CHARGED FUSION PRODUCT TRAJECTORY SIMULATIONS

Omar Leon[1]

[1]*Florida International University Department of Physics*

Spherical tokamaks use magnetic fields to produce and confine plasmas. Due to the magnetic properties of plasmas and the magnetic fields of tokamaks, current diagnostics focus on detecting neutral particles. When the temperature of plasmas, composed of hydrogen isotopes, are sufficiently energetic, the isotopes react with one another emitting protons and other charged particles. These reactions are called fusion reactions. A new diagnostic, the charged fusion product diagnostic is geared towards detecting protons in order to expand the methods of studying plasmas. However, the complex nature of plasmas and the magnetic fields makes calculating many plasma properties analytically impossible. In conjunction with the charged fusion product diagnostic, a program called the Orbit Code was developed and tailored to the requirements of the diagnostic in order to determine the trajectories of protons emitted from fusion reactions inside in the Mega Amp Spherical Tokamak (MAST) located at the Culham Centre for Fusion Energy (CCFE). The approximations used in the Orbit Code for its calculations had to be reviewed for accuracy and correctness. Using these predictions, the diagnostic can be placed in the locations that most effectively samples the plasmas and compare the experimental results with the theoretical models created through the Orbit Code.

CONTENTS

I. INTRODUCTION

A. Background

Plasmas are produced by heating a gas until a large portion of the atoms have been ionized. [**?** , pg.2-3] Their ionized property allows them to be altered by magnetic fields; machines, called tokamaks (figure 1), use this fact to produce and confine plasmas.

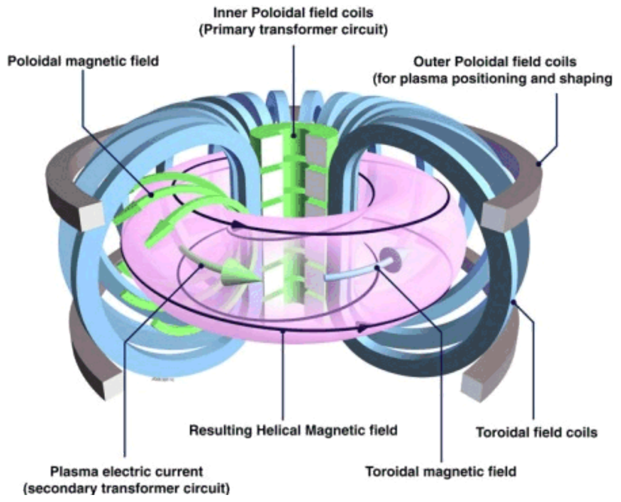

Inner Poloidal field coils
(Primary transformer circuit)
Poloidal magnetic field
Outer Poloidal field coils
(for plasma positioning and shaping
Resulting Helical Magnetic field
Toroidal field coils
Plasma electric current
(secondary transformer circuit)
Toroidal magnetic field

FIG. 1: General Setup of Tokamak [**?**]

The process by which tokamaks create and maintain plasmas includes several steps. First, gas is introduced into the tokamak and a current is run through the primary transformer circuit, located in the center of the tokamak, to induce a current(carried by a few pre-existing ions) through the gas, which acts as the secondary coil of the transformer. As the gas gains temperature, electrons are removed from the atoms, the gas becomes more and more ionized, and the gas transitions into a plasma state. This method of heating, by driving current through the plasma, is called ohmic heating.[**?**]

To achieve fusion within the plasma, the gas is composed of hydrogen isotopes, either deuterium, tritium, or a mixture of both, and plasma temperatures need to exceed 100 million degrees Celsius. To achieve these temperatures, two methods are employed, either simultaneously or individually: radio frequency heating and neutral beam injection. In radio frequency heating, currents of high frequency oscillations are induced in sections of the plasma that resonate with the current frequency. In doing so, the particles in the induced current gain kinetic energy, thus increasing the temperature in the region. In neutral beam injection, neutral deuterium atoms bombard the plasma and the kinetic energy of the collisions increases the temperature.[?]

Due to the strong magnetic fields, diagnostic systems, instruments used to measure physical parameters, traditionally rely on detecting neutral particles as their trajectories are not altered by the magnetic fields of the tokamak. A charged fusion product diagnostic (CFPD), shown below, was developed to detect charged reaction products in order to determine the emissivity profile of plasmas, a distribution function representing the density of fusion reactions. The full diagnostic is show in the figures below.

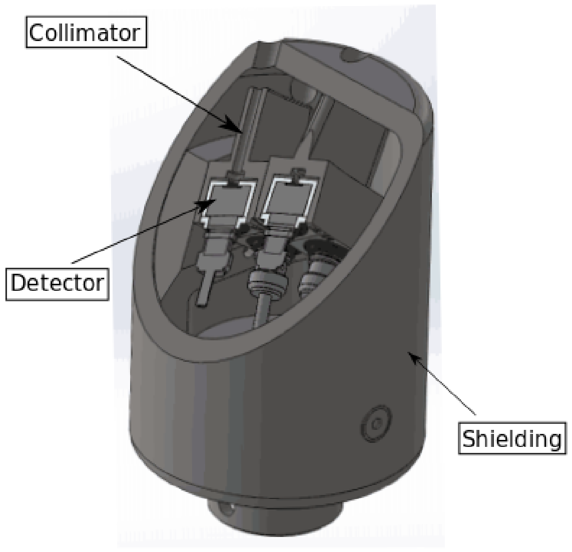


FIG. 2: Sliced view of Fully Constructed CFPD[? , pg.1]

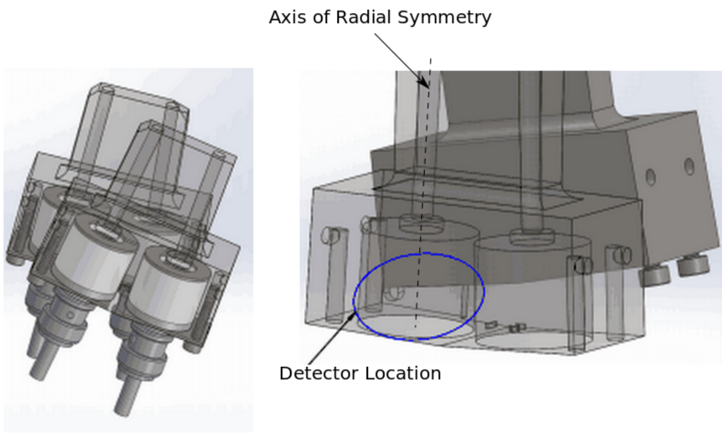


FIG. 3: The left side of the image is the collimator & detector Setup. The right side of the image is a transparent view of same setup[? , pg.1]

Figure 2 shows the CFPD diagnostic, where it is shown that each detector varies in distance relative to the top of the shielding. Figure 3, shows a more detailed view of the detectors in conjunction with their respective collimators, mechanical structures that filter incoming particles . To test this diagnostic, deuterium plasmas were studied heated with neutral beam injections. This fusion reaction results in one of the two following reactions:

$$^2H +^2 H \rightarrow^1 H +^3 H + 3.27MeV$$

FIG. 4: Deuterium (2H) and Deuterium react to create a proton (1H) and a triton (3H) with a release of energy , given in MeV.[**?** , pg. 18]

$$^2H +^2 H \rightarrow^3 He +^1 n + 4.03MeV$$

FIG. 5: Deuterium (2H) and Deuterium react to create a neutron (n) and helium-3 ions (3He) with a release of energy, given in MeV.[**?** , pg. 18]

The proton of the above reactions will be detected by the CFPD while the neutron will be detected by a neutron camera. The data collected from both systems will be compared to ensure the CFPD is functioning properly. This can be done under the assumption that a similar number of protons and neutrons are emitted. As the emitted charged particles move through the tokamak they experience a force due to the magnetic field. A new convention of angles was defined to describe where the detectors are oriented with respect to the center of the tokamak (shown in figure 6).

The center of the tokamak is the origin in figure 6, the ϕ angle is defined as the angle between the detector and the x-z plane, and the θ angle is the angle between a projection of the detector on the x-z plane and the z-axis.

To alter the CFPD's distance from the center of the tokamak and to rotate it, the CFPD is mounted on a reciprocating probe arm (see figure 7) that is controlled remotely and is taken to be along the x axis of figure 6 for the purposes of determining the orientation of the detectors.

