Bend Magnet Heat Loads and Out of Orbit Scenarios

Timothy Valicenti
Department of Mechanical Engineering, Brown University

August 12, 2016

Abstract

This paper presents an analytical calculation of the spatial power spectrum emitted from relativistic electrons passing through a series of bend magnets. Using lattice files from the software Elegant, both the ideal and missteered trajectories taken by the beam are considered in determination of the power profile. Calculations were performed for the Advanced Photon Source Upgrade multi-bend-achromat storage-ring. Results were validated with Synrad, a monte-carlo based program designed at CERN. The power distribution and integrated total power values are in agreement with Synrad's results within one percent error. The analytic solution used in this software gives a both quick and accurate tool for calculating the heat load on a photon absorber. The location and orientation can be optimized in order to reduce the peak intensity and thus the maximum thermal stress. This can be used with any optimization or FEA software and gives rise to a versatile set of uses for the developed program. [7]

1 Introduction

1.1 Problem Statement

When accelerated through a magnet field, relativistic electrons emit synchrotron radiation. This radiation creates a spatial power distribution on intercepting surfaces which may be used to calculate the resultant heat load. The problem may be broken up into several simpler steps. These involving computing the ideal path that the bunch of electrons take; adding real orbital errors to this trajectory; ray tracing the emitted photons; and determining the intensity of the power where each ray lands.

1.2 Solution

A parameterization of the ideal path is used, thus discrete time steps are chosen. Though it can be changed, the program uses 100 equidistant time steps per dipole magnet that is analyzed. This will create 100 photons that impact the input photon absorber per bend magnet. For each of these, a vertical spread of 1000 points is created and the full power spectrum is calculated - this number may also be easily changed. This gives a one-hundred-thousand-point mesh of varying density.

Many initial values such as the magnetic field strength, geometrical constants of the magnets, and beam energies are taken from a lattice file created by the program Elegant. A lattice file from Elegant or of identical formatting must be used for the software to work properly. To simplify the equations that follow, the general rotation matrix is given by Equation 1. Any parameterized function centered at the origin and acted upon by the matrix $\overline{\overline{R}}_{\bar{u}}(\theta)$ is rotated counterclockwise about the vector $\bar{u}$ by an angle θ. Figure 1 displays the global coordinate notation used by the program.

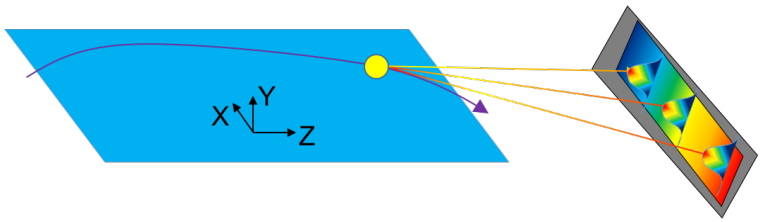

Figure 1: An electron emitting radiation onto a planar surface. Each photon ray creates a vertical distribution. The global ZXY-coordinate system used is analogous to the more common XYZ-cartesian system.

$$\overline{\overline{R}}_{\bar{u}}(\theta) = \begin{bmatrix} cos\ \theta + u_Z^2(1-cos\ \theta) & u_Z u_X(1-cos\ \theta) - u_Y sin\ \theta & u_Z u_Y(1-cos\ \theta) + u_X sin\ \theta \\ u_X u_Z(1-cos\ \theta) + u_Y sin\ \theta & cos\ \theta + u_X^2(1-cos\ \theta) & u_X u_Y(1-cos\ \theta) - u_Z sin\ \theta \\ u_Y u_Z(1-cos\ \theta) - u_X sin\ \theta & u_Y u_X(1-cos\ \theta) + u_Z sin\ \theta & cos\ \theta + u_Y^2(1-cos\ \theta) \end{bmatrix} \tag{1}$$

2 Theory and Method

2.1 Ideal Trajectory

In order to solve for the proper trajectory, values in the lattice files must be used. The bend magnet of interest contains values denoted by subscript 1 while the element preceding it in the lattice file contains values denoted by subscript 0. In a dipole field, the path taken by the electrons follows an arc of a circle. This can be done using a parametrization with a radius of curvature given by Equation 2 [3].

$$\rho = \frac{m(\gamma)v}{q_e B} \tag{2}$$

The values of which to parametrize the arc are contained in the set $t : (0, t_f)$ where t_f is the total time spent in the dipole region given by Equation 3. Δs is the arc length of the curve.

$$t_f = \frac{\Delta s}{v} \tag{3}$$

In order to place the trajectory in the proper location in global ZXY-position space, the curve must be oriented and translated properly. If θ_0 is the angle that the beam enters a dipole field at with respect to the Z-axis, and if $\overline{r}_1$ is the initial position of the entering beam, the parametrization is given by the following equation.

$$\overline{r}(t) = \overline{\overline{R}}_{\overline{Y}}(\theta_0) \begin{bmatrix} \rho sin(\frac{v}{\rho}t) \\ \rho(cos(\frac{v}{\rho}t) - 1) \\ 0 \end{bmatrix} + \overline{r}_1, \qquad t = 0..t_f \tag{4}$$

The initial condition, $\overline{r}_1$, is given by Equation 5. δs is the effective drift length between the magnet of interest and the preceding element.

$$\overline{r}_1 = \begin{bmatrix} Z_0 \\ X_0 \\ 0 \end{bmatrix} + \delta s \begin{bmatrix} cos\ \theta_0 \\ sin\ \theta_0 \\ 0 \end{bmatrix} \tag{5}$$

2.2 Off Orbit Trajectories

In reality, there are orbital errors in the beam trajectory. The lattice files give Courant-Snyder (or Twiss) parameters (β and α) which can be used to define two phase space ellipses - one for errors along the local x-axis and one for errors along the local y-axis. To achieve the values at the start of the magnet, they must be taken from the previous element in the lattice file and translated across a drift if it is present. Given the values contained within a transfer matrix, $M(s_b, s_a)$, between two points on the beam

$$\overline{\overline{M}}(s_b, s_a) = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} \tag{6}$$

one can use the following general translation matrix to solve for the Twiss parameters at the start of a dipole:

$$\begin{bmatrix} \alpha_b \\ \beta_b \\ \gamma_b \end{bmatrix} = \begin{bmatrix} m_{11}m_{22} + m_{12}m_{21} & -m_{11}m_{21} & -m_{12}m_{22} \\ -2m_{11}m_{12} & m_{11}^2 & m_{12}^2 \\ -2m_{21}m_{22} & m_{21}^2 & m_{22}^2 \end{bmatrix} \begin{bmatrix} \alpha_a \\ \beta_a \\ \gamma_a \end{bmatrix} \tag{7}$$

where

$$\gamma = \frac{1 + \alpha^2}{\beta} \tag{8}$$

The transfer matrix for a drift is given by Equation 9.

$$\overline{\overline{M}}(s_1, s_0) = \begin{bmatrix} 1 & \delta s \\ 0 & 1 \end{bmatrix} \tag{9}$$

which leads to the following equations for the Twiss parameters at the start of the bend magnet [5] .

$$\alpha_1 = \alpha_0 - \gamma_0 \delta s \tag{10}$$
$$\beta_1 = \beta_0 - 2\alpha_0 \delta s + \gamma_0 \delta s^2 \tag{11}$$
$$\gamma_1 = \gamma_0 \tag{12}$$

The equations defining the phase space ellipses are given below [6].

$$A_u = \gamma_{x,1} x^2 + \alpha_{x,1} x x' + \beta_{x,1} x'^2 \tag{13}$$
$$A_u = \gamma_{y,1} y^2 + \alpha_{y,1} y y' + \beta_{y,1} y'^2 \tag{14}$$

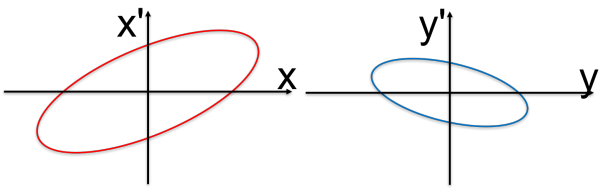

Figure 2: The (x, x') and (y, y') phase space ellipses describing orbital errors

To find a properly missteered path, one first selects a point on or within each ellipse corresponding to the orbital errors. The trajectory then needs to be fixed to reflect the error. To do this transformation properly, the local x-axis must be known (denoted by $\overline{x}_l$). This is orthogonal to the *ideal* direction of travel, $\overline{s}_l$, and the vertical axis.

$$\overline{x}_l = \overline{y}_l \times \overline{s}_l \tag{15}$$

By definition, $x' = \frac{dx}{ds}$ and $y' = \frac{dy}{ds}$. In order to apply the correct rotation matrices one should consider the spherical coordinate system shown in Figure 3. Equations 16 through 18 give the off orbit coordinates in terms of the angles θ and ϕ. If solved for in terms of x' and y', rotation matrices may be used.

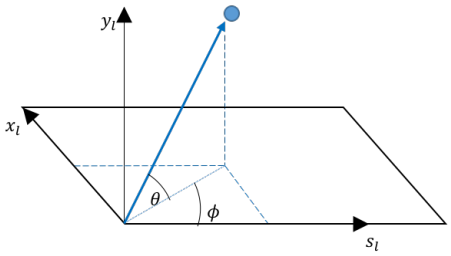

Figure 3: The angles θ and ϕ may solved for in terms of x' and y'.

$$dx = cos\ \theta\ sin\ \phi \qquad dy = sin\ \theta \qquad ds = cos\ \theta\ cos\ \phi \tag{16}$$

$$\frac{dx}{ds} = tan\ \phi \qquad \frac{dy}{ds} = tan\ \theta\ sec\ \phi \tag{17}$$

$$= x' \qquad = y' \tag{18}$$

$$\phi = tan^{-1}(x') \qquad \theta = tan^{-1}(\frac{y'}{\sqrt{1+x'^2}}) \tag{19}$$

$$\overline{r}(t)_{err} = \overline{\overline{R}}_{\overline{x}_{new}}(-\theta)\ \overline{\overline{R}}_{\overline{y}_l}(\phi)\ (r(t) - \overline{r}_1) + (x\ \overline{x}_l + y\ \overline{y}_l + \overline{r}_1) \tag{20}$$

$$\overline{x}_{new} = \overline{\overline{R}}_{\overline{y}_l}(\phi)\overline{x}_l \tag{21}$$

As shown in Equation 20, the ideal trajectory must first be centered on the origin before using the rotation matrices. After orienting the path to contain the x' and y' errors, the parametrization may be translated back to its initial position, $\overline{r}_1$, and then again along the $\overline{x}_l$ and $\overline{y}_l$ axes by the x and y displacement values obtained by each respective phase space ellipse. This should be done to the first magnet used in the analysis. The orbital errors in the following magnets can be found by using transfer matrices.

