DarsakX User Manual

v. 2023-1

Neeraj K. Tiwari

January 25, 2024

Contents

1 Introduction

DarsakX is a Python package developed to estimate the imaging performance of an X-ray telescope constructed with coaxially aligned multiple shells in the Wolter-1 optics (WO1) configuration or a conical approximation of WO1. Each shell consists of two mirrors: a paraboloid as the primary mirror and a hyperboloid as the secondary mirror. An X-ray photon reaches the detector after undergoing double reflection within a shell, first with the primary mirror and then with the secondary mirror.

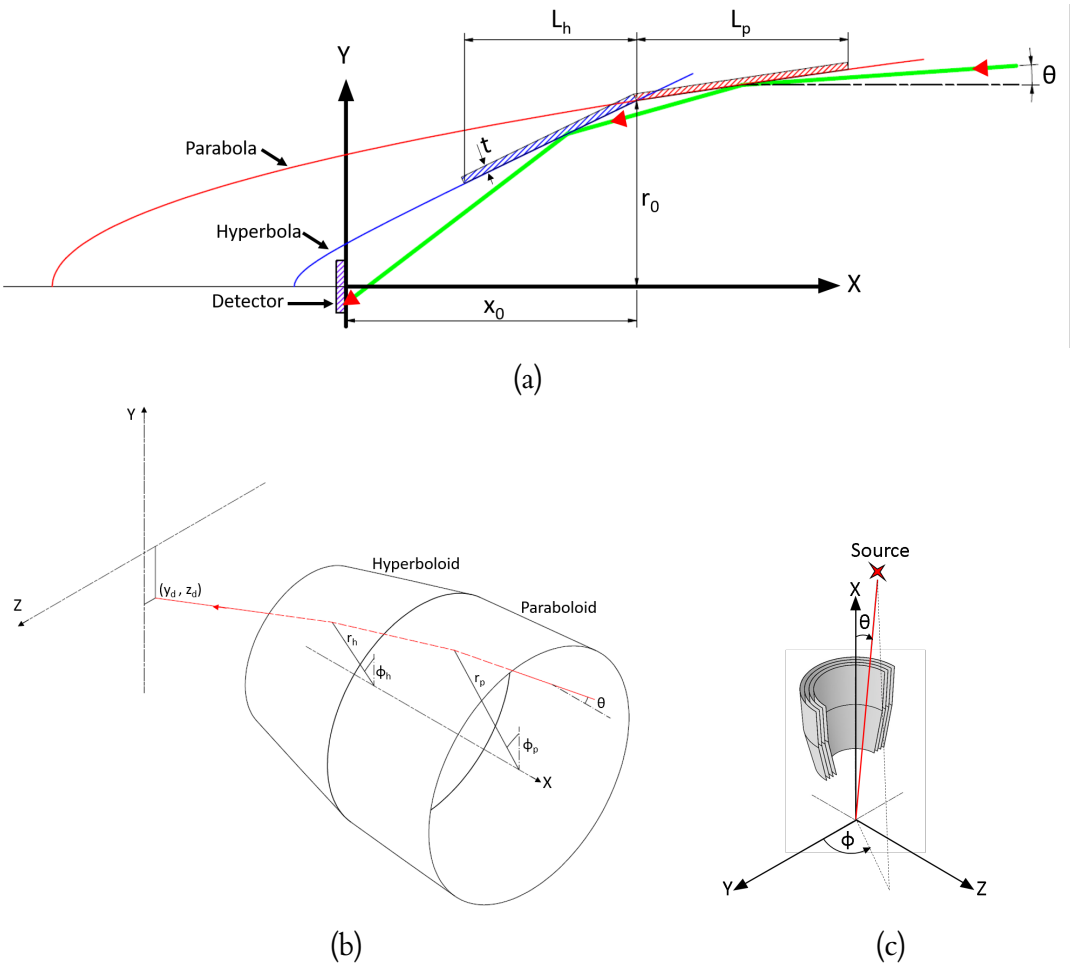

Figure 1: (a) A 2D schematic diagram of WO1 with various geometrical parameters. (b) A ray trace through the full shell of WO1. (c) Definition of the source location with respect to the cross-sectional view of the multi-shell WO1.