B. The Orbit Code

The complicated magnetic fields inside a tokamak plasma require a numerical solution of the equation of motion for the charged particles. A program, call the Orbit Code, was modified to

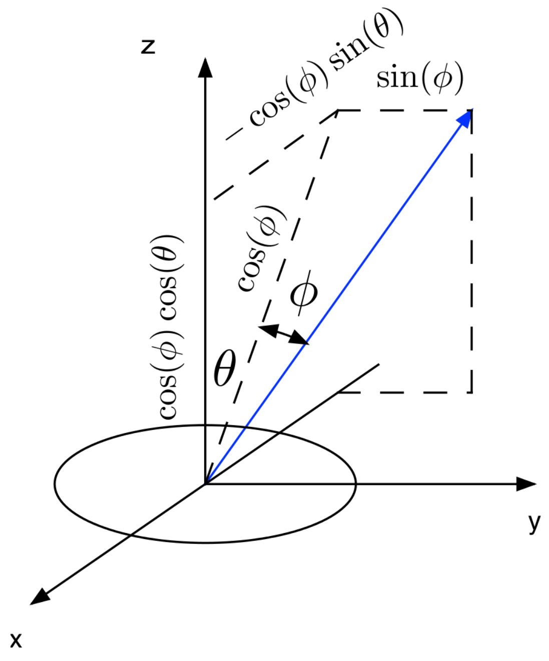


FIG. 6: Angles Defined for Detector Orientation

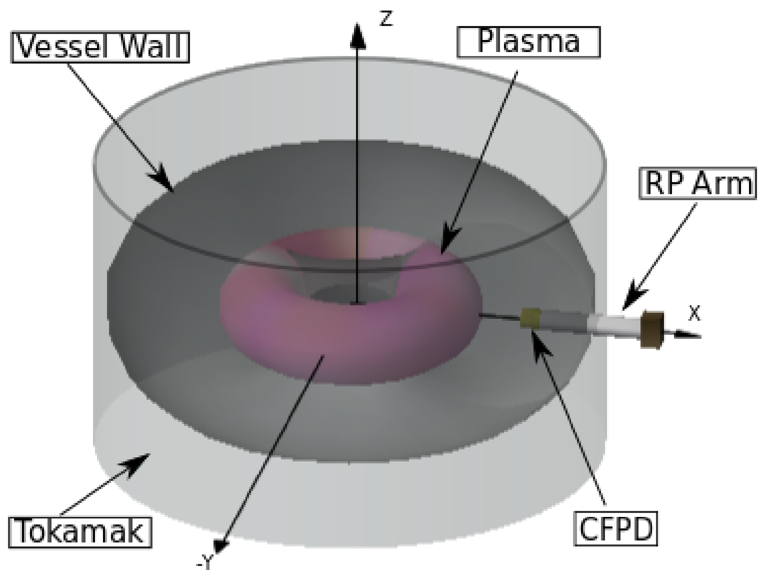


FIG. 7: Rendering of Detector Setup Within Tokamak (not to scale)

calculate a proton trajectories and the charged particle emissivity of the plasma along these trajectories using MAST's equilibrium magnetic field configuration and physical limitations. For the CFPD, the Orbit Code needed to be adapted to more accurately approximate the experimental setup. The new version can determine which portions of the plasma was sampled by each detector. This made the Orbit Code an invaluable part of the experiment when deciding how to orient the diagnostic between each experimental run for the most effective data collection.

The code is structured into three different portions: the input section, the Fortran portion, and the Python portion. Each section had its own alterations to meet the experiment's demands. The original graphical output of the Orbit Code is shown in figure 8.

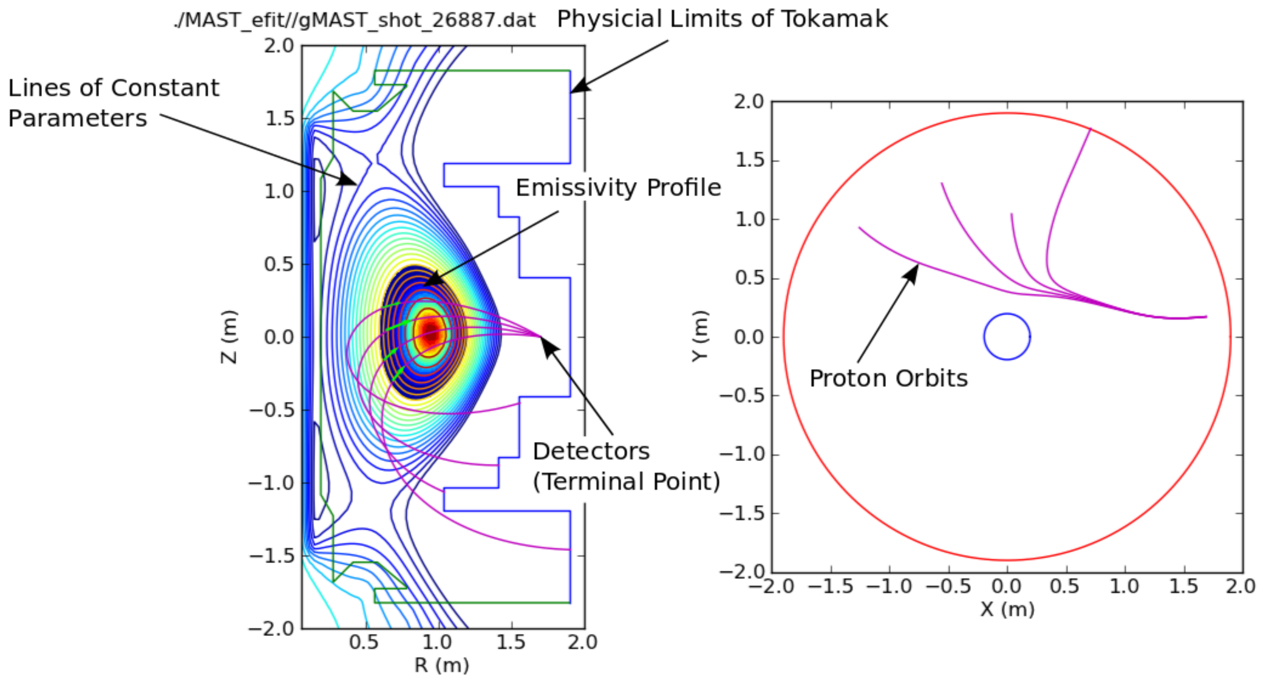


FIG. 8: Original Output Graph of the Orbit Code

In figure 8, the leftmost graph is a vertically sliced view of the tokamak and plasma presenting the lines of constant parameter, such as pressure and magnetic flux; the colored region describes the emissivity profile where the most dense region is deep red; and the purple lines demarcate the proton trajectories, or orbits. Protons that are emitted from the emissivity profile, tangential to the orbits, towards the outside of the tokamak will be detected as they terminate on the same point. The rightmost graph is a top of the horizontal mid-plane of the tokamak and proton trajectories. There were several factors in the code that limited how well it could approximate the physical experimental setup. Chief among these factors was the inability to individualize each detector orientation, calculate the effects of rotation on the CFPD.

Furthermore, the CFPD utilizes a cylindrical collimator, which acts as the mechanical filter for the detector. The Orbit Code has a square collimator; as a result, it is necessary to calculate the deviation between the solid angle of acceptance between a square collimator and a cylindrical collimator. Using a square collimator, the Orbit Code is capable of segmenting the collimator entrance into equal parts along both sides of the square. This function allows for the determination of the amount of area a detector can probe; the more segmented the collimator entrance the better this scope is defined. Figure 9 gives a pictorial representation of the result of segmenting a single detector.

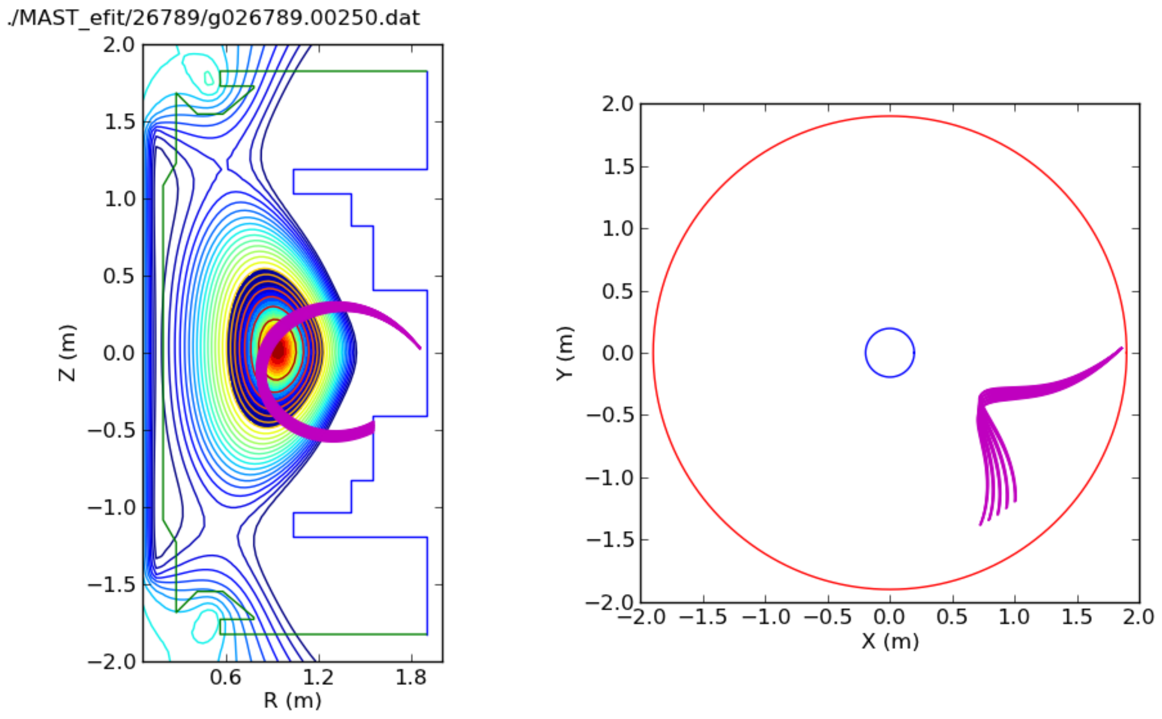


FIG. 9: Single Detector Probing Fusion Region

The colored region of lines that terminate at the same point describes the area probed by one, specific detector.

II. INPUT SECTION

The input section for the Fortran portion of the Orbit Code focuses on three main type of files. They are as follows:

1. **Efit file:**

   A data file that gives the magnetic field configuration of the plasma at an instant in time. It is used to model the plasma's overall shape and the trajectories of the particles emitted from fusion reactions.

2. **Limiter file:**

   A data file which provides the physical structure inside the tokamak, setting limits to many of the calculations performed by the Orbit Code.

