=0;P_Store=zeros(1,2000);distance=0;nn=2;
while(P<mfp) %Loop until we have answers...
%For debugging purposes
%for kk=1:100

%If heading outside the system Turn Around: Do twice, just incase you need
%two translations
Vector=Reflection(Rows,Vector,Lines,Pitch,1);
Vector=Reflection(Rows,Vector,Lines,Pitch,1);

%dd will be variable for length till change medium

%Translation Matrix (I am sure there is a better way to do this)
XYT=zeros(Rows+1,3); %Values saved [Ytranslated,Xtranslated,Distance]
%The Plus 1 is for the circle consideration

%Lets go for one interaction to the next Boundary Line:
for j=1:Rows %Loop over all Boundary Lines
    %If we have a sloped line (This should include horizontal lines)
    if(Lines(j,5)~=0 && Vector(1,5)~=0 && Lines(j,3)~=Vector(1,3))
        %There is an intersection
        x2=(Lines(j,3)*Lines(j,2)-Vector(1,3)*Vector(1,2)+Vector(1,1)-
Lines(j,1))/(Lines(j,3)-Vector(1,3));
        y2=Vector(1,3)*(x2-Vector(1,2))+Vector(1,1);
        %Check if intersection is in the right direction
        XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
    end
    %If my vector is traveling up and down
    if(Lines(j,5)~=0 && Vector(1,5)==0)
        x2=Vector(1,2);
        y2=Lines(j,3)*(x2-Lines(j,2))+Lines(j,1);
        XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
    end
    %If my intersecting line is up and down
    if(Lines(j,5)==0 && Vector(1,5)~=0)
        x2=Lines(j,2);
        y2=Vector(1,3)*(x2-Vector(1,2))+Vector(1,1);
        XYT=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines);
    end
end

if(any(XYT)) %If any elements of XYT are non zero
    A=XYT(:,3);
    n=find(A==min(A(A>0)));
    xt=XYT(n(1),2);
    yt=XYT(n(1),1);
    Length=XYT(n(1),3); %this is the track length
end

%%Checking If My tracing is done correctly
if(isnan(Vector(1,3))) %If we have an infinite slope
    xplot=ones(1,100).*Vector(1,2);
    yplot=linspace(Vector(1,1),Vector(1,1)+Vector(1,4)*Length,100);
else

```



```

        xplot=linspace(Vector(1,2),Vector(1,2)+Length*Vector(1,5),100);
        yplot=Vector(1,3).*(xplot-Vector(1,2))+Vector(1,1);
    end
    plot(xplot,yplot,'r','LineWidth',3);
    hold on;

    %Calculations for distances
    [distance,P,P_Store,nn]=Distances(Vector,R,xt,yt,Em,Ef,distance,P,P_Store,nn,Length,mfp,Min_dis,Q
_CHECK);

    Vector(1,1)=yt; %New Location for Vector
    Vector(1,2)=xt;
    %Plot Our Vector for its path
    Length=0.3;
    if(isnan(Vector(1,3))) %If we have an infinite slope
        xplot=ones(1,100)*Vector(1,2);
        yplot=linspace(Vector(1,1),Vector(1,1)+Vector(1,4)*Length);
    else
        xplot=linspace(Vector(1,2),Vector(1,2)+Length*Vector(1,5));
        yplot=Vector(1,3).*(xplot-Vector(1,2))+Vector(1,1);
    end
    plot(xplot,yplot,'g','LineWidth',3);
    hold on
    grid on

end

sum=0;

for L=2:nn-1
    sum=sum+((-1)^L)*exp(-P_Store(L));
end
Flux=I*sum;

```

## Reflection.m

```

function [Vector]=Reflection(Rows,Vector,Lines,Pitch,LT)
%This function will reflect across a cell given input values

%Translation Matrix (I am sure there is a better way to do this)
XYT=zeros(Rows,3); %Values saved [Ytranslated,Xtranslated,Angle]

tol=0.0001;
%Looping over all our lines
for j=1:Rows
    %We need an inversion in logic: One for reflection, another for
    %Right_Direction
    if(LT==1)
        TF=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)))<tol;
    elseif(LT==0)
        TF=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)))>tol;
    end

    %For Debugging Purposes:
    %Value=abs(Lines(j,5)*(Vector(1,1)-Lines(j,1))-Lines(j,4)*(Vector(1,2)-Lines(j,2)));
    %disp('      Cos_L(x) Sin_L(y)  Lx      Ly      Vx      Vy      Val');
    %disp([Lines(j,5),Lines(j,4),Lines(j,2),Lines(j,1),Vector(1,2),Vector(1,1),Value]);

    %Check if on line (or check if not on line in the case of a
    %Right_Direction
    if(TF)
        %Find Normal to the line:
        XN=-1*Lines(j,4);
        YN=Lines(j,5);
        %Find the inward normal
        d1=((Lines(j,2)+XN)^2+(Lines(j,1)+YN)^2)^0.5;
        d2=((Lines(j,2)-XN)^2+(Lines(j,1)-YN)^2)^0.5;
    end
end

```