Figure-1(a) displays the various parameters required to define a single shell in the WO1 configuration. x_0 represents the focal length of this shell, while L_h and L_p represent the projected lengths of the paraboloid and hyperboloid sections along the X axis, respectively. r_0 is the radius of the shell, defined as the distance between the intersection of the primary and secondary mirrors and the optical axis of the shell. The variable t represents the thickness of the shell.

Figure-1(b) shows a full shell of WO1 along with a ray and its trajectory. Figure-1(c) demonstrates how the source location is defined with respect to the telescope coordinate system, which consists of multiple shells, as depicted in its sectional view. The source loca-

tion is defined by the angle θ it makes with respect to the optical axis of the shell. The angle ϕ can be used to define the source location in the YZ plane. However, since we assume that the shell is completely symmetric about the X axis, the telescope's performance is independent of the angle ϕ, hence we consider $\phi = 0$ at all times. There is another parameter, ξ, which is defined as the ratio of the incident angles at which an on-axis ray strikes the primary and secondary mirrors at their intersection[ref].

In DarsakX, a telescope can be defined by arranging multiple shells, with each shell resembling Figure-1(b), coaxially aligned along the X axis. Each shell can be defined by its geometrical parameters $(x_0, r_0, L_h, L_p, t, \xi)$.

2 DarsakX Installation

3 DarsakX Raytrace

In DarsakX, the rtrace function is used to trace the rays for a multi-shell telescope. It traces the rays originating from a source at infinity, which is defined by the parameters θ and $\phi = 0$, until they reach the telescope's detector. The traced results obtained by the rtrace function provide information about each ray, as shown in Table 1.

Table 1: rtrace output parameters

Output Parameters	**Symbol**
The direction of each ray before it hits the primary mirror.	$[n_{ip_x}, n_{ip_y}, n_{ip_z}]$
The intersection position of each ray with the primary mirror.	$[x_p, r_p, \phi_p]$
The direction of each ray once it is reflected from the primary mirror.	$[n_{rp_x}, n_{rp_y}, n_{rp_z}]$
The intersection position of each ray with the secondary mirror.	$[x_h, r_h, \phi_h]$
The direction of each ray once it is reflected from the secondary mirror.	$[n_{rhx}, n_{rhy}, n_{rhz}]$
The intersection position of each ray with the detector.	y_d, z_d
Ray incident angle on primary mirror	α_p
Ray incident angle on secondary mirror	α_h

It's important to note that the rtrace function only provides geometrical information about each ray. It does not take into account the reflectivity properties of the mirror shell or the thin film coating properties on the mirror. Therefore, the ray-traced results are independent of the energy of the rays.

The rtrace function can be used as explained in the following Code Example 3.1.

Code Example 3.1

```
1. from darsakx import rtrace
2. raytrace_data=rtrace(Radius=r0,Focallength=x0,Lengthpar=lp,Lengthhyp=lh
   ,ShellThickness=st,Theta=theta,Raydensity=rd,Xi=xi,DetectorPosition=0,
   NumCore=None,SurfaceType="wo", Error="no",Approx="no",Gp=None,dGp
   =None,Gh=None,dGh=None)
```

3.1 rtrace Parameters

- **Radius**: Radius is represented as a 1D list. Radius of each shell can be provided in the form of list, as shown below.
  $r0 = [r_{01}, r_{02}, r_{03}, r_{04}.....r_{0n}]$ # in mm
  or it can be provided in the form of numpy array as well.
  $r0 = numpy.array([r_{01}, r_{02}, r_{03}, r_{04}.....r_{0n}])$ # in mm

- **Focallength**: Focal-length is is also represented as a 1D list. Focal-length of each shell can be provided in the form of list, as shown below.
  $x0 = [x_{01}, x_{02}, x_{03}, x_{04}.....x_{0n}]$ # in mm
  or it can be provided in the form of numpy array as well.
  $x0 = numpy.array([x_{01}, x_{02}, x_{03}, x_{04}.....x_{0n}])$# in mm
  Focal lengths of each shell can be same or they can be different as well.