3. **Name list (.nml) file:**

   A data file that contains a variety of parameters that are necessary for calculation, including the detector orientations and where the collimator entrances are located in the tokamak, and what efit and limiter files are used.

This .nml file needed to be altered to allow for a greater degree of flexibility for each detector. Prior to alterations, many parameters were determined by using a step function with an initial value. For example, the θ angles of the detectors were written in the following way:

```
theta_port = 60.

vary_theta_port = F

theta_port_stop = 83.

theta_port_step = 7.5
```

In this manner, the Fortran portion of the code would use the initial position given and then, if vary_theta_port equaled True, it would alter the initial value by the step amount until the stop limit was reached.

In order to grant the user the ability to choose which detector and how many to use in the analysis and the individual orientation, the .nml file was divided into two files. The first file is called Static_MAST.nml where the parameters that rarely change are stored. Examples of these parameters are the efit file and the model used to model the plasma's emission profile. The second file is called dynamic_input.data and it contains the information that changes more frequently such as the radial positions of the detector with respect to the center of the tokamak. This second file is used exclusively in the Python portion of the Orbit Code and includes all of the physical parameters of the detector system and the radial position of the collimator entrance closest to the plasma. The input format of the Static_MAST.nml file is identical to its unaltered counterpart; it is in the dynamic_input.data file where the format changes. When the two files are used in conjunction, the .nml file created and used by the Fortran portion has the following, altered format:

```
detectors = 4

detector_number(1)= 1

detector_number(2)= 2

detector_number(3)= 3

detector_number(4)= 4

theta_port(1)= 40.0

theta_port(2)= 40.0

theta_port(3)= 40.0

theta_port(4)= 40.0
```

This method gave the user of the program much greater flexibility when assigning values to each detector. In this new format, the detectors and detector_number allow the user to choose how many detectors and which detectors to use in the analysis. Additionally, every parameter that used to be calculated using the step method described above is given its own value independent of the other parameters.

III. FORTRAN SECTION

All of the computations for the plasma parameters including the particle trajectories, the magnetic flux, and the emissivity profile occur in this section of the Orbit Code.

A. Acquiring the Data

The .nml file is read by a file called rdpar_nml.f, short for read parameters name list, and the parameters in the .nml file are stored as several variables. In the original file, rdpar_nml.f used the step and stop inputs to create each detector orientation. This file was altered to execute the following code instead:

```
        do i = 1, detec
                ports(i) = theta_port(detector_number(i))*dtr
                portsph(i) = phi_port(detector_number(i))*dtr
                Al0s(i) = gyro_angle(detector_number(i))*dtr
                be0s(i) = pitch_angle(detector_number(i))*dtr
                rds(i) = rdist(detector_number(i))
                zds(i) = zdist(detector_number(i))
                phda(i) = phdangle(detector_number(i))
        end do
```

where dtr is the conversion value from degrees to radians. In this manner, each detector is assigned a unique set of variables.

B. Coordinating and Executing Fortran subroutines

A separate file, named orbit.f, organizes the data and calls a separate subroutine, named chrien, to execute all of the calculations to model the plasma, emissivity density, and particle trajectories. This process occurred through a series of conditional statements based on the original .nml file. Since these conditionals became obsolete after the .nml file was changed, the orbit.f file was altered to do the same function with a single *loop* as follows:

```
        do j= 1, detec
                port = ports(j)
                portph = portsph(j)
                RD = rds(j)
                be0 = be0s(j)
                al0 = al0s(j)
                zd = zds(j)
                phd = phda(j)
                call chrien
        end do
```

Through this single do loop, each time chrien is called, each detector has its own unique parameters.

IV. ORBIT CODE APPROXIMATIONS

The Orbit Code uses a square collimator entrance; however, the CFPD uses a cylindrical collimator. To ensure that the proper theoretical rates of emission are calculated, the solid angle of acceptance for a square and cylindrical collimator need to be similar. The code also has the capability of segmenting the collimator entrance into smaller, equal pieces. By segmenting the collimator entrance, it is possible to determine how much of the fusion region as single detector can probe. Figure 10 shows the range of the bundle, shown in figure 9, as a function of segments. The use of this segmentation needed to be justified by ensuring that the efficiency of the detector is not affected by the segmentation.

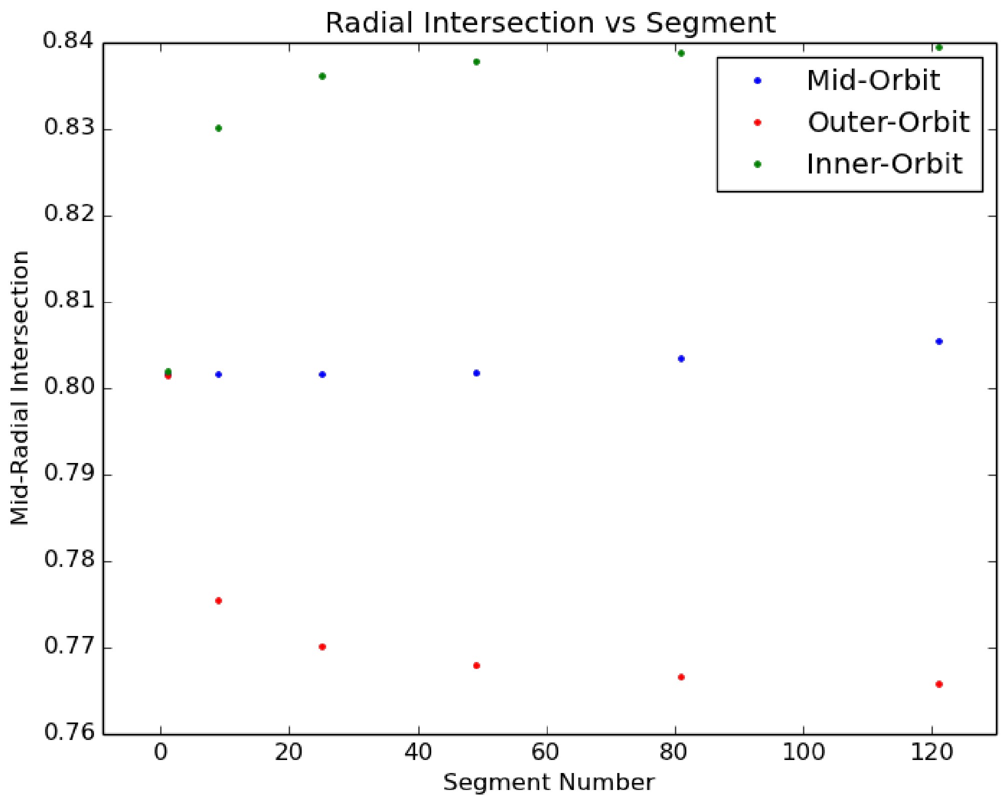


FIG. 10: The range of the bundle, the difference between the other and inner orbits remain constant after 49 segments.

A. Collimator Geometry

Regardless of the shape of the collimator, the amount of particles that enter through the collimator and reach the active area of the detector is dependent on the angle between the incoming particles and the collimator entrance, θ. Therefore, the amount of particles which enter the collimator is $A_{entrance} \cos\theta$ where $A_{entrance}$ is the area of the collimator entrance. The effective solid angle of acceptance of a collimator is given by equation 1, where T(θ) is the transmission coefficient which is determined by the shape of the collimator. Figure 11 depicts a cross sectional view of the relationship between the incoming particles and the collimator.

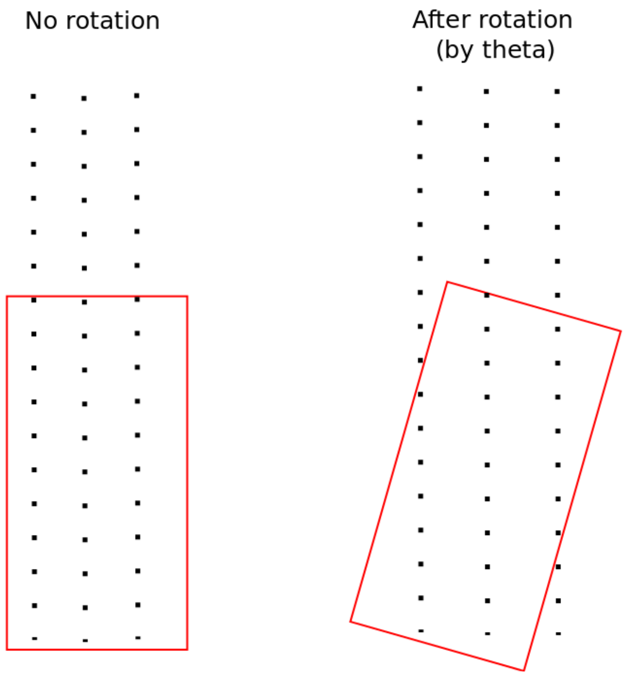
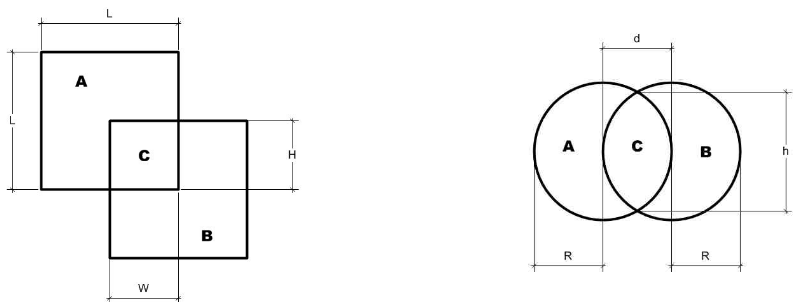


FIG. 11: The red rectangle is a cross sectional view of the collimator and the black dots are incoming particles.

$$SA_{effective} = \int_0^{\theta max} T(\theta) A_{entrance} \cos\theta \mathrm{d}\Omega \tag{1}$$

The transmission coefficient is the area of overlap between the area created by the incoming particles extended to the vertical height of the detector and the active area of the detector. The area of overlap based on the type of collimator is shown in figure 12.