2.3 Ray Tracing

Ray traces are calculated in two steps. A horizontal distribution of center rays are drawn out by considering each time step in the trajectory parametrization. For each time step, the rays follow in a straight path tangent to the normalized velocity vector, $\overline{t}_1(t)$, and start at the position of the particle, $\overline{r}(t)$. The distance traveled is given in Equation 22 by the parameter d and determines the point, $P(d,t)$.

$$P(d,t) = d * \overline{t}_1(t) + \overline{r}(t) \tag{22}$$

The photon absorber lies along a plane specified by the user. This is given by a normal vector, $\hat{n}_A$, and an arbitrary point lying on the plane, P_0. Using the equation of a planar surface, the distance, d, can be solved for by substituting in the equation for the ray. This gives the points, $P(d^*, t)$, that the rays impact the absorber at at zero vertical angle.

$$\hat{n}_A \cdot (P - P_0) = 0 \tag{23}$$

$$d^*(t) = \frac{\hat{n}_A \cdot (P_0 - \overline{r}(t))}{\hat{n}_A \cdot \overline{t}_1(t)} \tag{24}$$

From each center ray, a set of vertical rays may be drawn. Each one deviates vertically along the direction of the center ray that impacts at $P(d^*, t)$. A secondary plane with a normal vector, $\hat{n}_R$, orthogonal to the vertical direction, $\overline{Y}$, and the direction of the center ray, $\overline{t}_1(t)$, is used. The line of intersection between this plane and the photon absorber contains the end points of all of the rays in the vertical spread at time t. Figure 4 gives a visualization of the vertical spread while Equations 29 and 30 give expressions for the X and Z values on the absorbing surface for a given parameterization of vertical distance, Y. The program currently runs for $Y = \pm 5\ [mm]$ (though this can be easily changed). Each vertical line is centered at $P(d^*, t)$.

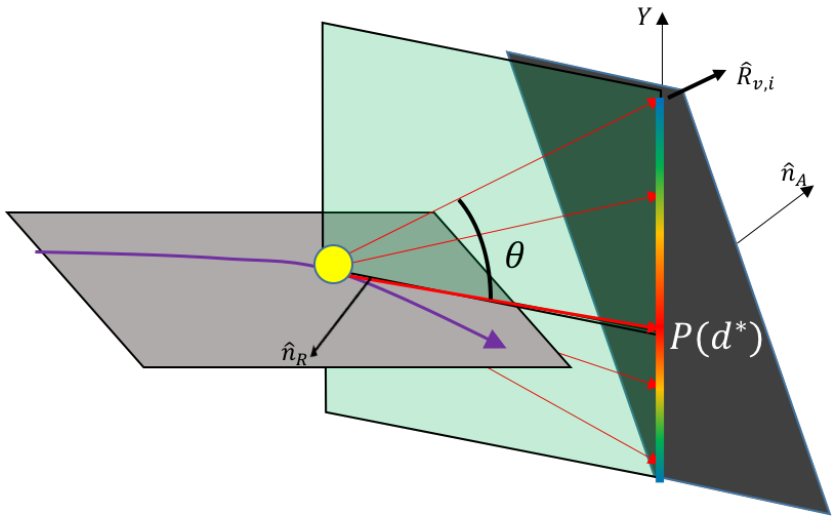

Figure 4: A vertical set of rays may be found for each zero-angle ray located at $P(d^*, t)$. This is done using a plane stretched vertically from the zero-angle ray (See the vertical transparent plane above).

$$0 = \hat{n}_A \cdot (P - P(d^*)) \tag{25}$$

$$= n_{A,Z}(Z - Z_{d^*}) + n_{A,X}(X - X_{d^*}) + n_{A,Y}(Y - Y_{d^*}) \tag{26}$$

$$0 = \hat{n}_R \cdot (P - P(d^*)) \tag{27}$$

$$= n_{R,Z}(Z - Z_{d^*}) + n_{R,X}(X - X_{d^*}) \tag{28}$$

$$X = \left(\frac{n_{R,Z}n_{A,Y}}{n_{A,Z}n_{R,X} - n_{R,Z}n_{A,X}}\right)(Y - Y_{d^*}) + X_{d^*} \tag{29}$$

$$Z = \left(\frac{n_{R,X}n_{A,Y}}{n_{R,Z}n_{A,X} - n_{A,Z}n_{R,X}}\right)(Y - Y_{d^*}) + Z_{d^*} \tag{30}$$

2.4 Power Distributions

For each point $P(d^*, t)$ in the horizontal distribution, the vertical power distribution may be calculated by using the angle, θ, between the zero-angle ray located at $P(d^*, t)$ and the vertical rays previously calculated [4]. The power, $\frac{\partial^2 P}{\partial\theta\partial\psi}$, may be converted from an angular spread to a spatial spread by dividing it by the square of the distance that a given ray travels, D^2. The factor, f, is the projection of the power onto the photon absorber and is found by dotting the planar normal vector and the unit vector describing each vertical ray's tangential direction together [1].

$$\frac{\partial^2 P}{\partial\theta\partial\psi} = f * P_{d,0} \frac{1}{(1+Q^2)^{\frac{5}{2}}} \left(1 + \frac{5}{7}\frac{1}{(1+Q^2)}\right) \qquad Q = \gamma\theta \tag{31}$$

$$P_{d,0}(W/mrad^2) = 5.421 * E(GeV)^4 I(A) B(T) \tag{32}$$

$$\frac{\partial^2 P}{\partial x_A \partial y_A}(W/mm^2) = \frac{1}{D^2}\frac{\partial^2 P}{\partial\theta\partial\psi} \tag{33}$$

$$f = |\hat{R}_{v,i} \cdot \hat{n}_A| \tag{34}$$

3 Results and Discussion

3.1 Accuracy

To verify the accuracy of the program. An analysis was run for the M3.1, M3.2, M2.5, and M2.4 reverse magnets and the planned B-crotch absorber for the APS-U at Argonne National Laboratory. The green fan in Figure 5 in composed of the individual rays from the 400 time steps used in the trajectory parametrization. Figures 6 and 7 show a top down view and a side view of the resultant power spectrum respectively. Numerical data from another distinguished program, Synrad, is overlaid on top of Figure 7 [2]. The total integrated power calculated in Synrad was 2.84 kW while the integrated power given by this program was 2.8428 kW. As shown, the deviation in both the total power and the distribution shape are less than one percent. This verifies the accuracy of the software.

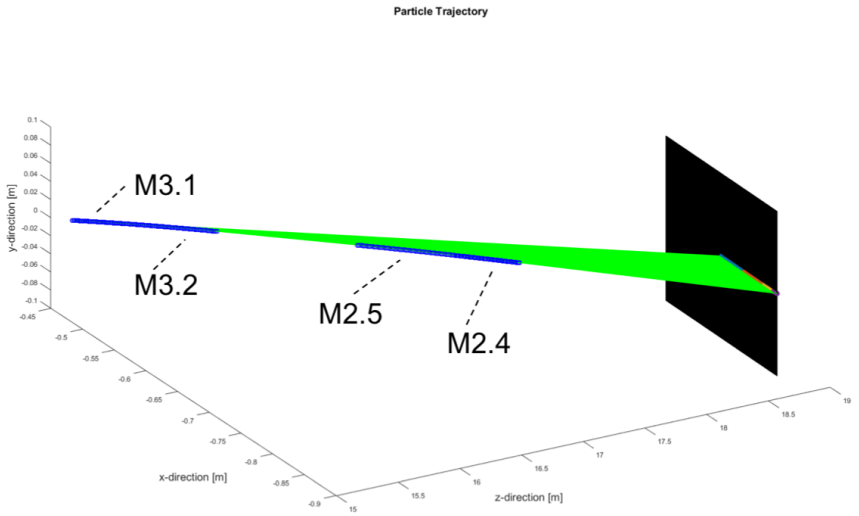

Figure 5: Ray trace results for the M3.1, M3.2, M2.5, and M2.4 magnets radiating onto the B-crotch absorber.

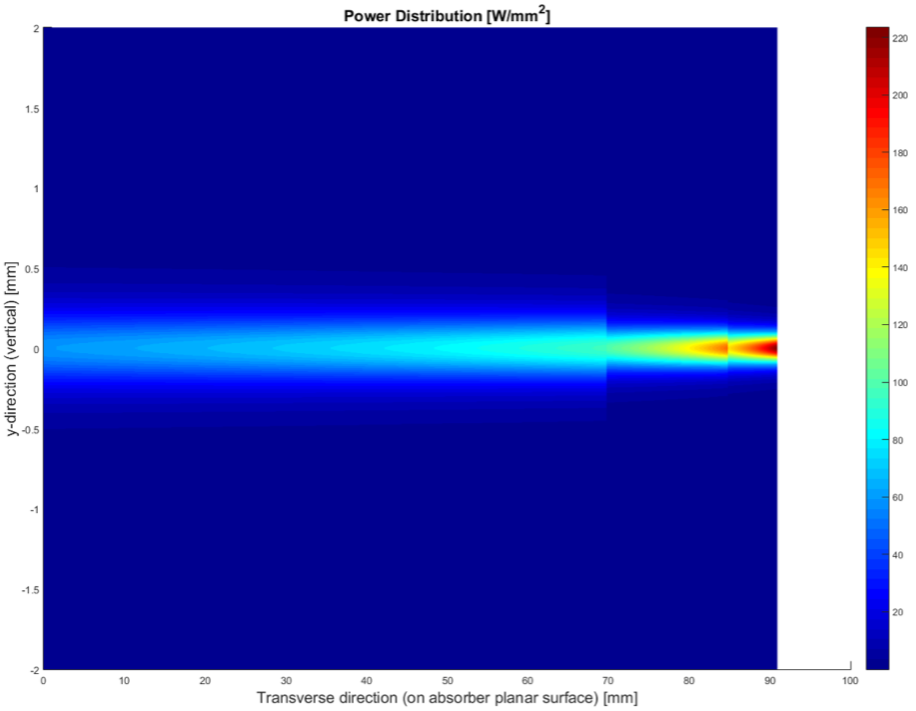

Figure 6: A top down view of the power load.

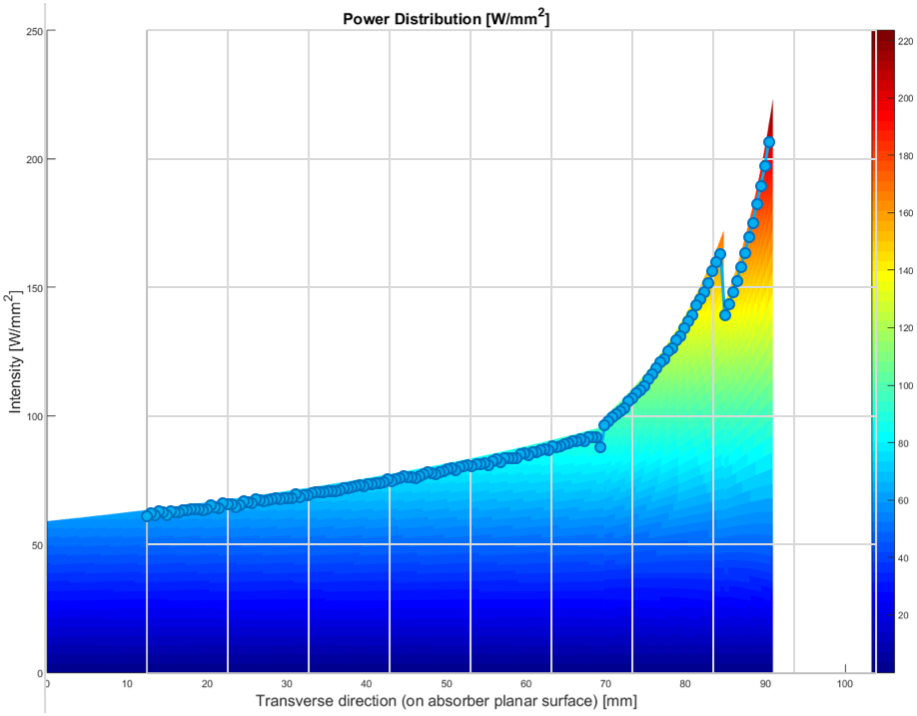

Figure 7: Side view with Synrad data overlaid. Note: the Synrad data only ranges between 12 and 88 [mm]