- **Lengthpar**: Length of parabola section is also represented as a 1D list. Length of parabola section of each shell can be provided in the form of list, as shown below.
  $lp = [Lp_1, Lp_2, Lp_3, Lp_4.....Lp_n]$# in mm
  or it can be provided in the form of numpy array as well.
  $lp = numpy.array([Lp_1, Lp_2, Lp_3, Lp_4.....Lp_n])$ # in mm

- **Lengthhyp**: Length of hyperbola section is also represented as a 1D list. Length of hyperbola section of each shell can be provided in the form of list, as shown below.
  $lh = [Lh_1, Lh_2, Lh_3, Lh_4.....Lh_n]$ # in mm
  or it can be provided in the form of numpy array as well.
  $lp = numpy.array([Lh_1, Lh_2, Lh_3, Lh_4.....Lh_n])$ # in mm

- **ShellThickness**: Thickness of the shell is also represented as a 1D list. Thickness of each shell can be provided in the form of list, as shown below.
  $st = [t_1, t_2, t_3, t_4.....t_n]$ # in mm
  or it can be provided in the form of numpy array as well.
  $st = numpy.array([t_1, t_2, t_3, t_4.....t_n])$ # in mm

  Note: The length of Radius, Focallength, Lengthpar, Lengthhyp, and ShellThickness should be the same.

- **Theta:** Theta, represented as θ in Figure-1, defines the location of the source. The rtrace function can trace the rays for multiple Thetas at once. The Theta can be provided as follows.
  $theta = [theta_1, theta_2, theta_3, theta_4.....theta_n]$ # in degrees
  or it can be provided in the form of numpy array as well.
  $theta = numpy.array([theta_1, theta_2, theta_3, theta_4.....theta_n])$ # in degrees

- **Raydensity:** Ray density is defined as the number of rays per unit area. It can be provided as a floating-point number with units in $rays/cm^2$.

- **Xi:** The Xi (or ξ) is an optional parameter. If not provided, its default value is 1 for all the shells. The Xi value for each shell can be provided in the form of a list, as shown below.
  $Xi = [xi_1, xi_2, xi_3, xi_4.....xi_n]$
  or it can be provided in the form of numpy array as well.
  $Xi = numpy.array([xi_1, xi_2, xi_3, xi_4.....xi_n])$

Note: The length of Radius, Focallength, Lengthpar, Lengthhyp, ShellThickness and Xi should be the same.

- **DetectorPosition:** Detector Position is also an optional parameter, and its default value is zero. Detector Position can be provided as a floating-point number with units in mm.

- **NumCore:** Number of cores that can be provided for parallel processing. This is an optional parameter. The number of cores provided must be less than the maximum number of cores available in the system. Parallel processing is only useful for multiple shells as it distributes the ray tracing jobs of different shells to different cores.

- **SurfaceType:** Surface type defines the type of surface, whether it is Wolter-1 optics (wo) or Conical approximation (co), that has to be considered for all the shells. Surface type is an optional parameter, and its default value is "wo". It can be provided as "wo" for Wolter-1 optics or "co" for conical approximation optics as a string.

- **Error:** This parameter defines figure error present in Wolter-1 optics (`SurfaceType="wo"`) or in Conical approximation (SurfaceType="co"). It is an optional parameter, and its default value is "no" which means either surface is perfect `wo` or perfect `co` without any figure error. If `Error="yes"` is provided, each shell will be considered with figure error. However, the user must provide figure error with more parameters such as `Gp`, `dGp`, `Gh`, and `dGh` for each shell.

- **Approx:** This parameter defines the method, approximate or without-approximate, to trace the rays for the shell that has figure error present in it. This is an optional parameter, and the default method is without-approximate. The user can choose the approximate method by providing `Approx="yes"`.

  Note that this parameter has to be provided only when the parameter `Error="yes"` is set, as the approximate method is used to trace the rays only when non-zero figure error is present in the shell.