(a) Area of Overlap Created by a Square Collimator

(b) Area of Overlap Created by a Circular Collimator

FIG. 12: Section A is the area of particles that have gone through the collimator entrance. Section B is the active area of the detector. Section C is the area of overlap between section A and B.

TABLE I: Effective Solid Angle of Acceptance ($Sr\dot{m}^2$)

Square Collimator	Circular Collimator	Deviation (%)
9.83×10^{-8}	9.82×10^{-8}	0.1

The results produced by equation 1 when using each collimator and the deviation between the two are shown in table 1.

B. Segmentation Efficiency

In theory, the total efficiency of the detectors should be the same regardless of the number of segmentations were applied to the collimator entrance. However, the numerical calculations performed by Fortran have slight deviations in the total efficiency of the detector as the number of segments increases (figure 13). By comparing the efficiencies as the number of segments increase, it is possible to determine whether or not the segmentation function of the code can be used for data analysis.

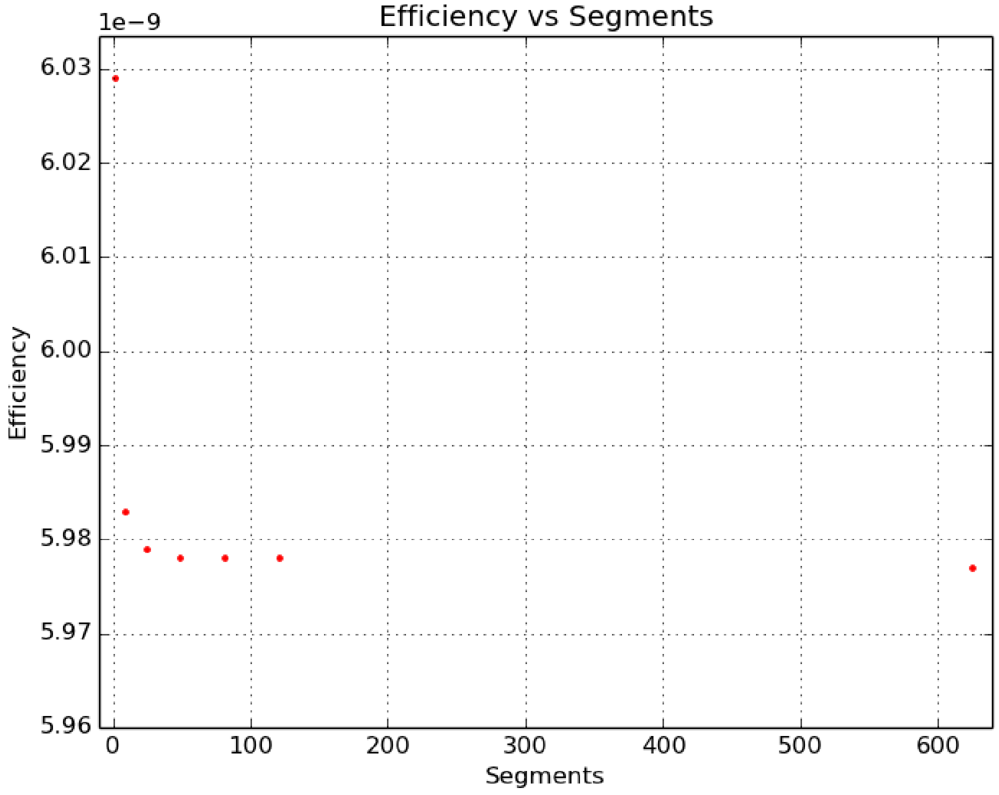


FIG. 13: The total efficiency of the detectors notably decreases from 1 segment to 25 segments but stays at the same relative level as the segments increase beyond 25 segments.

TABLE II: Total Efficiency vs Segments

Segments	Total Efficiency	Deviation of Total Efficiencies (%)
1	6.029×10^{-9}	0.
9	5.983×10^{-9}	0.76
25	5.979×10^{-9}	0.83
49	5.978×10^{-9}	0.85
81	5.978×10^{-9}	0.85
121	5.978×10^{-9}	0.85
625	5.977×10^{-9}	0.86

V. PYTHON SECTION

The Python section runs the Fortran executable, organizes the output files created by Fortran and generates a plot that visually represents the results. The original code ran using two Python scripts, run_orbit_nml.py and plot_orbits_combined.py. The former executed the Fortran executables and organized the output files created by these executables whereas the latter plotted the Fortran output files. A separate Python file, run_all.py, was created to accomplish several tasks. The file reads the dynamic_input.data and Static_MAST.nml, extracts all of the necessary information, and stores them as variables. These variables are then used to calculate the following (figure 14 will be used as a reference) :

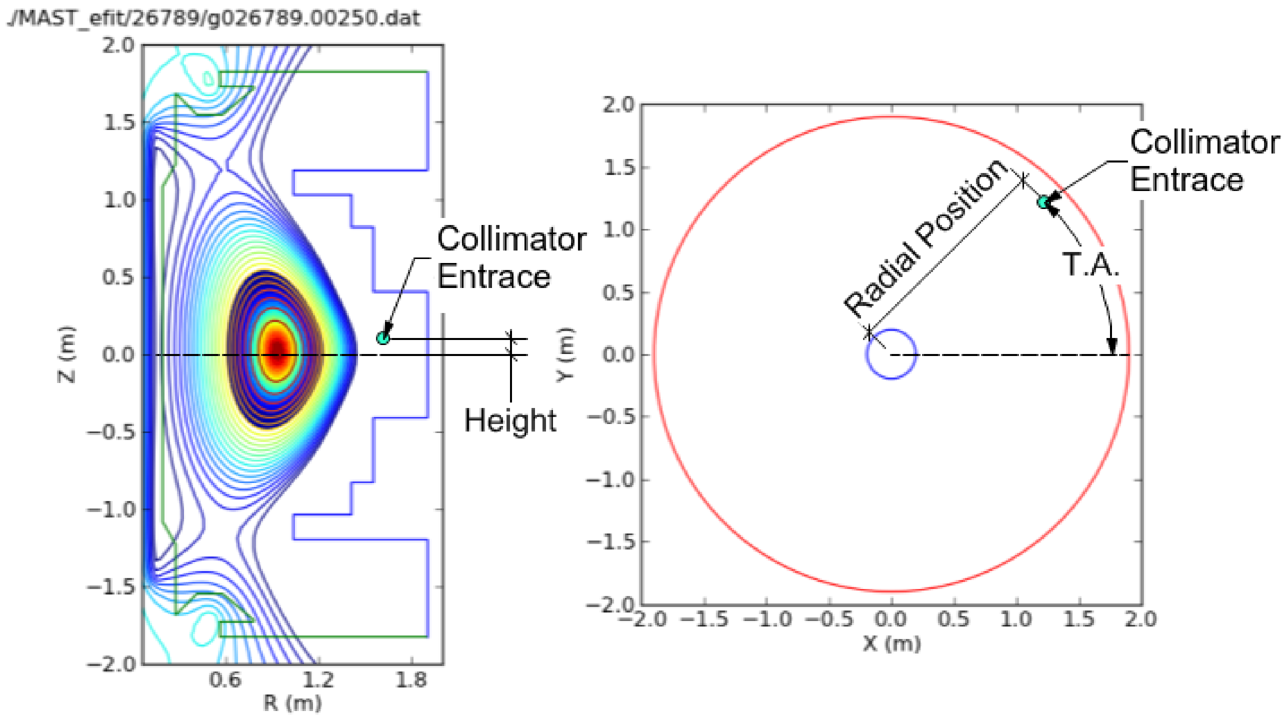


FIG. 14: Output Detailing the Detector Height, Toroidal Angle (T.A.), and Radial Distance

- The distance of each collimator entrance from the center of the tokamak, as shown in the right most portion of figure 14.

  The radial position is measured from the center of the tokamak to the position of collimator entrance that is closest to the plasma; this entrance is used as a reference for the other entrances.

$$RDist = RDist\_coll + \Delta coll\_pos * 1. * 10^{-3}$$

where RDist_coll is the position of the reference collimator and coll_pos is the distance from the reference collimator to each collimator in the system.

- The height of each collimator entrance relative to a central horizontal plane, as shown in the left most portion of figure 14.

  This is calculated through the following equation:

  $$Det\_height = RP\_height + coll\_height\_offset * 1. * 10^{-3}$$

  where RP_height is the height of the reciprocating probe arm and coll_height_offset is the height of each collimator entrance relative to the radial axis of the system.

- The toroidal angle (T.A.) of the collimator entrances , as shown in the right most portion of figure 14.

  This angle dictates where the collimator entrances are positioned within the tokamak.

  $$PHDangle = Probe\_angle + \arctan(\frac{coll\_horizontal\_offset*1.*10^{-3}}{RDist})$$

  where Probe_angle is the toroidal angle of the reciprocating probe and coll_horizontal_offset is the horizontal distance of each collimator entrance from the radial axis of the system.

- Update the θ and ϕ angles of the detectors if the reciprocating probe arm is rotated.