```

%For Debugging Purposes
%disp('      j      LXN      VNX      d1      d2      LX      LY');
%disp([j,XN,YN,d1,d2,Lines(j,2),Lines(j,1)]);

if(d2<d1)
    XN=-XN;
    YN=-YN;
end
%Check if we are going the right way (dot product should be less
%than 90. If it is greater, then we store a matrix which holds
%reflection coordinates. The reason we have multiple reflection
%coordinates is because we might be on top of two lines

dot=acosd(XN*Vector(1,5)+YN*Vector(1,4));

%For Debugging Purposes
%disp('      j      LXN      VNX      LYN      VYN      dot');
%disp([j,XN,Vector(1,5),YN,Vector(1,4),dot]);

if(LT==1)
    TF2=dot>90;
elseif(LT==0)
    TF2=dot>90;
end

if(TF2)
    XYT(j,2)=Vector(1,2)+XN*Pitch;
    XYT(j,1)=Vector(1,1)+YN*Pitch;
    XYT(j,3)=acosd(XN*Vector(1,5)+YN*Vector(1,4));
end

end

end

if(any(XYT)) %If any elements of XYT are non zero
    n=find(XYT(:,3)==max(XYT(:,3))); %Find the line of the max angle
    Vector(1,2)=XYT(n(1),2);
    Vector(1,1)=XYT(n(1),1);
end

%For Debugging purposes
%disp('XYT...Did we find anything?');
%disp([XYT]);

```

## Right\_Direction.m

```

function [XYT]=Right_Direction(XYT,Vector,x2,y2,j,limits,Lines)

tol=0.000000001;CHECK=1;

%Find the distance you need to go
distance=((Vector(1,2)-x2)^2+(Vector(1,1)-y2)^2)^0.5;

if (distance>tol)
    %If you are within the bounds of habitation, This is all you need
    %after the first step, but if you do not start on the edge, you
    %need to check normals to make sure you are going in the right
    %direction.
    if(y2+tol<limits(j,1)||y2>limits(j,2)+tol||x2+tol<limits(j,3)||x2>limits(j,4)+tol)
        CHECK=0;
    end
    if(CHECK==1)%Check Normals and directions
        CHECK2(1:2)=Vector(1,1:2);
        DOS=Reflection(1,Vector,Lines(j,:),90,0); %will update
    else
        DOS=1;CHECK2=1;
    end
end

```

```

        if(CHECK==1 && (CHECK2(1)~=DOS(1,1) || CHECK2(2)~=DOS(1,2))) %Update worthy Values
            XYT(j,2)=x2;
            XYT(j,1)=y2;
            XYT(j,3)=distance;
        end
    end

%For Debugging purposes
%disp('      x2      x1      xmin      xmax      y2      y1      ymin      ymax')
%disp([x2,Vector(1,2),limits(j,3),limits(j,4),y2,Vector(1,1),limits(j,1),limits(j,2)]);
%disp('      Line      Slope      Dist_cal CHECK(1=keep) CHECK2/=DOS(1,1)');
%disp([j,Vector(1,3),distance,CHECK,CHECK2,DOS(1,1)]);

```

## Distances.m

```

function
[distance,P,P_Store,nn]=Distances(Vector,R,xt,yt,Em,Ef,distance,P,P_Store,nn,Length,mfp,Min_dis,Q
_CHECK)
passed=0;

%Please note...this code will fail if you start in the fuel. Min_dis will
%need to be modified.

%Determine Distance/FLUX Stuff
a=1+Vector(1,3)^2;
b=2*(Vector(1,3)*Vector(1,1)-Vector(1,2)*(Vector(1,3)^2));
c=(Vector(1,3)^2)*(Vector(1,2)^2)+Vector(1,1)^2-2*Vector(1,3)*Vector(1,2)*Vector(1,1)-R^2;
%Determine if we passed through fuel

debug=0;
%For Debugging purposes
%disp('      mv      x1      y1      R      a      b      c');
%disp([Vector(1,3),Vector(1,2),Vector(1,1),R,a,b,c]);
%debug=1;
%disp(' V_sin(x4)      V_cos(y5)');
%disp([Vector(1,4),Vector(1,5)]);

if (abs(Vector(1,5))<0.000001 && isreal((R^2-xt^2)^0.5)) %If cos(b)=0 %Vertical
    xcmin=xt;xcmax=xt;
    ycmin=-1*(R^2-xt^2)^0.5;
    ycmax=(R^2-xt^2)^0.5;
    passed=1;
elseif (abs(Vector(1,4))<0.000001 && isreal((R^2-yt^2)^0.5)) %If sin(b)=0 %Horizontal
    ycmin=yt;ycmax=yt;
    xcmin=-1*(R^2-yt^2)^0.5;
    xcmax=(R^2-yt^2)^0.5;
    passed=1;
elseif (isreal((b^2-4*a*c)^0.5) && abs(Vector(1,5))>0.000001 && abs(Vector(1,4))>0.000001)
    xcmin=(-b-(b^2-4*a*c)^0.5)/(2*a);
    xcmax=(-b+(b^2-4*a*c)^0.5)/(2*a);
    ycmin=Vector(1,3)*(xcmin-Vector(1,2))+Vector(1,1);
    ycmax=Vector(1,3)*(xcmax-Vector(1,2))+Vector(1,1);
    passed=1;
end