- **Gp:** This parameter defines the figure error present in the paraboloid section or primary section for each shell. It has to be provided only when the parameter `Error="yes"` is set. `Gp` has to be provided in a list form where each element in this list has to be a Python function that should provide the value of $G_p(x)$ for a given value of x, as shown below.
  $Gp = [Gp_1, Gp_2, Gp_3, Gp_4.....Gp_n]$ # in mm
  or it can be provided in the form of numpy array as well.
  $Gp = numpy.array([Gp_1, Gp_2, Gp_3, Gp_4.....Gp_n])$ # in mm

- **dGp:** This parameter defines the derivative of the figure error present in the paraboloid section or primary section for each shell. It has to be provided only when the parameter `Error="yes"` is set. `dGp` has to be provided in a list form where each element in this list has to be a Python function that should provide the value of $G'_p(x) = \frac{d(G_p(x))}{dx}$ for a given value of x, as shown below.
  $dGp = [dGp_1, dGp_2, dGp_3, dGp_4.....dGp_n]$ # in mm
  or it can be provided in the form of numpy array as well.
  $dGp = numpy.array([dGp_1, dGp_2, dGp_3, dGp_4.....dGp_n])$ # in mm

- **Gh:** This parameter defines the figure error present in the hyperboloid section or secondary section for each shell. It has to be provided only when the parameter `Error="yes"`

is set. `Gh` has to be provided in a list form where each element in this list has to be a Python function that should provide the value of $G_h(x)$ for a given value of x, as shown below.
$Gh = [Gh_1, Gh_2, Gh_3, Gh_4.....Gh_n]$ # in mm
or it can be provided in the form of numpy array as well.
$Gh = numpy.array([Gh_1, Gh_2, Gh_3, Gh_4.....Gh_n])$ # in mm

- **dGh:** This parameter defines the derivative of the figure error present in the hyperboloid section or secondary section for each shell. It has to be provided only when the parameter `Error="yes"` is set. `dGh` has to be provided in a list form where each element in this list has to be a Python function that should provide the value of $G'_h(x) = \frac{d(G_h(x))}{dx}$ for a given value of x, as shown below.
$dGh = [dGh_1, dGh_2, dGh_3, dGh_4.....dGh_n]$ # in mm
or it can be provided in the form of numpy array as well.
$dGh = numpy.array([dGh_1, dGh_2, dGh_3, dGh_4.....dGh_n])$ # in mm

  Note: The length of Radius, Focallength, Lengthpar, Lengthhyp, ShellThickness, Xi, Gp, dGp, Gh and dGh should be the same.

3.2 rtrace Output

Once the rays have been traced as demonstrated in Code Example 3.1, ray-traced data can be acquired using the following steps.

Code Example 3.2

```
1. data=raytrace_data.data( )
```

Users do not have to extract this data unless they want to attempt something more advanced than the capabilities of the default post-processing options provided in DarsakX, which are explained in the next section. Therefore, for now, users can choose to skip this sub-section if they wish.

The data obtained from Code Example 3.2 is structured as a multidimensional list. This traced data contains comprehensive information about each ray's trajectory, from the source to the detector, acquired for various values of θ. If the telescope is constructed with multiple shells, and rtrace has been simulated for multiple values of θ, the data will contain information about every ray corresponding to each shell for each value of θ. Table-1 presents the information that can be obtained for each ray.

The data's first dimension has two components: the first one contains information about every ray corresponding to each θ and each shell, and the second one contains the values of all θ. You can extract them as follows:

Code Example 3.3

```
1. rays_information_for_allTheta_allShell=data[0]
2. theta=data[1]
```

The information regarding rays for all the shells at a specific value of θ =theta_i can be extracted as follows:

Code Example 3.4

```
1. theta_i=theta[i]
2. rays_information_for_allShell=rays_information_for_allTheta_allShell [i]
```

The information about all the rays corresponding to the i^{th} shell (shells are arranged in increasing order of their diameter) can be extracted as follows:

Code Example 3.5

```
1. rays_information=rays_information_for_allShell [i]
```

The information about all the rays, as described in Table-1, corresponding to a specific θ and a particular shell can be extracted as follows:

Code Example 3.6

```
1. x_p=rays_information["xp"]
2. r_p=rays_information["rp"]
3. φ_p=rays_information["phi_p"]
4. n_prx=rays_information["npx"]
5. n_pry=rays_information["npy"]
6. n_prz=rays_information["npz"]
7. x_h=rays_information["xh"]
8. r_h=rays_information["rh"]
9. φ_h=rays_information["phi_h"]
10. n_hrx=rays_information["nhx"]
11. n_hry=rays_information["nhy"]
12. n_hrz=rays_information["nhz"]
13. y_d=rays_information["yd"]
14. z_d=rays_information["zd"]
15. α_p=rays_information["theta_p"]
16. α_h"=rays_information["theta_h"]
```

4 DarsakX Post Processing

This section explains how DarsakX utilizes the ray-traced data to estimate various performance parameters of an X-ray telescope, such as the Point Spread Function (PSF), Energy Encircled Fraction (EEF), Effective Area, Vignetting Factor, the shape of the sharp focusing detector, and 3D visualization of ray-traced data.