  The following steps show how this was calculated:

  1. Before the probe arm is rotated, the x,y, and z positions of each detector, denoted by a vector $\hat{r}$, is given by

  $$x = -1 * \cos(\phi) * \sin(\theta)$$

  $$y = \sin(\phi)$$

  $$z = \cos(\phi)\cos(\theta)$$

After a rotation by an angle, α, the new vector, $\hat{r}'$ is calculated using the rotation matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -1{*}\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}$ obtaining

$$x' = x$$

$$y' = y * \cos(\alpha) - z * \sin(\alpha)$$

$$z' = y * \sin(\alpha) + z * \cos(\alpha)$$

2. The angle between the vector normal to the x-z plane, $\hat{n}$, and $\hat{r}'$ is calculated using the definition of the dot product which produces

$$\cos(\beta) = \hat{r'} \cdot \hat{n}$$

Since the complimentary angle to β is the new phi angle, ϕ', β can be written as $90 - \phi'$. Therefore, $\cos(\beta)$ is equivalent to $\sin(\phi')$. The new ϕ' is given by

$$\phi' = \arcsin(y')$$

3. To find θ', $\hat{r}'$ was projected onto the x-z plane by using the matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ and the definition of the dot product. Therefore,

$$\theta' = \arccos(\frac{\vec{rxz} \cdot \hat{z}}{|\vec{rxz}|}) = \arccos(\frac{z'}{\sqrt{x'^2 + z'^2}})$$

- The input value for the reciprocating probe arm controller.

A set point value that determines the distance the collimator entrances will have to the center of the tokamak must be dialed into the probe arm controller. The value depends on whether the gas is on or off, a parameter controlled by MAST. The calculations are

$$pdo = coll\_dist - .165$$

$$SpGOn = 1200 - (2526 - RDist * 1000. - pdo * 1000.)$$

$$SpGOff = 1200 - (2526 - RDist * 1000. - 105.4 - pdo * 1000)$$

where coll_dist is the distance from the end of the reciprocating probe arm and the collimator entrance farthest from it.

After the above calculations are implemented, all of the parameters are written into the new .nml file. Finally, the script runs run_orbit_nml.py, prompts the user to run plot_orbits_combined.py, and gives the value to be physically entered to the controller of the probe arm.

A separate script, setpoint.py, is created to quickly determine where the detectors need to be repositioned inbetween runs. This script also calculates where the protons intersect the z = 0 plane within the area of emissivity; it reads the Orbit Code output file on particle trajectories and searches for radial positions whose corresponding z-position are relatively close to 0. By calculating the radial positions of protons that intersect the z = 0 plane, the data taken by the CFPD can be more accurately compared to the data taken by the neutron camera.

VI. RESULTS

By changing the way values are inputted for the Fortran section and how the python section is ran, the new structure of the Orbit Code is shown in figure 15.

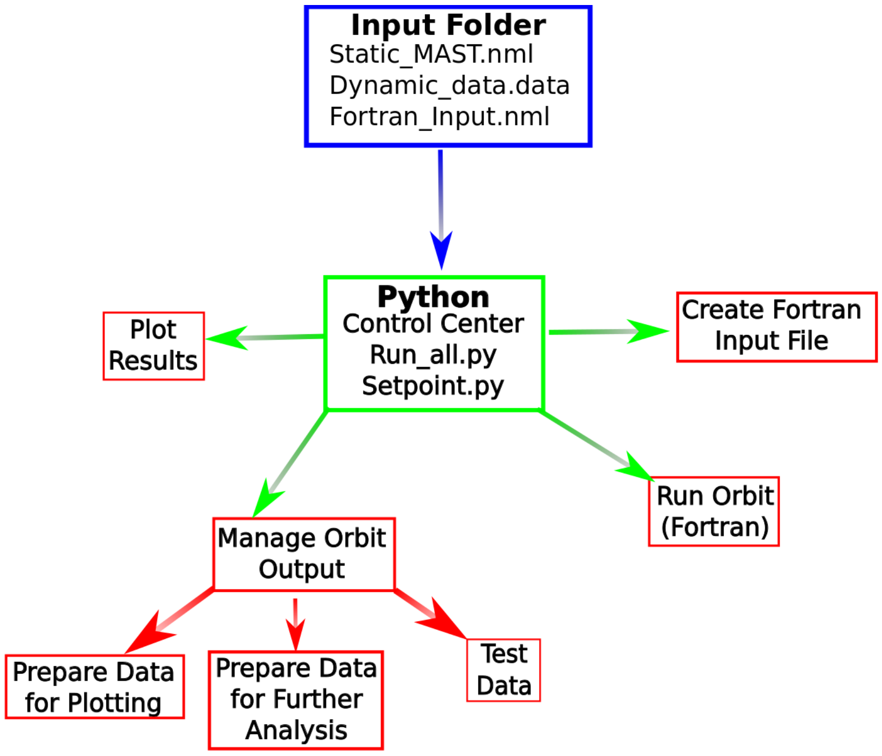


FIG. 15: Current Structure of the Orbit Code

After all of the changes, the new output graphs appear as shown in figure 16.

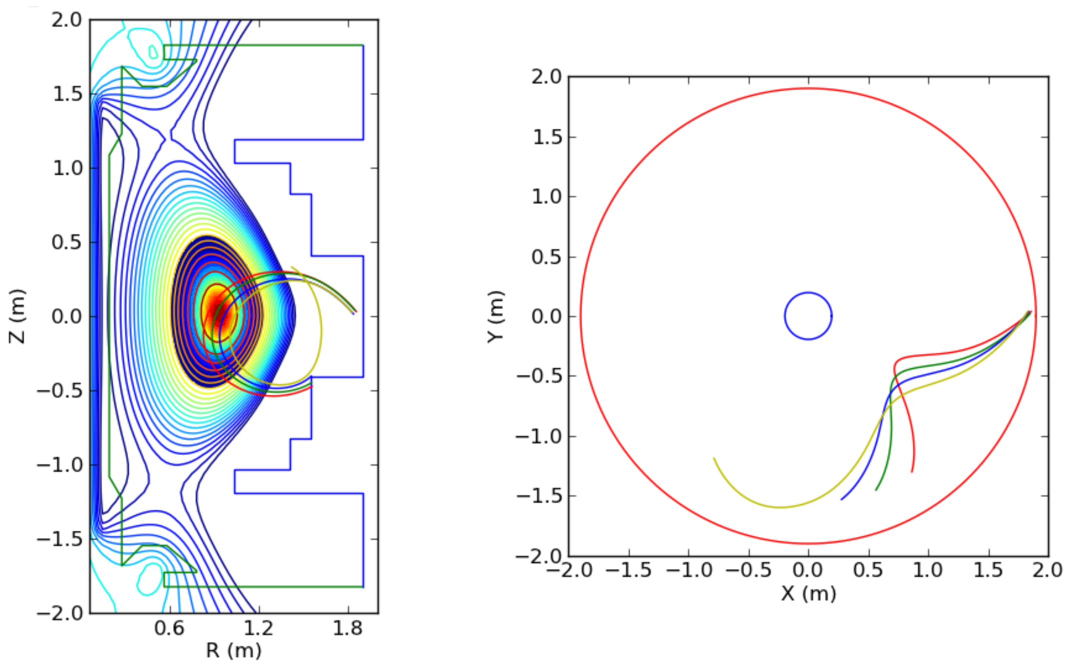


FIG. 16: Current Output Graph of the Orbit Code

The new graph correctly represents the physical position of the detectors relative to each other. The lines also have different colors to better differentiate the proton trajectories.

VII. POST EXPERIMENT APPLICATIONS

Along with the ability to calculate the trajectory and model the emissivity of the plasma, the Orbit Code is also capable of solving for the theoretical rates which should be measured by the detector. These theoretical rates can be compared emissivity rates obtained through experimentation to determine what the proper emissivity model should be. This process will be shown for the data collected from a plasma that had sawtooth instability. Figure 17 shows the particle counts obtained from a sawtooth instability, the name for the instability comes from the sawtooth pattern seen in the plot.

Using these plots, it is possible to calculate the charged particle rates seen by each detector at a given moment in time. The Orbit Code is capable of using the emissivity function inputted in the Static_MAST.nml and the calculated trajectories to determine what the theoretical particles rates are. These two different rates can be compared to determine what the proper emissivity function is.

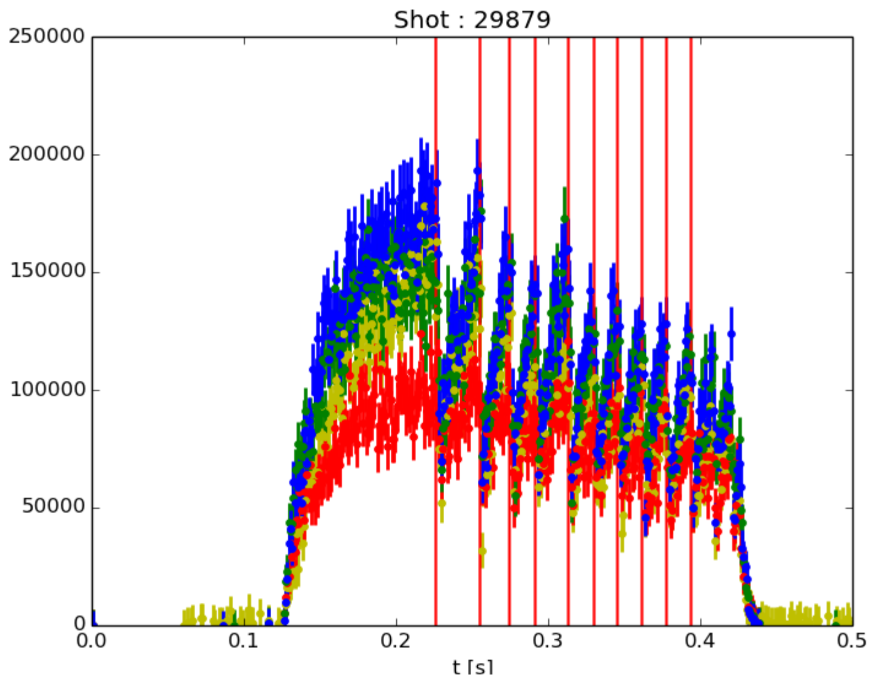


FIG. 17: Plot of experimental data of counts vs time (s)

For example, emissivity function a, is a simple exponential function that is dependent on a radial position, r, a height, z, and the emissivity relative to the emissivity at the origin of the fusion reactions ψ_{rel}, given by the Efit file:

$$S(r, z, \psi_{rel}) = \alpha \psi_{rel}^{\lambda} \tag{a}$$

α and λ are two constants which need to be determined. The emissivity of equation a, .225 seconds into the plasma shot, is shown in figure 18.