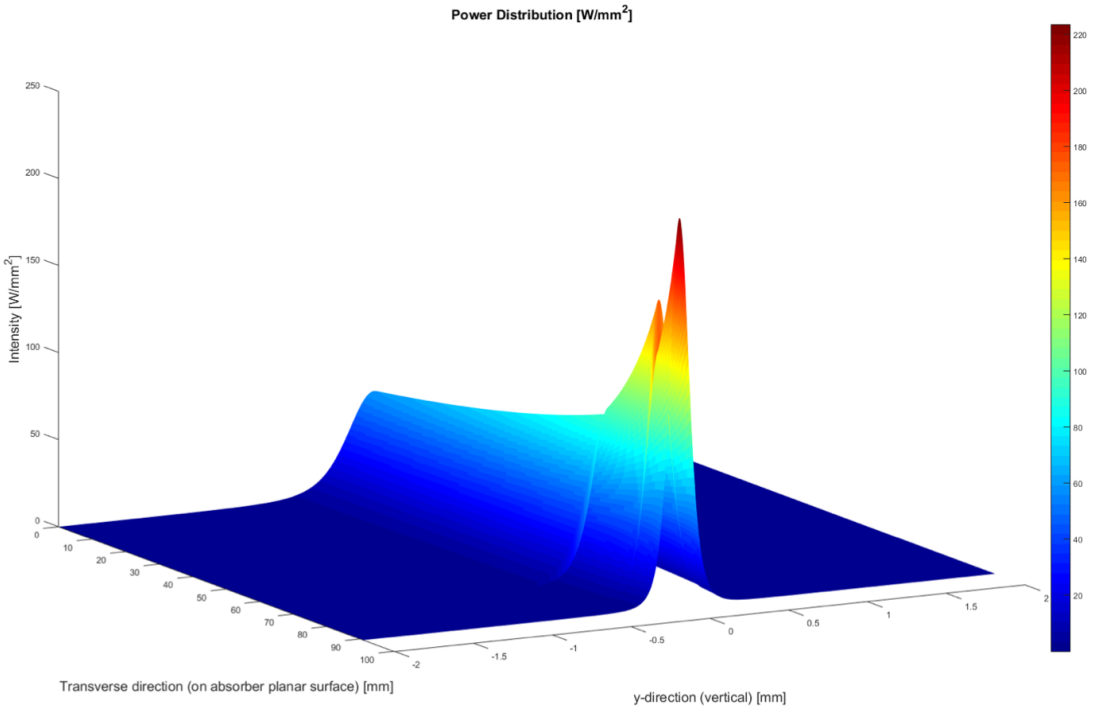

Figure 8: A 3D view of the distribution

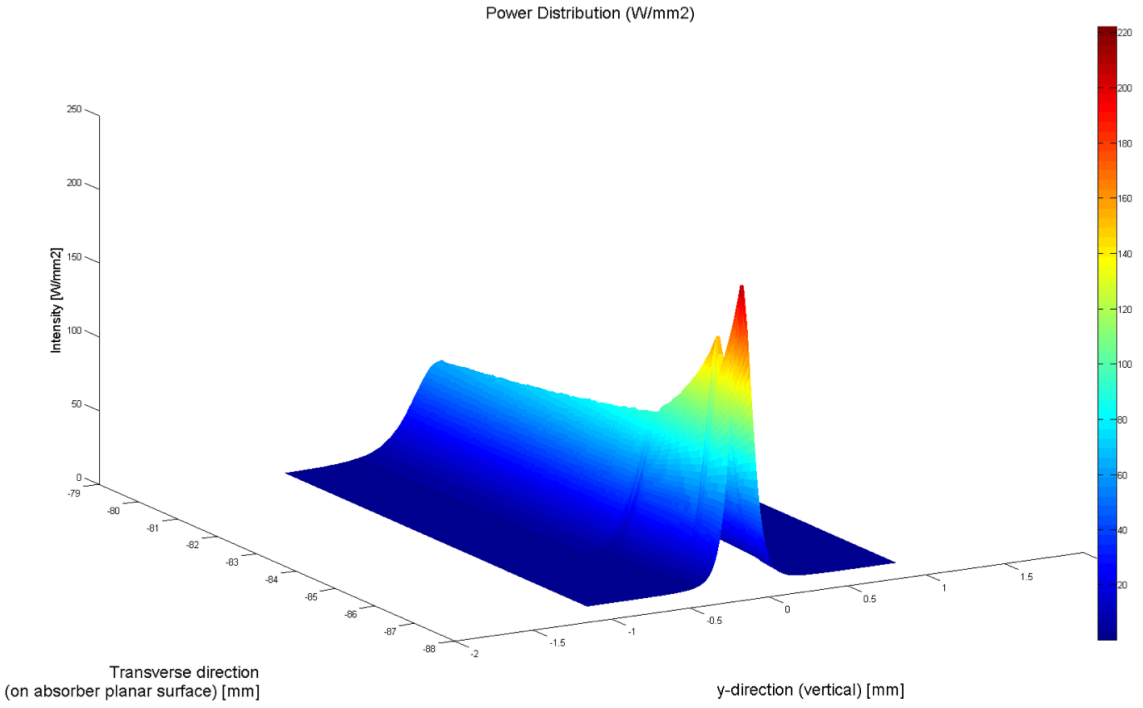

Figure 9: A 3D view of Synrad's distribution. Note: the Synrad data only ranges between 12 and 88 [mm]. The transverse axis is also mislabeled.

3.2 Limitations

There a few limitations with the current state of the program. Five notable ones include (1) using multiple photon absorbers, (2) having a photon absorber normal vector with a non-zero vertical component, (3) using orbital errors with multiple bend magnets, (4) using downstream magnets that don't emit onto the input absorbing surface, and (5) using absorbing surfaces that are non-planar.

3.2.1 Multiple Photon Absorbers

The program applies the heat map to a single photon absorber that is given by the user. This is assumed to be an infinite plane and all regions that intercept photons are plotted. If one wishes to analyze multiple absorbers or an absorber with a gap, running separate simulations is recommended.

3.2.2 Normal Vectors of Non-Zero Y-component

When the code runs to completion, the results are accurate. As it is now, however, the code breaks down when the vertical component of the input normal vector is non-zero. In order to see whether some of the synchrotron radiation overlaps and thus superimposes, an interpolation method has to be used. Matlab's built-in function, *interpn*, requires a very strict monotonically increasing meshgrid to interpolate in space. When the normal vector is titled upwards (or downwards) the mesh may no longer be monotonically increasing and the code throws an error. To fix this, one could either (1) improve the interpolation method or (2) comment it out and instead append the heatmaps together rather than interpolating and then adding. This assumes that the ray traces never overlap however. Commenting out the interpolation code would not show the user whether overlap occurs.

3.2.3 Missteering with Multiple Magnets

As it is now, the code that computes the orbital errors will solve for different possible trajectories of a single magnet. The default creates 64 'extreme' paths that incorporate values such as the maximum x-displacement or the maximum y' value (and all permutations of these values). Unfortunately there is no simple way to match up the missteered paths of one magnet with the missteered paths of the following magnet. In theory, if one starts out with an orbital offset, the orbital errors down stream should be contained on or within the phase space ellipse that corresponds to the location of interest. One may suggest that the current code just start out with an initial position and orientation that reflects an orbital offset and then send it through the rest of the code. The problem with this is that the current program does not handle quadrupoles. If there is a non-zero y' orbital error, the electron will move upwards until a focusing quadrupole is reached. MATLAB's function for reading in excel data (from the Lattice file) does not handle strings and replaces them with 'NaN'. For this, even if the code could handle quadrupoles, there is not a simple method for 'knowing' that the next row of the lattice file to analyze is one. As a result of this issue, the missteering code is currently commented out and set aside for further exploration.

3.2.4 Downstream Magnets

If some of the magnets input into the program for analysis are behind the photon absorber, then no rays will be traced out as none of them land on the absorbing surface. Currently as the code is, the code will break down for the same reason previously mentioned in Matlab's *interpn* function. When the data suddenly cuts off, the mesh grid is no longer monotonically increasing and the interpolation breaks down. As long as the input magnets are realistic in that they precede the photon absorber's position, the code should work well.

3.2.5 Non-planar Absorbers

The program only allows for planar surfaces defined by the input normal vector and position. If special curved surfaces are used, one method would be for the user to input a series of nodes and to write a code that determines photon impact location. If the shape can be written using a 2-variable parameterization (such as a sphere or toroid), that may also be a method for determining ray traces on curved surfaces.

4 Conclusion

This project began with the goal of determining the power distributions on arbitrarily placed planar surfaces. As shown, the program completes this with degrees of accuracy within one percent from the software Synrad. The code is versatile in that it can handle any bend magnet that will emit radiation onto the input absorbing surface. The functions are autonomous and set up the possibility of integrating into other programs such as COMSOL or ANSYS. A particle trajectory created in another program may be used in the functions created for this program. Additionally, the analytic method used gives rise to quick runtimes that may be better suited for optimization processes. Though advantageous in some regard, it also has its limitations. As discussed, the code would benefit from utilizing arbitrary surfaces, stable interpolation, and incorporation of missteering effects. The code is easily adaptable however and this paper gives the user the necessary information to improve the program to meet their needs.

5 References

References

[1] Capatina, Dana. private communication (2016).

[2] Carter, Jason. private communication; Synrad data (2016).

[3] Chao, Alexander Wu., and M. Tigner. Handbook of Accelerator Physics and Engineering. River Edge, NJ: World Scientific, 1999. Print.

[4] Dejus, Roger. "Power Distribution from a Dipole Source." Internal APS Memo (2003): 1-8. Print.

[5] Edwards, D. A., and M. J. Syphers. An Introduction to the Physics of High Energy Accelerators. New York: Wiley, 1993. Print.

[6] Harkay, Katherine. "Maximum Beam Orbit in MBA and Ray Tracing Guidelines." 2nd ser. (2014): 1-9. Print.

[7] Suthar, Kamlesh. private communication (2016).