So far, during the ray trace, coating properties on the mirror or mirror reflectivity were not considered. Hence, the ray trace was independent of X-ray photon energy. However,

for post-processing, we will now take into account the mirror's coating and reflectivity properties. Therefore, the telescope's performance will depend on the photon energy.

4.1 PSF

Continuing from Code Example 3.1, the PSF can be estimated as follows:

Code Example 4.1

```
1. psf=raytrace_data.psf(Thetaforpsf= theta,Pixel_size=pixel_size,Theta_Reflectivity
   =theta_rf,Reflectivity_p=rfp,Reflectivity_h=rfh,IsReflectivityCon="yes")
```

4.1.1 Parameters

- **Thetaforpsf:** This is the θ at which the PSF has to be generated. The θ can have a value from the list of θ values provided by the user in Code Example 3.1 for which rays have been traced. If the provided θ does not match any value in the list of θ values provided in Code Example 3.1, the closest matching θ from the list will be considered to generate the PSF. The θ unit is in degrees.

- **Pixel_size:** This is the detector's pixel size. Its unit is in microns.

- **Theta_Reflectivity, Reflectivity_p, Reflectivity_h, and IsReflectivityCon:** These four parameters are used to define the reflectivity properties of each shell of the telescope. Reflectivity properties can vary between primary and secondary mirrors within a shell, and they can also differ between shells.

  If the reflectivity property of each shell is different, set the parameter IsReflectivityCon to "no"; its default value is "yes" which implies that all shells have the same reflectivity property.

  The reflectivity of a photon on a reflective surface depends on the energy of the photon and the incident angle. Therefore, each mirror (primary or secondary) in a shell, with a specific type of multi-layer coating property, has a reflectivity curve. This curve provides reflectivity as a function of incident angle for a constant energy. Users must provide this reflectivity curve for both mirrors in each shell.

  For known coating parameters on a mirror surface, a reflectivity curve can be obtained from the Python package DarpanX. To learn how to generate a reflectivity curve using DarpanX, please refer to the example.

  In DarsakX, the reflectivity curve can be provided as discrete data points representing reflectivity as a function of incident angle. DarsakX will interpolate these data points to calculate reflectivity for all sets of incident angles.

  If IsReflectivityCon="yes" then Theta_Reflectivity, Reflectivity_p, and Reflectivity_h can be provided as follows:

  Theta_Reflectivity=$[theta_1, theta_2, theta_3, \ldots, theta_n]$
  Reflectivity_p=$[rp_1, rp_2, rp_3, \ldots, rp_4]$
  Reflectivity_h=$[rh_1, rh_2, rh_3, \ldots, rh_4]$

Here, Theta_Reflectivity contains the set of incident angles for which reflectivity is provided for the primary and secondary mirrors. Reflectivity_p and Reflectivity_h represent the reflectivity of the paraboloid mirror (primary mirror) and hyperboloid mirror (secondary mirror), respectively, corresponding to the incident angles provided in the list of Theta_Reflectivity. Note that in this case, a separate reflectivity curve is provided for both the primary and secondary mirrors in each shell. These curve remains the same for all shells but differs between the primary and secondary mirrors within a single shell.

If IsReflectivityCon="no" then different reflectivity curves must be provided for both mirrors in each shell. Hence, Theta_Reflectivity, Reflectivity_p, and Reflectivity_h can be provided as follows:

$$\text{Theta\_Reflectivity=} \quad \big[[theta_{11}, theta_{21}, theta_{31}, \ldots, theta_{n1}], \; [theta_{12}, theta_{22}, theta_{32}, \ldots, theta_{n2}], \; \vdots \; [theta_{1r}, theta_{2r}, theta_{3r}, \ldots, theta_{nr}]\big]$$

$$\text{Reflectivity\_p=} \quad \big[[rp_{11}, rp_{21}, rp_{31}, \ldots, rp_{n1}], \; [rp_{12}, rp_{22}, rp_{32}, \ldots, rp_{n2}], \; \vdots \; [rp_{1r}, rp_{2r}, rp_{3r}, \ldots, rp_{nr}]\big]$$

$$\text{Reflectivity\_h =} \quad \big[[rh_{11}, rh_{21}, rh_{31}, \ldots, rh_{n1}], \; [rh12, rh_{22}, rh_{32}, \ldots, rh_{n2}], \; \vdots \; [rh_{1r}, rh_{2r}, rh_{3r}, \ldots, rh_{nr}]\big]$$