The comparison between the rates produced from function a and the experimental data is shown in figure 19.

At time .230 seconds, the emissivity profile and rate comparisons are shown in figures 20 and 21.

If a different model, emissivity function b, was used the following graphs would be achieved.

$$S(r, z, \psi_{rel}) = Ae^{\left(-(\frac{rr_0}{\sigma})^2\right)} \tag{b}$$

The emissivity of equation b, .225 seconds into the plasma shot, is shown in figure 22.

The comparison between the rates produced from function a and the experimental data is

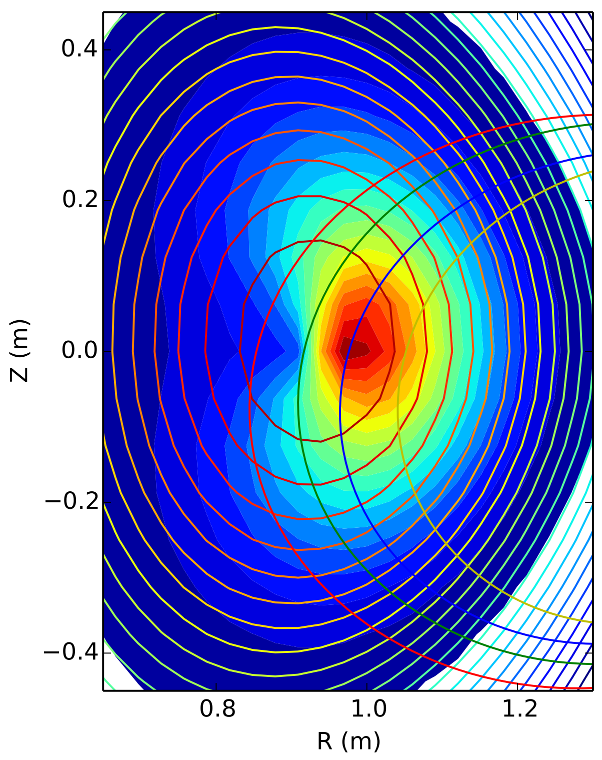


FIG. 18: Emissivity Profile using equation a at time .225 s.

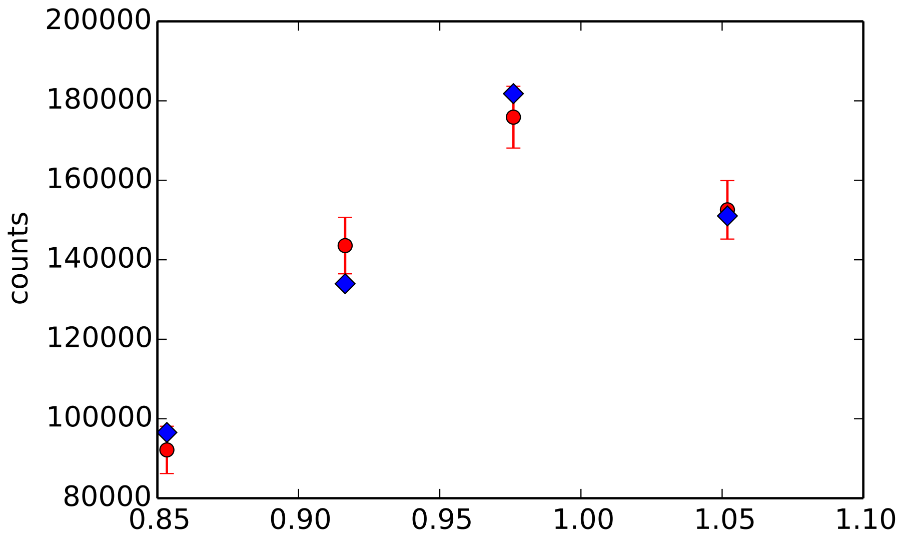


FIG. 19: Emissivity Profile using equation a at time .225 s. The red points are the experimental data and the blue points are the theoretical rates.

shown in figure 23.

At time .230 seconds, the emissivity profile and rate comparisons are shown in figures 24

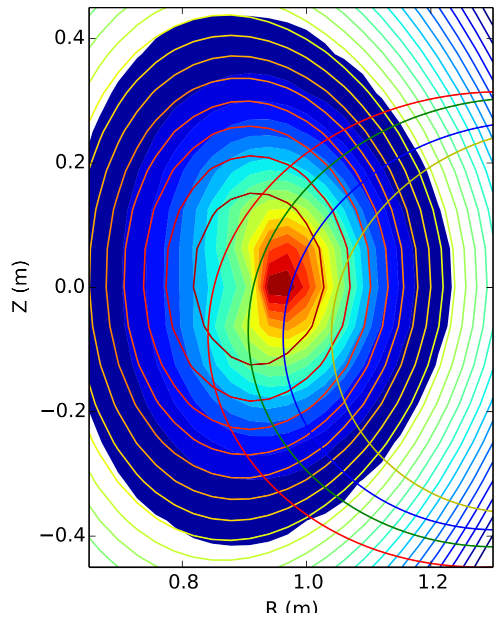


FIG. 20: Emissivity Profile using equation a at time .230 s.

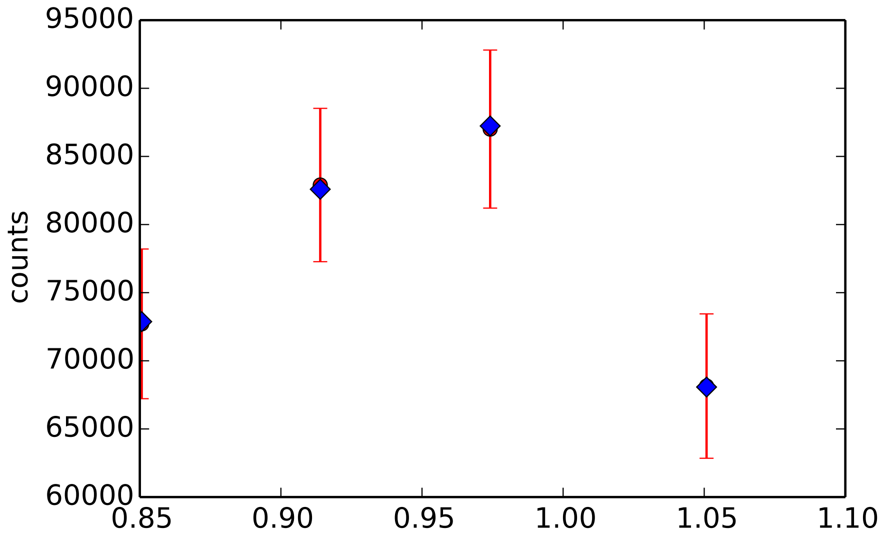


FIG. 21: Emissivity Profile using equation a at time .230 s. The red points are the experimental data and the blue points are the theoretical rates.

and 25.

By comparing the two models at different times, it is shown that neither are equal to the experimental rates at all channels at all times however emissivity function b stays within the accepted error of the experimental rates through both times. Additionally, the times chosen are the peak and trough of the sawtooth instability, where the experimental and theoretical

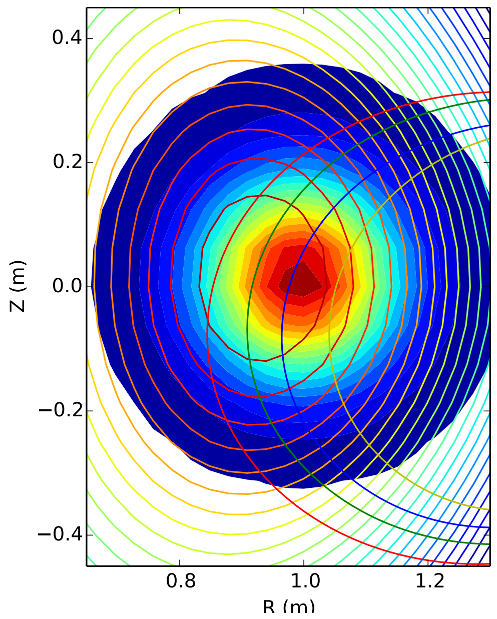


FIG. 22: Emissivity Profile using equation b at time .225 s.