6 Appendix

6.1 Code

```
%% CalcPower
% This function takes in a point, p0, that lies on an absorbing plane, np,
% the normal vector of that plane, and latRows, a vector of numbers
% indicating which rows of the imported lattice file 'Version6Lattice.xlsx'
% (which must be changed in the code below if the name changes) want to be
% analyzed. With this it finds the intensity distribution, the total
% integrated power, and the peak power level on the specified plane.
function [TotIntVecs, TotalPower, Peak] = CalcPower(p0, np, latRows)
 close all
%Lattice = xlsread('Version6Lattice.xlsx');      % Read in the lattice files.
Lattice = xlsread('NoVP.xls');                  % Read in the lattice files.

n = 100;                                        % Number of points to plot per segment in the horizontal angle.
yN = 1000;                                      % Number of points to plot in the vertical direction.
m = 2*10^-3;                                    % Set range of vertical values to consider in the intensity profile (in [m]).

latRows = latRows - 3;                          % The first 3 rows of the lattice files are non-numeric. This corrects for user input.
segments = length(latRows);                     % Determine the number of segments being used.
totR = zeros(3,n*segments);                     % Define matrix containing the cumulative path.
totV = zeros(3,n*segments);                     % Define matrix containing cumulative velocities.
totCent = zeros(3,n*segments);                  % Define matrix containing cumulative center points.
HeatMaps = zeros(yN,n,4,segments);              % Define matrix containing a heatmap from each segment.
PrevCentCls = 0;                                % Determine the number of values in the 'Centers Matrix' - totCent so far (start at zero).

figure                                          % Note that 'row - 2' is used to correctly chose the row
for iter = 1:segments                           %  that the user asked to analyze. The are offset by 2
    row = latRows(iter);                        % Consider the current row.
    prev = getPrev(row, Lattice);               % Aquire the previous row as well.




    ds = Lattice(row, 2) - ...                  % The arc length is simply sEnd - sStart.
        Lattice(row, 1);
    L = Lattice(row, 5);                        % The effective magnetic length.
    theta0 = Lattice(prev, 8);                  % The angle coming in.
    thetaF = Lattice(row, 8);                   % The angle leaving the segment.
    Z_prev = Lattice(prev, 7);                  % The end Z value of the last member
    X_prev = Lattice(prev, 6);                  % The end X value of the last member
    drift = Lattice(row, 1) - Lattice(prev, 2); % The drift distance between the current and previous members
    r0 = [Z_prev + drift*cos(theta0);...        % The initial position of the beam in the current member
          X_prev + drift*sin(theta0);...
          0];
                                                % Calculate the twiss parameters from the end of the previous segment
    betaX0 = Lattice(prev, 9);                  % The betaX0 twiss parameter.
    betaY0 = Lattice(prev, 10);                 % The betaY0 twiss parameter.
    alphaX0 = Lattice(prev, 13);                % The alphaX0 twiss parameter.
    alphaY0 = Lattice(prev, 14);                % The alphaY0 twiss parameter.
    gammaX0 = (1+alphaX0^2)/betaX0;             % The gammaX0 twiss parameter.
    gammaY0 = (1+alphaY0^2)/betaY0;             % The gammaY0 twiss parameter.

    betaX1 = betaX0 - 2*alphaX0*drift+...       % Translate the betaX value along the drift if present
        gammaX0*drift^2;
    alphaX1 = alphaX0-gammaX0*drift;            % Translate the alphaX value along the drift if present
    betaY1 = betaY0 - 2*alphaY0*drift+...       % Translate the betaY value along the drift if present
        gammaY0*drift^2;
    alphaY1 = alphaY0-gammaY0*drift;            % Translate the alphaY value along the drift if present


    [r,v,B] = DipoleTrajectory(L, ...           % Calculate the trajectory of the current segment
        theta0, thetaF, r0, ds, n);
    totR(:,((iter-1)*n+1):(iter*n)) = r;        % Add the path to the cumulative matrix
    totV(:,((iter-1)*n+1):(iter*n)) = v;        % Add the velocity to the cumulative matrix
%    [rs,vs] = Missteer(r,v, betaX1, ...         % Calculate missteered paths.
%                betaY1, alphaX1, alphaY1, ...
%                DataType, userXs_OR_N, ...
%                userYs, n);

    [centers, rays, np, t2, count] = ...        % Calculate ray traces.
        RayTrace(r, v, p0, np, n);

    if count > 0
        Intensity =  Spectrum(centers, ...      % Find the power distribution.
            rays, np, t2, B, yN, m);
        HeatMaps(:,:,:,iter) = Intensity;       % Add the current intensities to the matrix of heatmaps.
    end

    totCent(:,PrevCentCls+1:...
        PrevCentCls+count) = ...                % Add the centers to the cumulative matrix.
        centers;
    PrevCentCls = ...                           % Update the most previous index value of the 'Centers Matrix'
      PrevCentCls+count;
end

totCent = totCent(:,1:PrevCentCls);
TotInt = InterpMaps(HeatMaps);                  % Calculate the superposition of heatmaps by interpolating
[TotalPower, Peak] = ...
    TotalP(TotInt, totCent, np);                % Integrate the distribution
TotIntVecs = Vectorize(TotInt);                 % Put into form easily read by other programs (see 'Vectorize' script).
 end
```

```m
%% DipoleTrajectory
% This code takes in initial conditions and finds the ideal path
% that bunches of electron would travel through.
function [r,v,B] = DipoleTrajectory(L, theta0, thetaF, r0, ds, n)
% Define Global Parameters & Constants
Y = [0;0;1];                        % Global Y-direction (vertical).
c = 2.99792458*10^8;                % Speed of Light in [m/s].
KE = 6*10^3;                        % Kinetic Energy in [MeV].
m_e = 0.511;                        % Single Electron mass in [MeV/c^2].
m_eSI = 9.10938356*10^-31;          % Single Electron mass in SI in [kg].
q_e = -1.60217662*10^-19;           % Single Electron charge in [C].
E = KE + m_e;                       % Total Energy in [MeV].
gma = E/m_e;                        % Relativisitc gamma in [ ].
p = sqrt(E^2-m_e^2);                % Momentum in [MeV/c].
V = c*sqrt(1-(1/gma)^2);            % Velocity in [m/s].
Bp = p/300;                         % Beam rigidity in [T-m].

r = zeros(3,n);                     % Time Dependent Position Vectors.
v = zeros(3,n);                     % Time Dependent Velocity Vectors.
%% Calculated Parameters
dTh = thetaF - theta0;              % Kick given by the dipole [rad].
B = (dTh/L)*Bp;                     % Magnetic field strength [T].
rho = (gma*m_eSI*V)/(q_e*B);        % Bending radius of the beam in a dipole in [m].
w = V/rho;                          % Equivelent dipole frequency in [s^-1].
t_0 = 0;                            % Initial start time in [s].
t_f = ds/V;                         % End time of the particle in the dipole in [s].
t = linspace(t_0,t_f, n);           % Time step vector.

r(1,:) = rho*sin(w*t);              % Z: Forward component, z (note that this is the x-value
                                    %  in MATLAB's plotting function and different from
                                    %  the electron's longitudinal component, s).
r(2,:) = rho*(cos(w*t)-1);          % X: Transverse component, x (note that this is the
                                    %  y-value in MATLAB's plotting function).
r(3,:) = zeros(1,n);                % Y: Assume the ideal y-component is 0.

v(1,:) = V*cos(w*t);                % Vz: z-component of velocity in [m/s].
v(2,:) = -V*sin(w*t);               % Vx: x-component of velocity in [m/s].
v(3,:) = zeros(1,n);                % Vy: assume the velocity in y is also zero.
%% Rotation and Translation from initial conditions
Rz = RotateAxis(theta0, Y);          % Rotation Matrix about the vertical axis.
T = makeGrid(r0,n);                 % Translation matrix.
r = Rz*r + T;                       % Rotate and translate the path.
v = Rz*v;                           % Adjust the velocity to the rotation.
%% Plotting
hold on
plot3(r(1,:), r(2,:), r(3,:), 'b-o')
title({'Particle Trajectory'}, 'Fontsize', 15, 'FontName', 'Arial')
xlabel('z-direction [m]', 'Fontsize', 15, 'FontName', 'Arial')
ylabel('x-direction [m]', 'Fontsize', 15, 'FontName', 'Arial')
zlabel('y-direction [m]', 'Fontsize', 15, 'FontName', 'Arial')
%% Aditional Plotting (uncomment if desired)
% %% Calculate the full circle that the trajectory is a part of
% m = 100*n;                          % use more points to keep a similarly dense circle
% t_C = linspace(0,2*pi/w, m);        % repeat the above steps to obtain the trajectory
% x_C = rho*(cos(w*t_C)-1);
% y_C = zeros(1,m);
% z_C = rho*sin(w*t_C);
% r_C = [z_C; x_C; y_C];
% r_C = Rz*r_C + T;
%
% %% Calculate naive path of APS based solely on it's effective radius
% R_APS = -351.183/2;                 % radius of APS-U
% t_APS = linspace(0,2*pi, 10*m);     % repeat steps on the parameterization
% x_APS = R_APS*(cos(t_APS)+1);
% z_APS = R_APS*sin(t_APS);
% plot(r_C(1,:), r_C(2,:), '-')
% plot(z_APS, x_APS, '-')
% legend('Extrapolated','Trajectory','Naive APS-U')
hold off
end
```

```
%% Missteer
% If used, this function takes in twiss parameters and calculates desired
% missteered paths. The user must input 'extrema' if they want orbital
% erros that include the extreme points on the phase space ellipses such as
% x_max or y'_max and whatnot. If the user types 'distribution' they must
% input a number N to choose N points on the phase space ellipse. If the
% user wants to input their own set of errors they must type 'custom.'
function [rs,vs] = Missteer(r,v, betaX, betaY, alphaX, alphaY, DataType, userXs_OR_N, userYs, n)
%% Define Global Parameters & Constants
Y = [0;0;1];                                % Global Y-direction (vertical).
bx_max = 8.177;                             % The maximum value of the beta function at the limiting aperature in x [m]
a_x = 9*10^-3;                              % The chamber half-aperture in x [m]
A_x = (a_x^2)/(bx_max);                     % The motion invariant for all beams in (x,x') [m]

by_max = 5.890;                             % The maximum value of the beta function at the limiting aperature in y [m]
a_y = 3*10^-3;                              % The chamber half-aperture in y [m]
A_y = (a_y^2)/(by_max);                     % The motion invariant for all beams in (y,y') [m]
%% APS-U Specifications
gammaX = (1+alphaX^2)/betaX;                % Define the gamma twiss paramter in x [m^-1]
gammaY = (1+alphaY^2)/betaY;                % Define the gamma twiss paramter in y [m^-1]

xM = sqrt(A_x*betaX);                       % The maximum transverse error in (x,x') [m]
xp_xM = -alphaX*sqrt(A_x/betaX);            % The corresponding x' value [rad]
xpM = sqrt(A_x*gammaX);                     % The maximum angular deflection error in (x,x') [rad]
x_xpM = -alphaX*sqrt(A_x/gammaX);           % The corresponding x value [m]

yM = sqrt(A_y*betaY);                       % The maximum transverse error in (y,y') [m]
yp_yM = -alphaY*sqrt(A_y/betaY);            % The corresponding y' value [rad]
ypM = sqrt(A_y*gammaY);                     % The maximum angular deflection error in (y,y') [rad]
y_ypM = -alphaY*sqrt(A_y/gammaY);           % The corresponding y value [m]

xp_0 = sqrt(A_x/betaX);                     % The deflection at x = 0 [m] given in [rad]
yp_0 = sqrt(A_y/betaY);                     % The deflection at y = 0 [m] given in [rad]
x0 = sqrt(A_x*gammaX);                      % The displacement error in x given in [m] for x' = 0 [rad]
y0 = sqrt(A_y*gammaY);                      % The displacement error in y given in [m] for y' = 0 [rad]
%% Initial Conditions
if strcmp(DataType,'extrema')
    % 8 Edge Case Points in x
    Xs = [[xM, xp_xM];...                   % x = X_max
        [0,xp_0];...                        % x = 0 with +x'(0)
        [0,-xp_0];...                       % x = 0 with -x'(0)
        -[xM, xp_xM];...                    % the negated point of (x,x') at x = X_max
        [x_xpM, xpM];...                    % x corresponds to x' = X'_max
        [x0, 0];...                         % x corresponds to x' = 0
        [-x0, 0];...                        % x corresponds to x' = 0 but negated
        -[x_xpM, xpM]];                     % the negated point of (x,x') at x' = X'_max

    % 8 Edge Case Points in y
    Ys = [[yM, yp_yM];...                   % y = Y_max
        [0,yp_0];...                        % y = 0 with +y'(0)
        [0,-yp_0];...                       % y = 0 with -y'(0)
        -[yM, yp_yM];...                    % The negated point of (y,y') at y = Y_max
        [y_ypM, ypM];...                    % y corresponds to y' = Y'_max
        [y0, 0];...                         % y corresponds to y' = 0
        [-y0, 0];...                        % y corresponds to y' = 0 but negated
        -[y_ypM, ypM]];                     % The negated point of (y,y') at y' = Y'_max

elseif strcmp(DataType,'distribution')
    if (isfloat(userXs_OR_N) && ...         % Check that the user input a float value and
            (rem(userXs_OR_N,1) == 0))      % that it is an integer fo the number of points, N
        N = userXs_OR_N;
    else
        error('The 8th argument must be an integer');
    end

    Xs = zeros(2*N,2);
    count = 1;
    for i = linspace(-xM,xM,N)              % Generate 2*N points on the (x,x') ellipse
        xp_i1 = ...
            (-2*alphaX*i + sqrt((2*alphaX*i)^2-...
            4*betaX*(gammaX*i^2-A_x)))/(2*betaX);
        xp_i2 = ...
            (-2*alphaX*i - sqrt((2*alphaX*i)^2-...
            4*betaX*(gammaX*i^2-A_x)))/(2*betaX);
        Xs(count,:) = [i, xp_i1];           % Add the pairs to the Xs matrix
        Xs(count+N,:) = [i, xp_i2];
        count = count + 1;
    end
```

```
        Ys = zeros(2*N,2);
        count = 1;
        for i = linspace(-yM,yM,N)                  % Generate 2*N points on the (y,y') ellipse
            yp_i1 = ...
                (-2*alphaY*i + sqrt((2*alphaY*i)^2-...
                4*betaY*(gammaY*i^2-A_y)))/(2*betaY);
            yp_i2 = ...
                (-2*alphaY*i - sqrt((2*alphaY*i)^2-...
                4*betaY*(gammaY*i^2-A_y)))/(2*betaY);
            Ys(count,:) = [i, yp_i1];               % Add the pairs to the Ys matrix
            Ys(count+N,:) = [i, yp_i2];
            count = count + 1;
        end

    elseif strcmp(DataType,'custom')
        if (ismatrix(userXs_OR_N) && ...            % Check that the user put in 2 matrices
                ismatrix(userYs))
            Xs = userXs_OR_N;
            Ys = userYs;

            [~, cXs] = size(Xs);
            [~, cYs] = size(Ys);


            if (((cXs ~= 2) || (cYs ~= 2)) || ...   % Check that each matrix has only 2 columns
                    (~isnumeric(Xs) || ~isnumeric(Ys)))
                error('The 8th and 9th arguments must be 2-column matrices.');
            end
        else
            error('The 8th and 9th arguments must be 2-column matrices.');
        end
    else
        error(strcat(...
            sprintf('Please input a DataType for plotting missteered values. \n'),...
            sprintf(' The options are: "extrema" (plots all combinations of points \n'),...
            sprintf(' including at least one extreme value on the phase spase ellipse); \n'),...
            sprintf(' "distribution" (uses a distribution of 2N equally spaced points \n'),...
            sprintf(' in x-xp space and in y-yp space to form 4N^2 combinations of \n'),...
            sprintf(' points); and "custom" (user must input 2 matrices containing \n'),...
            sprintf(' pairs of x-xp and y-yp points respectively).')))
    end
    %% Find All Desired Permutations
    [Lx, ~] = size(Xs);
    [Ly, ~] = size(Ys);
    Tot = Lx*Ly;                                    % Acquire the total number of permutations
                                                    %  of point-pairs. That is, pairs of (x,x') with (y,y').
    ICs = zeros(Tot,2,2);                           % There are Lx (x,x') points and Ly (y,y')
                                                    %  points. That leaves 'Tot' combinations
                                                    %  of Initial Conditions.
    k = 1;                                          % Insert the initial conditions into the matrix
    for i = 1:Lx
        for j = 1:Ly
            ICs(k,1,:) = Xs(i,:);
            ICs(k,2,:) = Ys(j,:);
            k = k + 1;
        end
    end
    %% Find the correctly misteered paths
    rs = zeros(3,n,Tot+1);                          % Prepare a matrix containing all of the paths (including the ideal path).
    vs = zeros(3,n,Tot+1);                          % Prepare a matrix containing all of the velocities (including the ideal path's).
    rs(:,:,1) = r;                                  % Let the first layer of the matrix contain the ideal path, r.
    vs(:,:,1) = v;                                  % Let the first layer of the matrix contain the ideal velocity, v.

    r0 = r(:,1);                                    % The current initial displacement.
    T0 = makeGrid(r0,n);                            % Initial Translation matrix.

    r_temp = r - T0;                                % Subtract the offset to simplify rotations about the origin.

    v0 = v(:,1);

    x_true = ...                                    % Use the initial velocity vector to determine the true x-transvere
        cross(Y, v0)/norm(cross(Y, v0));            %  direction in the particle's frame of reference.

    hold on
    for i = 1:Tot
        thetx = ICs(1,1,2);                         % The angle about which to rotate the path
                                                    %  around y-axis (z-axis in Cartesian).

        thety_true = ICs(1,2,2);                    % The angle about which to rotate the path
                                                    %  around x-axis (y-axis in Cartesian).

        thety = atan(tan(thety_true)*cos(thetx));   % Define the actual angle that must be used
                                                    %  in order to get the rotation correctly.

        x_err = ICs(1,1,1);                         % The current displacement error in x.
        y_err = ICs(1,2,1);                         % The current displacement error in y.

        Rz = RotateAxis(thetx, Y);                  % Rotation Matrix about the vertical axis.