In Theta_Reflectivity, rows represent incident angles for shells, and similarly, for Reflectivity_p and Reflectivity_h, rows represent reflectivity for shells.

4.1.2 Output

Executing Code Example 4.1, it will, by default, generate a PSF plot. Users can extract PSF data in the form of *Intensity* and *Range*, as shown below:

Code Example 4.2

```
1. Intensity, Range= psf
```

Where *Intensity* is a 2D numpy array that contains intensity values corresponding to each pixel, and *Range* provides details about the position of the extreme pixels on the detector plane. You can extract the intensity of a specific pixel as *Intensity*$[y_i , z_j]$.

4.2 EEF

The Energy-Encircled Fraction, $EEF_{x\%}$, defines the size of the central region of the PSF that contains $x\%$ of the total energy of the entire PSF. The $EEF_{x\%}$ can be obtained as follows:

Code Example 4.3

1. $EEF_{x\%}$ =raytrace_data.eef(Percentage=x,Theta_Reflectivity=theta_rf, Reflectivity_p=rfp,Reflectivity_h=rfh,IsReflectivityCon="yes")

4.2.1 Parameters

- **Percentage:** In $EEF_{x\%}$, the value of x represents a percentage of the total energy.
- **Theta_Reflectivity, Reflectivity_p, Reflectivity_h, and IsReflectivityCon:** These parameters can be defined in the same way as they were in Section 4.1.1.

4.2.2 Output

Executing Code Example 4.3, by default, generates a plot for $EEF_{x\%}$ vs. off-axis angle θ. Users can extract the $EEF_{x\%}$ data in the form of *EEF* and *theta*, as shown below:

Code Example 4.4

1. EEF, theta= $EEF_{x\%}$

Here, *EEF* is represented in arc-seconds, and *theta* is represented in arc-minutes.

4.3 Effective Area

The Effective area of the telescope can be obtained as follows:

Code Example 4.5

1. Effective_Area =raytrace_data.effa(Theta_Reflectivity=theta_rf,Reflectivity_p=rfp, Reflectivity_h=rfh,IsReflectivityCon="yes")

4.3.1 Parameters

- **Theta_Reflectivity, Reflectivity_p, Reflectivity_h, and IsReflectivityCon:** These parameters can be defined in the same way as they were in Section 4.1.1.

4.3.2 Output

Executing Code Example 4.5, by default, generates a plot for effective area vs. off-axis angle θ. Users can extract the effective area data in the form of *effa* and *theta*, as shown below:

Code Example 4.6

```
1. effa, theta= Effective_Area
```

Here, *effa* is represented in cm^2, and *theta* is represented in arc-minutes.

4.4 Vignetting Factor

The Vignetting Factor of the telescope can be obtained as follows:

Code Example 4.7

```
1. Vignetting_Factor =raytrace_data.vf(Theta_Reflectivity=theta_rf,Reflectivity_p=rfp,
   Reflectivity_h=rfh,IsReflectivityCon="yes")
```

4.4.1 Parameters

- **Theta_Reflectivity, Reflectivity_p, Reflectivity_h, and IsReflectivityCon:** These parameters can be defined in the same way as they were in Section 4.1.1.

4.4.2 Output

Executing Code Example 5.1, by default, generates a plot for Vignetting Factor vs. off-axis angle θ. Users can extract the vignetting factor data in the form of *vf* and *theta*, as shown below:

Code Example 4.8

```
1. vf, theta= Vignetting_Factor
```

Here, *vf* has no unit, and *theta* is represented in arc-minutes.