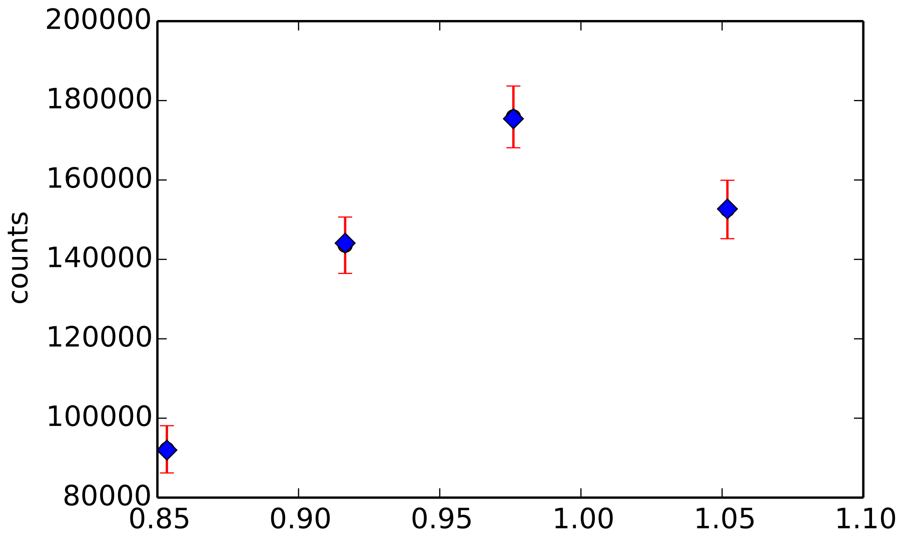


FIG. 23: Emissivity Profile using equation b at time .225 s. The red points are the experimental data and the blue points are the theoretical rates.

rates both show large differences in the emission of charged particle rates.

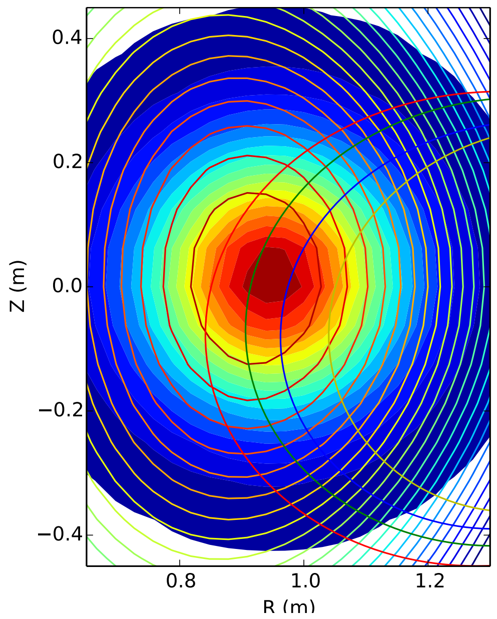


FIG. 24: Emissivity Profile using equation b at time .230 s.

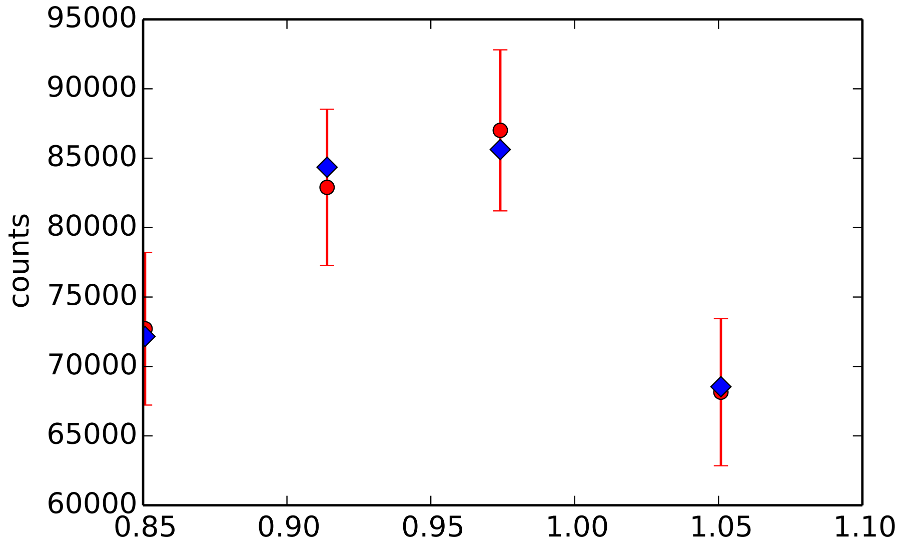


FIG. 25: Emissivity Profile using equation b at time .230 s. The red points are the experimental data and the blue points are the theoretical rates.

VIII. CONCLUSION

The modified Orbit Code is an integral part of planning each experimental run for the charged fusion product diagnostic. Inputting efit files of different times of the same plasma into the Orbit Code provides an insight into how the plasma behaves over time. This insight allows for a better educated decision of where the detectors need to be positioned in order to probe as many different sections of the plasmas as possible. These theoretical models also determine the safest proximities the detectors can have to the plasma. This important safety feature ensures that the system will remain out of harms way as any damage to the detector system may also damage the tokamak causing unpredictable consequences. The code also calculates the set point value for the control box ensuring an accurate position of the reciprocating probe arm.

For now, the necessary approximations to the Orbit Code have been complete. The geometry for the collimator shape, such as converting from a square to a circular collimator, does not need to be changed in the Orbit Code. This is due to the deviation between the solid angle of acceptances to be less than 1%. In other words, the difference between the acceptances is small enough for us to accept the acceptance of a square collimator as a proper approximation. Additionally, no correction factor needs to be included when calculating the total efficiency of the detectors as the segmentation of the collimator increases. Even though there was a noticeable drop in efficiency between 1 segment and 9, the deviation between the efficiencies was less than 1%. Furthermore, the efficiencies seem to be asymptotic with respect to the number of collimator segments as the difference in efficiency between 121 segments and 625 segments is 0.001 (as shown in Table 2).

IX. ACKNOWLEDGEMENTS

I would like to acknowledge Dr. Werner U. Boeglin, Dr. Douglass Darrow, and Mrs. Ramona Valenzuel Perez for giving me the opportunity and guidance to assist in this project. Scott Allan and Nigel Thomas-Davies for their tireless effort in aiding the FIU Plasma Physics Research Group along with the rest of the MAST workforce. Finally, the McNair program for their support in this endeavor, the undergraduate volunteers could not have made it to Culham, England without them. Lastly, I'd like to thank Adrianna Angulo, Pierre Avila, and Carlos Lopez for their help in making the Orbit Code as user friendly and error free as possible. This material is based upon work supported by the Department of Energy under Award Number DE-SC0001157.

Appendix A: Program to Calculate Solid Angle of Acceptance of a Cylinder

This follow is the code to calculate the solid angle of acceptance of a cylindrical collimator. The Main_Accept program performs the integration of equation 1.

```
PROGRAM Main_Accept


use acceptancemod


implicit none


INTERFACE
       FUNCTION max_theta(r,d)
              REAL, INTENT(IN) :: r,d
              REAL :: max_theta
       END FUNCTION max_theta

       FUNCTION cc(theta, d)
              REAL, INTENT(IN) :: theta,d
              REAL :: cc
       END FUNCTION cc

       FUNCTION area_overlap(x1,y1,r1,x2,y2,r2)
              REAL, INTENT(IN) :: x1,y1,r1,x2,y2,r2
              REAL :: area_overlap
       END FUNCTION area_overlap

       FUNCTION acceptance(zeta)
              REAL, INTENT(IN) :: zeta
              REAL :: acceptance
       END FUNCTION acceptance
       FUNCTION acceptancearea(zeta)
```

```
                REAL, INTENT(IN) :: zeta
                REAL :: acceptancearea
        END FUNCTION acceptancearea

END INTERFACE

real :: s1,s2, angle_max


! Declare local variables

        angle_max = max_theta(rs,dm)
        call qsimp(acceptancearea, 0., angle_max, S1)
        call qsimp(acceptance, 0., angle_max, S2)
        write(*,*) '---------------------------------------------------'
        write(*,*) '---------------------------------------------------'
        write(*,*) '---------------------------------------------------'
        write(*,*) 'The acceptance is ', S1, 'steridians*m^2.'
        write(*,*) '---------------------------------------------------'
        write(*,*) '---------------------------------------------------'
        write(*,*) '---------------------------------------------------'
        write(*,*) 'The solid angle of acceptance is ', S2, 'steridians.'
END PROGRAM Main_Accept
```

The max_theta function calculates the angle at which no particle will enter the collimator entrance

```
!------------------------------------------------------------------------------------------
!-------------------------------------------------------------------------------------------
! Calculates the maxium angle of entry theta.
```

```
Real Function max_theta(r,d)

implicit none

real, intent(in) :: r,d

       max_theta = atan(2.*r/d)

end function max_theta
```

The cc function calculates the center of the ”shadow” caused the particles entering the collimator entrance based on the angle between the incoming particles and the collimator.

```
!-----------------------------------------------------------------------------------------
!-----------------------------------------------------------------------------------------
! Calculates the center x point for the second circle.

real function cc(theta, d)

implicit none

real, intent(in) :: theta, d

       cc = d*sin(theta)/cos(theta)

end function cc
```