```

```
168
169 -           axis = Rz*x_true;                       % Define the next axis that the trajectory will be
170                                                     %  rotated with respect to.
171
172 -           Ry_true = RotateAxis(-thety, axis);     % Create the corresponding rotation matrix.
173
174 -           T = [[x_err*x_true(1),0,0];...          % Translation matrix based off of the displacement errors.
175                [0,x_err*x_true(2), 0];...
176                [0, 0, y_err]];
177
178 -           rs(:,:,i+1) = ...                       % Rotate and translate to get the missteered path.
179                Ry_true*Rz*r_temp + ...
180                (T+T0)*ones(3,n);
181
182 -           vs(:,:,i+1) = Ry_true*Rz*v;             % Apply the rotations on the velocity to correct it.
183
184
185 -           plot3(rs(1,:,i+1), ...                  % Plot the missteered path.
186                    rs(2,:,i+1), rs(3,:,i+1), '--')
187 -       end
188 -       hold off
189
190 -   end
```

```
1           %% RayTrace
2           % r         - the time-dependent position vector of a single particle: r = <rx,ry,rz>
3           % v         - the velocity of the particle
4           % p0        - a point on the plane
5           % np        - normal vector of the absorbing plane
6           % nTot      - the number of points in r or v (could also be found using the
7           %             'length' or 'size' methods.
8           function [centers, rays, np, t2, count] = RayTrace(r, v, p0, np, nTot)
9           %% Define Global Parameters & Constants
10 -        Y = [0;0;1];
11 -        Ymat = [zeros(1,nTot); ...
12                 zeros(1,nTot); ...
13                 ones(1,nTot)];                   % populate a matrix of size nTot with the Y vectors
14
15 -        vmags = (dot(v,v)).^(0.5);               % calculate the vectors t1, t2, and n (normal)
16 -        vMagMat = [vmags; vmags; vmags];         %  in the moving frame of the particle.
17 -        t1 = v./vMagMat;                         % Tangent vector 1.
18 -        nVec = Ymat;                             % The normal vector of the particle path.
19
20 -        t2 = cross(nVec,t1);                     % Unit tangent 2.
21 -        t2mags = (dot(t2,t2)).^(0.5);
22 -        t2MagMat = [t2mags; t2mags; t2mags];
23 -        t2 = t2./t2MagMat;                       % Normalize it.
24
25          %% Planar Geometry of the Absorbing Surface
26 -        pp = ceil(nTot*(7/8));                   % Unless given by the user, take the absorber
27                                                   %  to be 7/8s through the particle's path.
28 -        if (~exist('p0', 'var'))
29 -            p0 = r(:,pp);
30 -        end
31 -        if (~exist('np', 'var'))
32 -            tp1 = t2(:,pp);                      % The transverse vector of the plane
33 -            np = cross(Y, tp1);                  % The normal vector of the plane
34 -        end
35
36 -        npmags = (dot(np,np)).^(0.5);
37 -        npMagMat = [npmags; npmags; npmags];
38 -        np = np./npMagMat;                       % Normalize the normal vector in case not already done.
39
40          %% Calculating The Ray Traces
41 -        hold on
42 -        rays = zeros(3,2,nTot);                  % Rays are defined by a start and end point.
43 -        centers = zeros(3,nTot);                 % Centers are defined by a coordinate (ray end point).
44
45 -        count = 0;                               % Consider the rays at each iteration of time
46 -        for i = 1:nTot
47 -            ln0 = r(:,i);                        % ln0, the starting point of the ray, is guaranteed to be on the ray
48 -            M = t1(:,i);                         % M, the unit tangent of the ray, is the slope of the ray
49
50 -            if (dot(M,np) ~= 0)                  % Use the relation that ((P - P0) dot np) = 0. Plug in for P the equation
51 -                d = dot((p0 - ln0), np)/dot(M,np);%  of the line d*M+ln0, with parameter d. If dot(M,np) = 0 then they are
52                                                   %  orthogonal and the ray won't hit the absorber. Otherwise, solve for d.
53 -                if (d > 0)                       % If d is negative it means the ray must travel backwards. Ignore these.
54 -                    centers(:,count+1) = d*M+ln0;% The centers are given at the value of d.
55 -                    rays(:,:,count+1) = ...
56                         [ln0,centers(:,count+1)]; % The rays start at ln0 and end at the center point
57
58 -                    plot3(rays(1,:,1),...        % Plot the ray
59                         rays(2,:,1),...
60                         rays(3,:,1),'g-')
61
62 -                    count = count + 1;           % Increment the count.
63 -                end
64 -            end
65 -        end
66 -        centers = centers(:,1:count);            % Truncate the matrices in case some rays weren't caught (if d <= 0)
67 -        rays = rays(:,:,1:count);
68 -        scatter3(centers(1,:),centers(2,:),centers(3,:))% Plot the center points
69 -        hold off
70
71 -      end
```

```
%% Spectrum
% This function takes in the center points hit by the rays at zero vertical
% angle, the rays themselves, the normal vector of the absorbing plane, the
% matrix of unit tangent vectors that are perpendicular to the particle
% path, 'yN' - the number of vertical points to plot from each ray, and 'm'
% the vertical distance to cover (plus and minus 'm' that is), and with
% these it finds the power distribution along the absorbing plane.
function Intensity = Spectrum(centers, rays, np, t2, B, yN, m)
% Define Global Parameters & Constants
KE = 6*10^3;                            % Kinetic Energy in [MeV]
I = 200;                                % Storage Ring current in [mA]
m_e = 0.511;                            % Single Electron mass in [MeV/c^2]
E = KE + m_e;                           % Total Energy in [MeV]
gma = E/m_e;                            % Relativisitc gamma in [ ]
Pd0 = 5.421*(E/1000)^4*...              % Peak intensity value
    I*(10^-3)*abs(B);

[~, cls] = size(centers);               % Obtain the number of center points to consider

nz = np(1);                             % split up the coordinates of the normal vector of the plane
nx = np(2);
ny = np(3);

y_rad = linspace(-m,m,yN);              % discretize the vertical direction with yN points

Intensity = zeros(yN,cls,4);            % Create intensity matrix meshgrid. A column for each center point
                                        %  and a row for each vertical position.


hold on
for i = 1:cls                           % Iterate over every center point

    t2z = t2(1,i);                      % Break the second (outward) unit tangent vectors of the rays into components
    t2x = t2(2,i);
    % t2y = t2(3,i);                    % The y-component is always zero

    Cz = centers(1,i);                  % Break the current ray center points into components
    Cx = centers(2,i);
    Cy = centers(3,i);
                                        % Given that one uses the vertical distance, y, as a parameter, one can solve for
                                        %  x and z for which the vertical 'spray' of radiation covers from a given ray.
    y_rad = y_rad + Cy;                 % Have the vertical distribution centered around the center point.



    x_rad = (ny*t2z)/(t2x*nz-nx*t2z)*(y_rad-Cy)+Cx;          % The formula for the x-component on the line intersected by the absorbing plane
                %  and the plane created by the ray and the vertical direction, Y.

    z_rad = (ny*t2x)/(nx*t2z-t2x*nz)*(y_rad-Cy)+Cz;       %  and the plane created by the ray and the vertical direction, Y.


    Intensity(:,i,1) = z_rad;           % First Layer: global Z values;
    Intensity(:,i,2) = x_rad;           % Second Layer: global X values;
    Intensity(:,i,3) = y_rad;           % Third Layer: global Y values;

    ray = rays(:,:,i);                  % Consider the ith ray.
    vertRays = ...                      % Find the 'yN' amount of rays deviating in the vertical angular direction.
        [z_rad;x_rad;y_rad] ...
        - makeGrid(ray(:,1),yN);
    centRay = ...                       % Create a grid of the center line ray (one for each vertically deviated ray)
        makeGrid(ray(:,2)-ray(:,1),yN);
    vrayMags = ...                      % Define the distance these rays travel
        dot(vertRays,vertRays).^0.5;
    crayMags = ...                      % Define the distance the center ray travels
        dot(centRay,centRay).^0.5;
    cosTs = ...                         % Find the cosine of the angle between the two rays
        dot(vertRays, centRay)./...
        (vrayMags.*crayMags);
    thet = acos(cosTs);                 % Calculate the angle
    X = thet*gma;                       % Make a change of variables from theta -> X where X is some variable

    vrUnit = vertRays./...              % Make the vertical rays into unit vectors
        [vrayMags;vrayMags;vrayMags];

    normMat = makeGrid(np,yN);          % Create a grid of the vector normal to the absorbing plane

    f = abs(dot(vrUnit, normMat));      % Find the projection fraction of each ray onto the plane.

    Intensity(:,i,4) = ...              % Fourth Layer: Intensity values;
       Pd0*((1+X.^2).^(-5/2)).*...