4.5 Detector Shape

A curved detector can be employed to enhance off-axis angular resolution compared to a flat detector. The optimal detector curvature for achieving the best angular resolution for both on-axis and off-axis sources can be determined as follows:

Code Example 4.9

```
1. Detector_Shape =raytrace_data.det_shape(Percentage=x,Theta_Reflectivity=theta_rf,
   Reflectivity_p=rfp,Reflectivity_h=rfh,IsReflectivityCon="yes")
```

4.5.1 Parameters

- **Percentage:** In the context of estimating angular resolution, 'percentage' is defined as the proportion of total energy enclosed within the selected region of the PSF.
- **Theta_Reflectivity, Reflectivity_p, Reflectivity_h, and IsReflectivityCon:** These parameters can be defined in the same way as they were in Section 4.1.1.

4.5.2 Output

Executing Code Example 4.9, by default, generates a plot for detector shape in XY plane. Users can extract the detector shape data in the form of X and Y, as shown below:

Code Example 4.10

```
1. [X, Y], [theta, EEF]= Detector_Shape
```

Here, X and Y represent the curvature of the detector, both measured in millimeters. The variable *theta* denotes the angles at which the curvature is estimated, given in arc-minutes. The EEF is an Energy-Encircled Fraction radius that is optimized for the curved detector, expressed in arc-seconds and corresponding to the same percentage value as the one provided as input in Code Example 4.9.

4.6 3D visualization

The 3D visualization of ray-traced data can be obtained as follows:

Code Example 4.11

```
1. 3D_visualization =raytrace_data.gui(Theta0=theta,NumRays=N)
```

4.6.1 Parameters

- **Theta0:** The source position, θ, in degrees for ray trace visualization. If the provided θ does not match any value from the list of input θ values provided in code example 3.1, then the closest value will be considered.
- **NumRays:** Maximum number of rays for visualization.

4.6.2 Output

Executing Code Example 4.11, by default, generates a 3D plot for ray-traced data.

5 Example

5.1 Example-1

Code Example 5.1

```
1 from raytrace import rtrace
2 import darpanx as drp
3 import matplotlib.pyplot as plt
4 from ErrorFunctions import*
5 import numpy as np
6
7 ## Telescope parameters
8 theta=np.linspace(0,0.5,10) # Source location theta in degrees
```

```
r0=np.arange(150,300,10) # Radius of each shell
x0=10000*np.ones_like(r0) # Focal length of each shell in mm
lp=400*np.ones_like(r0) # Length of parabola of each shell in mm
lh=400*np.ones_like(r0) # Length of hyperbola of each shell in mm
st=np.zeros_like(r0)+1 # Thickness of each shell in mm

if __name__ == '__main__':

    ## Ray-Trace
    raytrace_data=rtrace(Radius=r0,Focallength=x0,Lengthpar=lp,
                         Lengthhyp=lh, ShellThickness=st,Theta
                         =theta,Raydensity=500, NumCore=5,
                         SurfaceType="wo")

    ## Reflectivity Calculation using DarpanX.................
    Energy=[4] # Energy of X-rays
    m=drp.Multilayer(MultilayerType="SingleLayer",SubstrateMaterial
                     ="SiO2", LayerMaterial=["Ir"],Period=300)
    theta_rf=np.arange(start=0,stop=5,step=0.01)
    m.get_optical_func(Theta=theta_rf,Energy=Energy,AllOpticalFun
                       ="yes")
    rf=m.Ra
    ##.....................................................

    ## Post Processing
    raytrace_data.psf(Thetaforpsf= 0.2,Pixel_size=20,
                      Theta_Reflectivity=theta_rf,Reflectivity_p
                      =rf,Reflectivity_h=rf)

    raytrace_data.eef(Percentage=50,Theta_Reflectivity=theta_rf,
                      Reflectivity_p=rf, Reflectivity_h=rf)

    raytrace_data.effa(Theta_Reflectivity=theta_rf,Reflectivity_p
                       =rf, Reflectivity_h=rf)

    raytrace_data.vf(Theta_Reflectivity=theta_rf,Reflectivity_p=rf
                     ,Reflectivity_h=rf)

    raytrace_data.det_shape(Percentage=60,Theta_Reflectivity=
                            theta_rf ,Reflectivity_p=rf,
                            Reflectivity_h=rf)

    raytrace_data.gui(Theta0=0.4,NumRays=20)
    plt.show()
```

References

[1] http:Ref-to-DarsakX-paper