The area_overlap function calculates the area of overlap between two circles.

```
REAL FUNCTION area_overlap(x1,y1,r1,x2,y2,r2)

! This function is meant to calculate the area of overlap between two circles.
```

```fortran
! Using the centers and radii of two circles their amount of overlap is
   calculated.

! Declaring the variables

use acceptancemod

implicit none

real, intent(in):: x1,y1,r1,x2,y2,r2



! Determining distace between the centers.



      write(*,*) 'The center of a circle is', [x1,y1],' with radius', r1,'.'
      write(*,*) 'The center of the other circle is', [x2,y2], 'with radius',
         r2,'.'

      distance = sqrt((x2-x1)**2 + (y2-y1)**2)
      write(*,*) 'Distance between the centers is', distance

! Determining if the circles overlap.



! If the distance between the centers is less than the sum of the radii the
   circles intersect or overlap.

      if (distance .lt. r1 + r2) then
```

```
! If the some of one radii and the distance between the centers is less than the
   other radii
! then the circles overlap.

              if (distance + r1 .le. r2 .or. distance + r2 .le. r1) then
                     ol = 0.

              else


                     ol = 1.



              end if

! If the distance between the centers is greater than the sum of the radii there
   is no overlap.

       else if (distance .ge. r1 + r2) then

              ol = 2.

       end if

! Calculating for the area of overlap if ol = 1.

       if (ol .eq. 1.) then

! Distance between x coordinates of centers.

              dx = abs(x2 - x1)
```

```
               write(*,*) 'The difference between the x coordinates of the center
                  is', dx

! Distance between y coordinates of centers.

               dy = abs(y2-y1)
               write(*,*) 'The difference between the y coordinates of the center
                  is', dy

! Center of the area of overlap.

               center_ellipse = abs((distance**2 + r1**2 - r2**2)/(2*distance))
               write(*,*) 'Center of the ellipse is', center_ellipse,'from the
                  center of the circles.'

               intersectx(1) = x1 + dx*center_ellipse/distance +
                  (dy/distance)*sqrt(r1**2 - center_ellipse**2)

               intersectx(2) = x1 + dx*center_ellipse/distance -
                  (dy/distance)*sqrt(r1**2 - center_ellipse**2)



               intersecty(1) = y1 + dy*center_ellipse/distance -
                  (dx/distance)*sqrt(r1**2 - center_ellipse**2)



               intersecty(2) = y1 + dy*center_ellipse/distance +
                  (dx/distance)*sqrt(r1**2 - center_ellipse**2)

               write(*,*) 'The points of intersection are (', intersectx(1),',',
                    intersecty(1),') and (', intersectx(2),',',intersecty(2),').'


! Calculating the area of intersection



                height = sqrt((intersectx(1) - intersectx(2))**2 +(intersecty(2) -
                    intersecty(1))**2)
                write(*,*) 'The height of the ellipse is', height

! First Circle of the Left

                if (x2 .ge. x1) then

                ! Geometric calculations.

                        theta = 2.*asin(height/(2*r2)) ! Angle of First Circle's
                            Sector
                        u1 = sqrt(r2**2 - (height/2)**2)
                        area1 = theta*(r2**2)/2 - u1*height/2 ! Area of First
                            Circle



                        if (distance > u1) then

                                phi = 2*asin(height/(2*r1)) ! Angle of Second
                                    Circle's Sector
                                u2 = sqrt(r1**2 - (height/2)**2)

                                area2 = phi*(r1**2)/2 - u2*height/2 ! Area of
```

```
                      Second Circle


            else if (distance == u1) then

                   area2 = 0.5*pi*r1**2 ! Area of Second Circle


            else if (distance < u1) then

                   phi = 2*asin(height/(2*r1)) ! Angle of Second
                      Circle's Sector

                   u2 = sqrt(r1**2 - (height/2)**2)

                   area2 = (2*pi - phi)*(r1**2)/2 + u2*height/2 ! Area
                      of Second Circle

            end if

            area_overlap = area1 + area2
            write(*,*) 'When OL=', ol,'the over lap area
               is',area_overlap

!                    First Circle on the Right
!
       else if (x2 .lt. x1) then


!                    Geometric calculations.
```

```
!
                    theta = 2.*asin(height/(2*r2)) ! Angle of Second Circle's
                       Sector
                    u1 = sqrt(r2**2 - (height/2)**2)
                    area1 = theta*(r2**2)/2 - u1*height/2 ! Area of Second
                       Circle

                    write(*,*) 'Area1 is', area1

                    if (distance > u1) then

                           phi = 2*asin(height/(2*r1)) ! Angle of Second
                              Circle's Sector
                           u2 = sqrt(r1**2 - (height/2)**2)

                           area2 = phi*(r1**2)/2 - u2*height/2 ! Area of
                              Second Circle
                           write(*,*) 'Area2 is', area2



                    else if (distance == u1) then

                           area2 = 0.5*pi*(r1**2) ! Area of Second Circle

                    else if (distance < u1) then

                           phi = 2*asin(height/(2*r1)) ! Angle of Second
                              Circle's Sector
                           u2 = sqrt(r1**2 - (height/2)**2)
```

```
                              area2 = (2*pi - phi)*(r1**2)/2 + u2*height/2 ! Area
                                  of Second Circle

                      end if

                      area_overlap = area1 + area2
                      write(*,*) 'When OL=', ol,'the over lap area
                          is',area_overlap



              end if

!             If there is a complete overlap.

      else if (ol .eq. 0.) then

              if (r1 < r2) then

                      area_overlap = pi*(r1**2)
                      write(*,*) 'When OL=', ol,'the over lap area
                          is',area_overlap



              else if (r1 >= r2) then
                      area_overlap = pi*(r2**2)
                      write(*,*) 'When OL=', ol,'the over lap area
                          is',area_overlap

              end if
```

```
!              If there is no overlap.


       else if (ol == 2.) then

              area_overlap = 0.

              write(*,*) 'When OL=', ol,'the over lap area is',area_overlap


       end if



end function area_overlap
```

The acceptancearea function provides equation 1 for the program to integrate.

```
!---------------------------------------------------------------------------------------------

!---------------------------------------------------------------------------------------------

! Calculates the acceptance with the direction of the particles taken into
   consideration


real function acceptancearea(zeta)


USE acceptancemod


implicit none




       Interface


       FUNCTION cc(theta, d)

              REAL, INTENT(IN) :: theta,d

              REAL :: cc

       END FUNCTION cc
```

```
        FUNCTION area_overlap(x1,y1,r1,x2,y2,r2)
                REAL, INTENT(IN) :: x1,y1,r1,x2,y2,r2
                REAL :: area_overlap
        END FUNCTION area_overlap

        End Interface

        real, intent(in) :: zeta
        real :: c2_area
        xp = cc(zeta, dm)
        ic = area_overlap(x,y,r,xp,yp,rp)

        if (xp == 0. .and. rp==r) then
                transmission = 1.
        else
                c1_area = pi*rp**2
                transmission = ic/c1_area
        end if

        c2_area = pi*rp**2
        acceptancearea = 2*pi*transmission*c2_area*cos(zeta)*sin(zeta)
end function acceptancearea
```

The acceptance function is a weighted integral of the solid angle of acceptance of a cylinder.

```
!---------------------------------------------------------------------------------------------
!---------------------------------------------------------------------------------------------
! Calculates the acceptance

real function acceptance(zeta)
```

```
USE acceptancemod

implicit none


        Interface

        FUNCTION cc(theta, d)
                REAL, INTENT(IN) :: theta,d
                REAL :: cc
        END FUNCTION cc

        FUNCTION area_overlap(x1,y1,r1,x2,y2,r2)
                REAL, INTENT(IN) :: x1,y1,r1,x2,y2,r2
                REAL :: area_overlap
        END FUNCTION area_overlap

        End Interface

        real, intent(in) :: zeta
        real :: c2_area
        xp = cc(zeta, dm)
        ic = area_overlap(x,y,r,xp,yp,rp)

        if (xp == 0. .and. rp==r) then
                transmission = 1.
        else
                c1_area = pi*rp**2
                transmission = ic/c1_area
```

```
       end if


       c2_area = pi*rp**2

       acceptance = 2*pi*transmission*sin(zeta)

end function acceptance
```

The makefile combines all of the files into an executable.

```
#-----Makefile for Soild Angle of Acceptance-------


%.o: %.f

       $(F90) -c $(FFLAGS1) $< -o $@


%.o: %.f90

       $(F90) -c $(FFLAGS) $< -o $@


Acceptance_OBJ = acceptancemod.o areaoverlap.o maxtheta.o cc.o qsimp.o trapzd.o

   MainAccept.o acceptance.o acceptancearea.f90


#INCLUDE = ../include/


F90    = gfortran


#FFLAGS = -ffixed-line-length-none $(FDFLAG) \

#         -fno-align-commons -w -fno-automatic -I$(INCLUDE)


FFLAGS = -ffree-form


FFLAGS1 =

#-----------------------------------------------------------

all: $(Acceptance_OBJ)
```

```
#       if (test -f "acceptancemod.mod"); then rm acceptancemod.mod; fi
        $(F90) $(Acceptance_OBJ) -o accept

#example of creating object files the long way
#magfld.o: magfld.f
#       $(F90) -c magfld.f $(FFLAGS) -o $@
#----------------------------------------------------------


clean:
        rm *.o *.a *.mod accept testing

#acceptancemodcheck.mod: acceptancemod.f90
#       if (test -f "acceptancemod.mod"); then rm acceptancemod.mod; fi
#       $(F90) -c $(FFLAGS) $< -o /temp/xx

#acceptancemod.mod: acceptancemod.f90
#       $(F90) -c $(FFLAGS) $< -o acceptancemod.o

#area_overlap.o : area_overlap.f90
#       $(F90) -c $(FFLAGS) $< -o area_overlap.o

#max_theta.o : max_theta.f90
#       $(F90) -c $(FFLAGS) $< -o max_theta.o

#qsimp.o : qsimp.f
#       $(F90) -c $(FFLAGS1) $< -o qsimp.o

#trapzd.o : trapzd.f
#       $(F90) -c $(FFLAGS1) $< -o trapzd.o
```

```
#acceptance.o : acceptance.f90
#	$(F90) -c $(FFLAGS) $< -o acceptance.o

#Main_Accept.o : Main_Accept.f90
#	$(F90) acceptancemod.o area_overlap.o max_theta.o qsimp.o trapzd.o
   acceptance.o -c $(FFLAGS) $< -o Main_Accept.o
```