       (1+(5/7)*(X.^2)./(1+X.^2))./...
       (vrayMags.^2).*f;

    y_rad = y_rad - Cy;                 % subtract back of Cy to 'zero' the y-values for the next iteration.

end
hold off
end
```

```
%% TotalP
% This code takes in the center points, the vector normal to the absorbing
% plane and the intensity values and coordinates and integrates across the
% surface to find the total power in [W].
function [TotalPower, Peak] = TotalP(Intensity, centers, np)
Y = [0;0;1];                            % The gloabl Y-direction (vertical)
[rws, cls, ~] = size(Intensity);        % Obtain the number of center points to consider

if (cls > 0)
    s = norm([centers(1:2,1);0] - ...   % The tranverse distance covered on the absorbing plane
        [centers(1:2,end);0]);
    xPlane = ([centers(1:2,end);0] - ...% The unit vector 'defining this distance'
        [centers(1:2,1);0])/s;
    yPlane = cross(np,xPlane);          % The other transverse direction on the absorbing plane
    yFac = 1/abs(dot(yPlane, Y));       % The fraction that the 'vertical' distance on the plane changes by
    y_mx = max(max(Intensity(:,:,3)));  % The maximum y-value.
    y_mn = min(min(Intensity(:,:,3)));  % The minimum y-value.
    dy = (y_mx-y_mn)/(rws - 1);         % The differential y-value
    dyPlane = yFac*dy;                  % The new bigger differential from tilting at an angle

    C0 = centers(:,1);                  % C0 is the first center point coming from the ray from r0
    tDist = zeros(rws,cls);             % Create a vector of values representing the transverse
                                        %  distance along the absorbing surface.
    toInteg = zeros(2,cls);             % Create a vector of values to integrate later.

    for i = 1:cls                       % Iterate over each center point
        D = norm(centers(:,i)-C0);      % The transverse distance on the absorbing surface
        tDist(:,i) = D*ones(rws,1);     % Place into new 2D coordinate matrix

        toInteg(1,i) = sum((10^6)*...   % Perform a trapezoid sumation rule in the vertical direction
            (Intensity(2:end,1,4) + ... %  to a collapse on a single [W/m] value on a given center point.
            Intensity(1:end-1,1,4))*...
            (dyPlane/2));

        toInteg(2,i) = D;               % Add the transverse distance this once-integrated value lies at.
    end

    heights = ...                       % Calculate the heights of the trapezoid rule rectangles for the 2nd integration
        toInteg(1,2:end) + toInteg(1,1:end-1);
    bases = ...                         % Calculate the bases of the trapezoid rule rectangles for the 2nd integration
        toInteg(2,2:end) - toInteg(2,1:end-1);

    TotalPower = sum(0.5*heights.*bases);   % Sum the trapezoid areas to achieve a total power in [W]
    Peak = max(max(Intensity(:,:,4)));      % The peak value is just the max intensity point.
    %% Plotting the Absorber
    hold on
    lin1 = -xPlane;                     % The transverse direction of the absorber
    lin2 = cross(lin1, np);             % The vertical direction on the absorber
    planeC = lin1*s+centers(:,end);     % Approximate center point of the plane of absorbtion
    ed1 = s*lin1+planeC;                % Trace out to two edge points
    ed2 = -s*lin1+planeC;
    cr1 = -s*lin2+ed1;                  % Trace out to four corner points
    cr2 = s*lin2+ed1;
    cr3 = s*lin2+ed2;
    cr4 = -s*lin2+ed2;
    corners = [cr1, cr2, cr3, cr4]';
    patch(corners(:,1),...
          corners(:,2),...
          corners(:,3),'k')             % Create the planar surface

    hold off
    %% Plotting the Heat Map
    figure
    surface(1000*tDist,...              % Convert the transverse distances to [mm]
            1000*yFac*Intensity(:,:,3),...  % Convert the vertical distances to [mm]
            Intensity(:,:,4),...
            'EdgeColor','none')
    colorbar
    colormap('jet')
    hold on
    title({'Power Distribution [W/mm^2]'}, 'Fontsize', 15, 'FontName', 'Arial')
    xlabel('Transverse direction (on absorber planar surface) [mm]', 'Fontsize', 15, 'FontName', 'Arial')
    ylabel('y-direction (vertical) [mm]', 'Fontsize', 15, 'FontName', 'Arial')
    zlabel('Intensity [W/mm^2]', 'Fontsize', 15, 'FontName', 'Arial')
    hold off
end

end
```

```
%% getPrev
% This function takes in the lattice file and the current row being
% considered and find the first row above it that isn't a vertex point.
function prev = getPrev(row, Lattice)
count = 1;                                          % Record how many rows are checked.
[rws, ~] = size(Lattice);                           % Acquire the total and hence the number of the last row.
prev = row - 1;                                     % Set the previous row to be one before the current row.

if (prev <= 0)                                      % If the current row is the first row, the previous loops
    prev = rws;                                     %  back and is the last row.
end

currPrev = Lattice(prev, :);                        % Acquire the information held in the previous row.

while (sum(abs(currPrev(9:16))) == 0)               % If all parameters are zero one knows this is a VERTEX-POINT.
    if (count >= rws)                               % If all rows have been considered and failed, there must be a
        error('Bad Lattice Files or Row Input Numbers') %   problem with the input or lattice file.
    end

    prev = prev - 1;                                % Otherwise decrement the previous row by 1.

    if (prev <= 0)                                  % If the current row is the first row, the previous loops
        prev = rws;                                 %  back and is the last row.
    end

    currPrev = Lattice(prev, :);                    % Reset the information of the previous row.
    count = count + 1;                              % Increment the count.
end

end
```

```
%% makeGrid
% This function takes in a vector, u, and a number, N, and creates a
% matrix containing N columns each of vector u.
function G = makeGrid(u,N)
 rws = length(u);    % Obtain the number of rows in the vecotr, u.
 O = ones(rws,N);    % Create a 'rws by N' matrix of ones
 T = zeros(rws);     % create a 'rws by rws' matrix of zeros
 for i = 1:rws
     T(i,i) = u(i);  % For every element in u, place it along the diagnol of T.
 end
 G = T*O;            % Multiply T and O to create a matrix of u's.
 end
```

```
%% RotateAxis
% This code takes in an angle, theta, and a unit vector, u, about
% which to create a rotation matrix about.
function Ru = RotateAxis(theta, u)
% Normalize u if needed.
u = u/norm(u);
% Define the components of the rotation matrix, Ru.
R11 = ...
    cos(theta)+u(1)^2*(1-cos(theta));
R12 = ...
    u(1)*u(2)*(1-cos(theta))-u(3)*sin(theta);
R13 = ...
    u(1)*u(3)*(1-cos(theta))+u(2)*sin(theta);
R21 = ...
    u(2)*u(1)*(1-cos(theta))+u(3)*sin(theta);
R22 = ...
    cos(theta)+u(2)^2*(1-cos(theta));
R23 = ...
    u(2)*u(3)*(1-cos(theta))-u(1)*sin(theta);
R31 = ...
    u(3)*u(1)*(1-cos(theta))-u(2)*sin(theta);
R32 = ...
    u(3)*u(2)*(1-cos(theta))+u(1)*sin(theta);
R33 = ...
    cos(theta)+u(3)^2*(1-cos(theta));
% Rotation Matrix about axis u in Cartesian.
Ru = ...
    [[R11, R12, R13];...
    [R21, R22, R23];...
    [R31, R32, R33]];
end
```

```
%% InterpMaps
% This function takes in multiple heatmaps and linearly interpolates them
% so that they can be superimposed.
function HeatMap = InterpMaps(HeatMaps)

[rws, cls, ~, segs] = size(HeatMaps);       % Find the dimesnions of the intensity distributions
qZ = zeros(rws, segs*cls);                  % Create a new Z matrix to contain the new superimposed heatmap data
qX = zeros(rws, segs*cls);                  % Create a new X matrix to contain the new superimposed heatmap data
qY = zeros(rws, segs*cls);                  % Create a new Y matrix to contain the new superimposed heatmap data

for i = 1:segs                              % Iterate over each map and fill the matrices with the coordinates
    qZ(:,((i-1)*cls+1):i*cls) = ...
        HeatMaps(:,:,1,i);

    qX(:,((i-1)*cls+1):i*cls) = ...
        HeatMaps(:,:,2,i);

    qY(:,((i-1)*cls+1):i*cls) = ...
        HeatMaps(:,:,3,i);
end

HeatMap = zeros(rws, segs*cls, 4);          % Create the matrix for the entire combined final heatmap
HeatMap(:,:,1) = qZ;                        % Add the coordinates previously achieved into it
HeatMap(:,:,2) = qX;
HeatMap(:,:,3) = qY;

for i = 1:segs                              % Iterate over every heatmap
    Zs = HeatMaps(:,:,1,i);
    Ys = HeatMaps(:,:,3,i);
    In = HeatMaps(:,:,4,i);
    IntIn = interpn(Zs',Ys',In',...         % Given the data from the current heatmap, linearly interpolate
        qZ, qY, 'linear', 0);               %  the power level at the rest of the queried points.
    HeatMap(:,:,4) = ...                    % Add the values to the final cumulative heatmap so that they superimpose
        HeatMap(:,:,4) + IntIn;
end

end
```

```
%% Vectorize
% This code takes in a heatmap composed of 4 meshgrids. 1 for Z values, 1
% for X values, 1 for Y values, and one for intensity values, and turns it
% into 4 column vectors of the ZXY-Intenisty data
function TotIntVecs = Vectorize(TotInt)
[rws, cls, ~] = size(TotInt);
TotIntVecs = zeros(rws*cls,4);
for i = 1:cls
    TotIntVecs((rws*(i-1)+1):(rws*i),:) = TotInt(:,i,:);
end

end
```