%For Debugging purposes
%if(passed==1)
%disp('      x2      y2      xcmax      ycmax      xcmin      ycmin');
%disp([xt,yt,xcmax,ycmax,xcmin,ycmin]);
%end

if(passed==1)%If we passed through fuel
    %If the vector is left traveling the dot with -x should be less than 90
    dot=acosd(-1*Vector(1,5));
    distance_fuel=((xcmax-xcmin)^2+(ycmax-ycmin)^2)^0.5;
    if(distance_fuel<0.00001||Length<Min_dis) %Either on edge of circle

```

```

        passed=0; %Or starting in the moderator somewhere
        %Below is redundant, and sometimes causes errors, but I want to keep it
        %to remind myself about where I came from.
        %elseif(Vector(1,4)==0)%traveling at 180 or 0 degrees
        %    disp('I was here')
        %    distance_before=((xcmax-Vector(1,2))^2+(ycmax-Vector(1,1))^2)^0.5;
        %    distance_after=((xt-xcmin)^2+(yt-ycmin)^2)^0.5;
        elseif(dot<90) %Traveling leftward
            distance_before=((xcmax-Vector(1,2))^2+(ycmax-Vector(1,1))^2)^0.5; %distance traveled
before the fuel
            distance_after=((xt-xcmin)^2+(yt-ycmin)^2)^0.5;
        elseif(dot>90) %Traveling rightward
            distance_before=((xcmin-Vector(1,2))^2+(ycmin-Vector(1,1))^2)^0.5;
            distance_after=((xt-xcmax)^2+(yt-ycmax)^2)^0.5;
        end

        if(passed==1) %Some redundancy is good.
            if(abs(Length-distance_fuel-distance_before-distance_after)>0.000001)
                disp('Your lengths are not adding up...do not worry about it');
                disp('    Fuel    D    Before    After    Length');
                disp([distance_fuel,distance_before,distance_after,Length]);
            end
            distance=distance+distance_before;
            if(Q_CHECK==0)
                P_Store(nn)=distance*Em+P_Store(nn-1); %part for moderator
                nn=nn+1; %for the fuel
                P_Store(nn)=distance_fuel*Ef+P_Store(nn-1);
                P=P_Store(nn);
                nn=nn+1; %new era of moderator
                distance=distance_after; %Resetting distance
            else
                P_Store(nn)=distance*Em*0+P_Store(nn-1); %part for moderator
                nn=nn+1; %for the fuel
                P_Store(nn)=distance_fuel*Ef+P_Store(nn-1)+distance*Ef;
                P=P_Store(nn);
                nn=nn+1; %new era of moderator
                distance=distance_after; %Resetting distance
            end
        else %Some Redundancy is bad
            distance=distance+Length;
        end

        else %If we did not pass through fuel
            distance=distance+Length;
        end

        if(Q_CHECK==0) %Q check is to check if code is working
            if(P_Store(nn-1)+distance*Em>mfp) %Make sure we don't go up to our mfps
                P_Store(nn)=distance*Em+P_Store(nn-1); %part for moderator
                nn=nn+1; %for the fuel
                P_Store(nn)=P_Store(nn-1)+0*Ef;
                P=P_Store(nn);
                nn=nn+1; %new era of moderator
                distance=0; %Resetting distance
            end
        else
            if(P_Store(nn-1)+distance*Ef>mfp) %Make sure we don't go up to our mfps
                P_Store(nn)=distance*Em*0+P_Store(nn-1); %part for moderator
                nn=nn+1; %for the fuel
                P_Store(nn)=P_Store(nn-1)+distance*Ef;
                P=P_Store(nn);
                nn=nn+1; %new era of moderator
                distance=0; %Resetting distance
            end
        end

        end

        %For Debugging Purposes:
        %disp('    Distance    Length    P_Store    P    mfp, theta');
        %disp([distance,Length,P_Store(nn-1),P,distance*Em,acosd(Vector(1,4))]);

```

```

A=exist('distance_fuel','var');
B=exist('distance_before','var');
if(A && debug==1 && B)
    disp('      Fuel D      Before      After      Length');
    disp([distance_fuel,distance_before,distance_after,Length]);
    disp('    Distance      P1      P2      P3      P4');
    findingstuff=find(P_Store(2:end)==0,2);
    disp([distance,P_Store(2),P_Store(3:findingstuff(2))]);
elseif(debug==1)
    disp('    Distance      P1      P2      P3      P4');
    findingstuff=find(P_Store(2:end)==0,2);
    disp([distance,P_Store(2),P_Store(3:findingstuff(2))]);
end

```