1	LatticeNa	H7BA-TwoSector-nux95-nuy36-3PW-Version6																
2																		
3	sStart	sEnd	ElementName	ElementType	Length	X	Z	theta	betax	betay	etax	etaxp	alphax	alphay	psix	psiy		
4	2.90809	2.908094	S01A:P0	MONI	0	0	2.908094	0	8.18504	5.902266	0.001108	-4.7E-15	-0.41711	-1.18748	0.395167	0.870895		
5	2.90809	3.146161	S01A:Q1	KQUAD	0.238067	0	3.146161	0	6.805503	7.861261	0.000997	-0.00092	5.812183	-7.59355	0.425995	0.907054		
6	3.14616	3.146161	S01A:P1	MONI	0	0	3.146161	0	6.805503	7.861261	0.000997	-0.00092	5.812183	-7.59355	0.425995	0.907054		
7	3.4731	3.711164	S01A:Q2	KQUAD	0.238067	0	3.711164	0	2.261581	16.27408	0.000528	-0.00052	1.558697	-0.50578	0.578886	0.954221		
8	3.76117	3.914261	S01A:M1.1	CSBEND	0.153095	-0.00037	3.91426	-0.00487	1.690956	16.4827	0.000796	0.004349	1.250975	-0.52146	0.682928	0.966622		
9	3.91426	4.117324	S01A:M1.2	CSBEND	0.203063	-0.00178	4.117318	-0.00902	1.245426	16.69766	0.002101	0.008508	0.943062	-0.53713	0.823317	0.978862		
10	4.11732	4.515446	S01A:M1.5-VP	VERTEX-POIN	0.398122	0	4.515446	0	0	0	0	0	0	0	0	0		
11	4.51545	4.72935	S01A:M1.3	CSBEND	0.213904	-0.00957	4.729293	-0.01641	0.659281	17.38404	0.009567	0.015891	0.014633	-0.58435	1.564846	1.014793		
12	4.72935	5.473428	S01A:M1.4	CSBEND	0.744078	-0.02381	5.473234	-0.02189	1.477437	18.29637	0.02343	0.021372	-1.11418	-0.64177	2.418858	1.056526		
13	5.47343	5.861645	S01A:M1.5	CSBEND	0.388217	-0.03279	5.861347	-0.02436	2.571155	18.8063	0.032206	0.023839	-1.7031	-0.67173	2.619383	1.077457		
14	5.95321	6.191277	S01A:Q3	KQUAD	0.238067	-0.04082	6.190881	-0.02436	4.325472	16.91169	0.042408	0.044243	-4.41727	8.792099	2.72171	1.095334		
15	6.31655	6.316546	S01A:P2	MONI	0	-0.04387	6.316113	-0.02436	5.506585	14.78159	0.04795	0.044243	-5.01132	8.212102	2.747381	1.103257		
16	6.31655	6.444672	S01A:S1	KSEXT	0.128126	-0.04699	6.444201	-0.02436	6.868594	12.75323	0.053619	0.044243	-5.61892	7.61888	2.768216	1.112589		
17	6.44467	6.572798	S01A:S1	KSEXT	0.128126	-0.05011	6.572289	-0.02436	8.386302	10.87688	0.059287	0.044243	-6.22652	7.025658	2.785098	1.123468		
18	6.79961	7.03768	S01A:Q4	KQUAD	0.238067	-0.06143	7.037033	-0.02436	12.82244	6.532756	0.073343	-0.01098	1.909175	0.235697	2.827298	1.181992		
19	7.03768	7.03768	S01A:P3	MONI	0	-0.06143	7.037033	-0.02436	12.82244	6.532756	0.073343	-0.01098	1.909175	0.235697	2.827298	1.181992		
20	7.24281	7.370934	S01A:S2	KSEXT	0.128126	-0.06955	7.370188	-0.02436	11.59019	6.393606	0.069683	-0.01098	1.788453	0.18185	2.854638	1.23358		
21	7.37093	7.49906	S01A:S2	KSEXT	0.128126	-0.07267	7.498276	-0.02436	11.13784	6.349659	0.068277	-0.01098	1.742039	0.161148	2.865915	1.25369		
22	7.70029	7.938353	S01A:Q5	KQUAD	0.238067	-0.08337	7.937439	-0.02436	8.755374	6.85871	0.060351	-0.03666	5.225784	-2.58757	2.90906	1.322365		
23	8.18196	8.310083	S01A:S3	KSEXT	0.128126	-0.09242	8.309059	-0.02436	5.317002	8.937505	0.046723	-0.03666	4.023866	-3.00465	2.96357	1.369861		
24	8.31008	8.438209	S01A:S3	KSEXT	0.128126	-0.09554	8.437147	-0.02436	4.338957	9.725872	0.042026	-0.03666	3.609596	-3.14841	2.990248	1.383604		
25	8.43821	8.438209	S01A:P4	MONI	0	-0.09554	8.437147	-0.02436	4.338957	9.725872	0.042026	-0.03666	3.609596	-3.14841	2.990248	1.383604		
26	8.56576	8.803826	S01A:Q6	KQUAD	0.238067	-0.10444	8.802655	-0.02436	2.499148	10.63981	0.031037	-0.01699	1.070752	2.920975	3.105929	1.418158		
27	8.85546	9.241316	S01A:M2.1	CSBEND	0.38586	-0.11553	9.240005	-0.02657	1.726643	8.255481	0.024031	-0.01477	0.695014	2.529032	3.318124	1.464854		
28	9.24132	9.586598	S01A:M2.2	CSBEND	0.345282	-0.1251	9.585154	-0.02889	1.349083	6.615831	0.019329	-0.01246	0.398466	2.219698	3.546331	1.511592		
29	9.5866	10.12327	S01A:M2.5-VP	VERTEX-POIN	0.536675	-0.13658	10.12171	-0.02436	0	0	0	0	0	0	0	0		
30	10.1233	10.14385	S01A:M2.3	CSBEND	0.020579	-0.14286	10.14212	-0.03487	1.171676	4.420162	0.014053	-0.00648	-0.08011	1.720459	4.005528	1.614824		
31	10.1439	10.46508	S01A:M2.4	CSBEND	0.321223	-0.15475	10.46313	-0.03911	1.311751	3.407301	0.012654	-0.00224	-0.35595	1.432678	4.267624	1.697691		
32	10.4651	10.97262	S01A:M2.5	CSBEND	0.507549	-0.17664	10.9702	-0.04718	1.894239	2.18378	0.013564	0.005823	-0.79167	0.977971	4.595445	1.884848		
33	11.0423	11.48034	S01A:Q7	KQUAD	0.438027	-0.20058	11.47736	-0.04718	1.501723	2.797982	0.011732	-0.01545	1.732524	-2.99043	4.857547	2.123586		
34	11.4803	11.48034	S01A:P5	MONI	0	-0.20058	11.47736	-0.04718	1.501723	2.797982	0.011732	-0.01545	1.732524	-2.99043	4.857547	2.123586		
35	11.7011	12.09097	S01A:M3.1	CSBEND	0.389898	-0.23147	12.0872	-0.05795	0.54311	5.767935	0.005581	0.000994	-0.22118	0.432877	5.69761	2.261343		
36	12.091	12.09098	S01A:M3.2-VP	VERTEX-POIN	1.51E-05	-0.22938	12.08732	-0.04718	0	0	0	0	0	0	0	0		
37	12.091	12.48087	S01A:M3.2	CSBEND	0.389883	-0.25615	12.47631	-0.06872	1.296536	3.760911	0.009141	0.017792	-1.92816	4.109292	6.211744	2.340412		
38	12.6067	13.19867	S01A:Q8	KQUAD	0.591958	-0.30544	13.19242	-0.06872	1.820331	2.336548	0.012352	-0.01491	2.401284	-2.38085	6.547692	2.674102		

Figure 10: Lattice file rows containing Vertex-Points

1	LatticeName	H7BA-TwoSector-nux95-nuy36-3PW-Version6																	
2																			
3	sStart	sEnd	ElementNam	ElementType	Length	X	Z	theta	betax	betay	etax	etaxp	alphax	alphay	psix	psiy			
4	2.9080942	2.9080942	S01A:P0	MONI	0	0	2.9080942	0	8.18503956	5.90226637	0.00110783	-4.709E-15	-0.4171074	-1.18748	0.39516656	0.87089516			
5	2.9080942	3.14616145	S01A:Q1	KQUAD	0.23806725	0	3.14616145	0	6.8055034	7.86126064	0.0009967	-0.0009177	5.81218272	-7.593555	0.42599522	0.90705371			
6	3.14616145	3.14616145	S01A:P1	MONI	0	0	3.14616145	0	6.8055034	7.86126064	0.0009967	-0.0009177	5.81218272	-7.593555	0.42599522	0.90705371			
7	3.47309715	3.7111644	S01A:Q2	KQUAD	0.23806725	0	3.7111644	0	2.26158079	16.2740752	0.00052816	-0.0005166	1.55869715	-0.5057843	0.57888572	0.95422113			
8	3.76116559	3.91426083	S01A:M1.1	CSBEND	0.15309524	-0.0003725	3.91426023	-0.0048657	1.69095617	16.4827041	0.00079568	0.00434895	1.25097459	-0.5214566	0.68292834	0.96662197			
9	3.91426083	4.11732363	S01A:M1.2	CSBEND	0.2030628	-0.0017827	4.11731798	-0.0090246	1.24542643	16.697663	0.00210104	0.00850773	0.94306182	-0.5371263	0.82331681	0.97886247			
10	4.11732363	4.72934999	S01A:M1.3	CSBEND	0.61202636	-0.0095652	4.72929346	-0.0164081	0.65928112	17.3840388	0.00956733	0.01589076	0.0146333	-0.5843544	1.56484648	1.01479273			
11	4.72934999	5.47342764	S01A:M1.4	CSBEND	0.74407766	-0.0238126	5.47323378	-0.0218896	1.47743723	18.2963724	0.02343041	0.02137162	-1.1141797	-0.6417725	2.41885773	1.05652639			
12	5.47342764	5.86164466	S01A:M1.5	CSBEND	0.38821702	-0.0327887	5.86134691	-0.024357	2.57115476	18.8062964	0.03220611	0.02383862	-1.7030984	-0.67173	2.61938338	1.07745653			
13	5.95320934	6.19127659	S01A:Q3	KQUAD	0.23806725	-0.0408167	6.19088107	-0.024357	4.32547216	16.9116909	0.04240762	0.04424288	-4.4172664	8.79209872	2.72171027	1.09533361			
14	6.31654598	6.31654598	S01A:P2	MONI	0	-0.0438676	6.31611133	-0.024357	5.50658521	14.7815851	0.0479499	0.04424288	-5.0113185	8.21210242	2.7473808	1.10325672			
15	6.31654598	6.44467193	S01A:S1	KSEXT	0.12812596	-0.0469881	6.44420125	-0.024357	6.86859432	12.7532253	0.05361856	0.04424288	-5.6189171	7.61888021	2.76821579	1.11258868			
16	6.44467193	6.57279789	S01A:S1	KSEXT	0.12812596	-0.0501085	6.5722892	-0.024357	8.38630173	10.8768798	0.05928722	0.04424288	-6.2265156	7.025658	2.78509836	1.12346754			
17	6.79961234	7.03767959	S01A:Q4	KQUAD	0.23806725	-0.0614306	7.03703301	-0.024357	12.8224377	6.53275571	0.07334253	-0.0109801	1.90917469	0.23569709	2.82729782	1.18199175			
18	7.03767959	7.03767959	S01A:P3	MONI	0	-0.0614306	7.03703301	-0.024357	12.8224377	6.53275571	0.07334253	-0.0109801	1.90917469	0.23569709	2.82729782	1.18199175			
19	7.24280783	7.37093379	S01A:S2	KSEXT	0.12812596	-0.0695468	7.37018836	-0.024357	11.5901878	6.39360627	0.06968335	-0.0109801	1.78845284	0.18185036	2.85463786	1.23357957			
20	7.37093379	7.49905975	S01A:S2	KSEXT	0.12812596	-0.0726673	7.49827632	-0.024357	11.1378402	6.34965928	0.06827651	-0.0109801	1.74203901	0.16114796	2.86591504	1.25368985			
21	7.70028596	7.93835322	S01A:Q5	KQUAD	0.23806725	-0.0833661	7.93743948	-0.024357	8.7553739	6.85870983	0.0603506	-0.0366588	5.22578362	-2.5875682	2.9090604	1.32236479			
22	8.1819572	8.31008316	S01A:S3	KSEXT	0.12812596	-0.0924195	8.30905916	-0.024357	5.31700202	8.93750524	0.04672342	-0.0366588	4.02386649	-3.0046512	2.96356986	1.36986127			
23	8.31008316	8.43820912	S01A:S3	KSEXT	0.12812596	-0.0955399	8.43714711	-0.024357	4.33895734	9.72587201	0.04202647	-0.0366588	3.60959589	-3.1484092	2.99024843	1.38360417			
24	8.43820912	8.43820912	S01A:P4	MONI	0	-0.0955399	8.43714711	-0.024357	4.33895734	9.72587201	0.04202647	-0.0366588	3.60959589	-3.1484092	2.99024843	1.38360417			
25	8.5657584	8.80382565	S01A:Q6	KQUAD	0.23806725	-0.1044444	8.8026552	-0.024357	2.49914821	10.6398067	0.03103714	-0.0169917	1.07075151	2.92097543	3.10592942	1.41815757			
26	8.85545558	9.24131605	S01A:M2.1	CSBEND	0.38586047	-0.115527	9.24000509	-0.0265747	1.7266427	8.25548073	0.0240312	-0.0147745	0.69501415	2.52903212	3.31812429	1.4648545			
27	9.24131605	9.58659781	S01A:M2.2	CSBEND	0.34528176	-0.1251012	9.58515401	-0.0288895	1.34908334	6.61583096	0.01932943	-0.0124599	0.3984658	2.21969761	3.54633118	1.51159231			
28	9.58659781	10.143852	S01A:M2.3	CSBEND	0.55725421	-0.1428646	10.1421242	-0.0348746	1.17167599	4.42016205	0.01405336	-0.006476	-0.0801098	1.72045907	4.00552818	1.61482416			
29	10.143852	10.4650752	S01A:M2.4	CSBEND	0.3212232	-0.1547454	10.4631274	-0.0391147	1.31175078	3.40730115	0.01265403	-0.0022366	-0.3559545	1.43267839	4.26762385	1.69769079			
30	10.4650752	10.9726238	S01A:M2.5	CSBEND	0.50754859	-0.1766367	10.9702023	-0.0471754	1.89423892	2.18377963	0.01356406	0.00582256	-0.7916707	0.97797063	4.59544474	1.8848479			
31	11.0423151	11.4803424	S01A:Q7	KQUAD	0.43802727	-0.2005797	11.477356	-0.0471754	1.50172293	2.79798174	0.01173191	-0.0154522	1.73252382	-2.9904345	4.85754739	2.12358611			
32	11.4803424	11.4803424	S01A:P5	MONI	0	-0.2005797	11.477356	-0.0471754	1.50172293	2.79798174	0.01173191	-0.0154522	1.73252382	-2.9904345	4.85754739	2.12358611			
33	11.7010705	12.0909685	S01A:M3.1	CSBEND	0.38989805	-0.2314731	12.0871962	-0.0579489	0.54311016	5.7679348	0.00558076	0.00099377	-0.2211801	0.43287706	5.69760987	2.26134254			
34	12.0909685	12.4808666	S01A:M3.2	CSBEND	0.38989805	-0.2561509	12.4763106	-0.0687223	1.29653648	3.76091108	0.00914139	0.01779203	-1.9281643	4.10929224	6.21174363	2.34041195			
35	12.606713	13.198671	S01A:Q8	KQUAD	0.59195802	-0.3054413	13.1924206	-0.0687223	1.82033079	2.33654797	0.01235187	-0.0149113	2.40128386	-2.3808543	6.5476918	2.67410221			
36	13.198671	13.198671	S01A:P6	MONI	0	-0.3054413	13.1924206	-0.0687223	1.82033079	2.33654797	0.01235187	-0.0149113	2.40128386	-2.3808543	6.5476918	2.67410221			
37	13.474736	13.7997371	S01A:M4.1	CSBEND	0.32500113	-0.3483067	13.7919532	-0.0785398	0.3646131	4.99346378	0.00586137	8.0821E-15	-4.374E-14	-9.437E-15	7.47106797	2.83702517			
38	13.7997371	14.1247382	S01A:M4.2	CSBEND	0.32500113	-0.375396	14.1158221	-0.0883573	0.77778929	3.86859637	0.00823539	0.0149113	-1.3751511	3.16873891	8.16030141	2.9079965			

Figure 11: Lattice file rows with removed Vertex-Points

6.2 Help

- Don't know how to run the program? Simply specify a point on the absorption plane, a normal unit vector, and a vector containing which rows in the lattice file the analysis should be done on.

```
>>
                              % Note: inputs are column vectors (spaced with ';' not ',')
p0 = [18.7079; -0.8732; 0]; % [global Z value; global X value; global Y value] all in meters
np = [0.9912; -0.1323; 0];  % [global Z value; global X value; global Y value] all in meters
latRows = [46;48;51;52];    % row numbers in the lattice file corresponding to each magnet

[TotIntVecs, TotalPower, Peak] = CalcPower(p0, np, latRows);
```

- Always input SI units.
- Are there gaps in the ray trace fans? This means the path is not continuous. Either there is an error in the lattice file values; the number of each row that is input is incorrect (double check the row numbers!), the wrong lattice file is being used (See the 'CalcPower' script above - lines 10 and 11. Whatever is uncommented, it better be the lattice file being used), or finally perhaps the offset (see the script for 'CalcPower' line 17) is wrong. As shown in Figures 10 and 11, the first 3 rows are non-numeric so they are not included in the matrix named 'Lattice' in the code. 3 is thus subtracted from the input values to correct for this. If there is a numerical value in one or more of the first few lines then the number currently being subtracted (3) must be changed.
- Want a different mesh size in the transverse direction? Change the value 'n' in line 13 of the script for 'CalcPower'. Note that this mesh is constant size for the particle's trajectory but not for the heatmap. On the heatmap it is more dense where the gradient is higher (often desirable as it is).
- Want a different mesh size in the vertical direction? Change the value 'yN' in line 14 of the script for 'CalcPower'. This is a constant sized mesh on the absorbing surface's vertical axis.
- Want to cover a greater vertical distance on the absorber? Change the value 'm' in line 15 of the script for 'CalcPower'. Currently set at 2 [mm], this is the vertical range that the heatmap covers. 'yN' is the number of points examined within this range.
- If the full trajectory is desired, simply change the output to include the 'totR' matrix defined on line 19 and updated on line 62 of 'CalcPower'.
- Lines 64 through 67 of 'CalcPower' calculate the missteered paths. If one examines the specific script 'Missteer' she or he will see that it is possible for the user to specify specific missteered values rather than just use the extreme errors. The script 'CalcPower' will have to be edited to reflect this input and to output the desired missteered paths.
- If one creates a more stable working interpolation function it would replace line 86 of 'CalcPower'. The format of the input, 'HeatMaps' currently in the code is this: it's an 'yN by n by 4 by seg' matrix (4-dimensional). If 'seg' is the number of rows/bend magnets being analyzed then imagine 'seg' 3-dimensional matrices all contained within the 4-dimensional matrix, 'HeatMaps'. As it appears, each of these 3D matrices are a separate heatmap corresponding to each segment (element, bend magnet, etc...). The structure of the individual heatmap is that they have a depth of 4 (or visualized as 4 layers). The first is an [yN by n] matrix of all of the Z values on the absorbing plane, then another of all the X values, then all of the Y values, and finally all of the intensity values. The third layer containing the Y values will have 'n' identical columns each with yN rows that span from '-m' to '+m'. Each column represents the vertical spread found from each zero-vertical-angle photon that was initially traced out onto the absorber.

- Notice that in addition to the 'Missteer' script, the 'RayTrace' script also has default values in the case that a normal vector and point were not specified. This doesn't work with the code as it is but just goes to show that these scripts are autonomous and can be used independently in other contexts.
- Lines 47 through 64 of the 'RayTrace' script are used to determine if the ray impacts the planar surface and if so to plot it. If a code that handled arbitrary shapes were designed this is where it would go.
- Figure 10 shows a lattice file that contains Vertex-Points. As evident in Row 30, there are sometimes mistakes in the lattice files. The length of the M2.3 magnet here is certainly not 0.02 meters. This messes up the B-field calculation and thus the heat load is incorrect. Figure 11 shows a lattice file with the Vertex-Points removed. Line 28 is the same M2.3 magnet but with a more realistic length of 0.557 meters. It's good to check which lattice file is being used and if the values being input into the code through the lattice files make physical sense.
- The best way to debug or understand Matlab code is to just play around with it. I suggest using the command window and just playing with the individual functions, checking the contents and sizes of different variables by unsuppressing them in the code itself. This would be especially beneficial to someone who plans to improve the project to include things like arbitrary absorbers, quadrupoles, and more.