DarsakX: A Python Package for Designing and Analyzing Imaging Performance of X-ray Telescopes

User Guide

Version 0.1

Neeraj K. Tiwari

March 7, 2025

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.

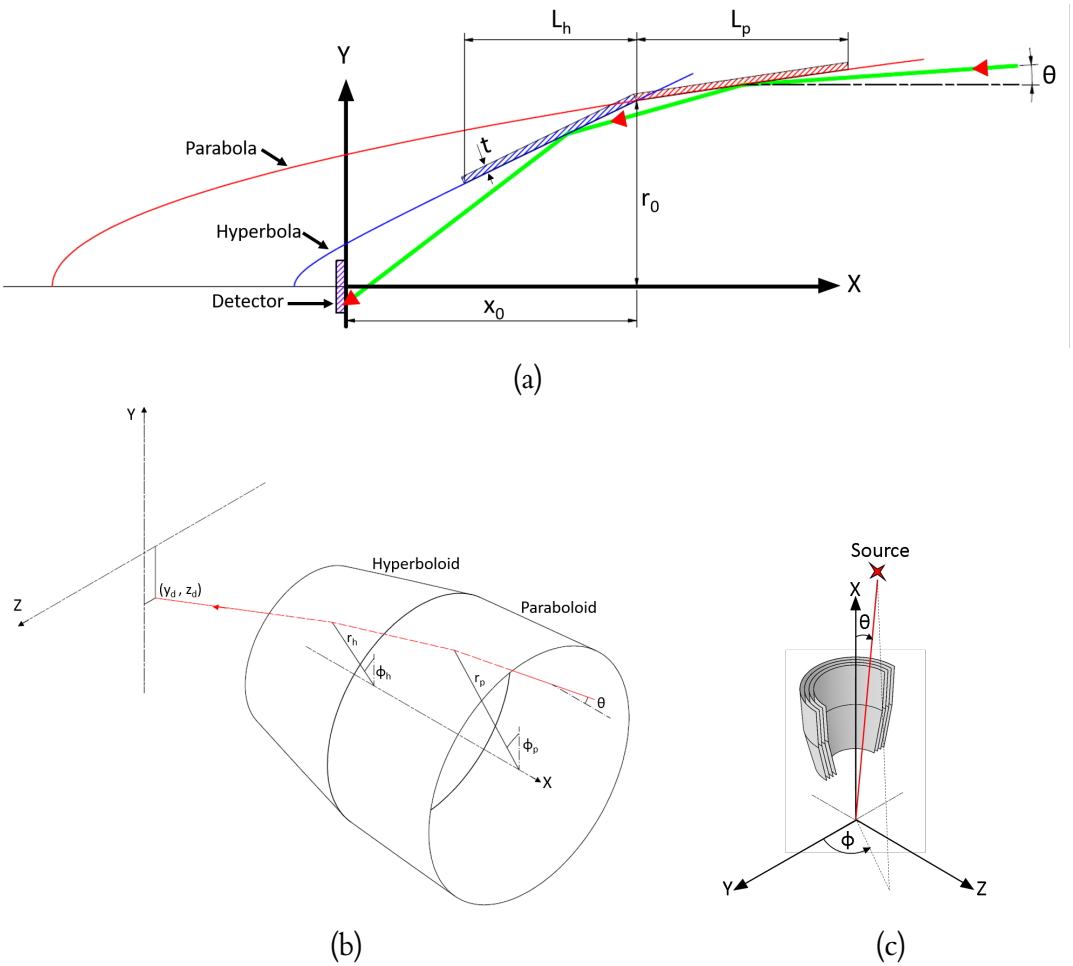


Figure 1: (a) A 2D schematic diagram of WO1 with various geometrical parameters. (b) A ray trace through the full shell of WO1. (c) Definition of the source location with respect to the cross-sectional view of the multi-shell WO1.

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.

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 $(x_0, r_0, L_h, L_p, t, \xi)$.

A detailed methodology used to develop DarsakX is provided in the published paper on DarsakX[1].

2 DarsakX Installation

DarsakX is provided in the Python 3 module. Other Python modules required for DarsakX are as follows:

- numpy
- scipy
- matplotlib
- tabulate

DarsakX requires X-ray reflectivity for the mirror as a function of the incident angle and energy of the ray as input parameters. It is recommended to utilize the DapanX Python module for estimating the reflectivity of the mirror, as it has been used in all examples within this manual. However, users have the flexibility to estimate reflectivity using any other method, provided that the input parameters are provided in a specified format, as described later in this document.

The DarsakX can be downloaded from the GitHub link[2]. After downloading, navigate to the *darsakx-main* directory, which contains the *setup.py* file, and execute the following command to install Darsak on your system.

Code Example 2.1

```
sudo python3 setup.py install
```

3 DarsakX Usage: Geometrical Ray Tracing

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.	$[n_{rh_x}, n_{rh_y}, n_{rh_z}]$
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,ParallelProcessingFor=None,SurfaceType="wo",Error="no",
   Approx="no",Gp=None,dGp=None,Gh=None,dGh=None)
```

3.1 darsakx.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
  The focal lengths of each shell can be the same or different.

- **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 can be used to distribute the job to different cores, either for multiple radii (which define the number of shells) or multiple thetas (which define the number of source locations). While defining the NumCore, ParallelProcessingFor must also be defined, either as *"shell"* or as *"theta"*.

- **ParallelProcessingFor:** This must be defined as *"theta"* if parallel processing is for theta values, or it needs to be defined as "*shell* " if parallel processing is for different shells in a multishell telescope. Note that when defining ParallelProcessingFor, NumCore must also be defined as mentioned above.

- **SurfaceType:** Surface type defines the type of surface, whether it is Wolter-1 optics (*wo*), Conical approximation (*co*), or user-defined optics (*uo*) 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, "*co*" for conical approximation optics, and "*uo*" for user-defined optics, all as strings.

- **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 whenever *Error="yes"* is specified. When SurfaceType="*uo*", this parameter is not required.

- **Approx:** This parameter defines the computational method, specifying whether it is an approximate method or a method without approximation, for tracing rays in a shell that has figure error. It is an optional parameter, and the default method is without approximation. The user can opt for the approximate method by setting *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. When SurfaceType="*uo*", this parameter is not required.

- **Gp:** This parameter defines the figure error present in the paraboloid section or primary section for each shell, but only when the SurfaceType is defined as either *"wo"* or *"co"* and must be provided only when the parameter *Error="yes"* is set. When the surface type is defined as *"uo"*, this parameter defines the surface of the primary mirror. Note that for *"co"* and *"wo"*, this parameter add as surface figure error to the base surface, whereas in the case of *"uo"* this parameter defines the base surface itself. *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, but only when the SurfaceType is defined as either *"wo"* or *"co"* and must be provided only when the parameter *Error="yes"* is set. When the surface type is defined as *"uo"*, this parameter defines derivative of the surface profile of the primary mirror. Note that for *"co"* and *"wo"*, this parameter adds as a surface figure error derivative to the base surface derivative, whereas in the case of *"uo"* this parameter defines the derivative of the base surface itself. *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, but only when the SurfaceType is defined as either *"wo"* or *"co"* and must be provided only when the parameter *Error="yes"* is set. When the surface type is defined as *"uo"*, this parameter defines the surface of the secondary mirror. Note that for *"co"* and *"wo"*, this parameter add as surface figure error to the base surface,

whereas in the case of *"uo"* this parameter defines the base surface itself. *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, but only when the SurfaceType is defined as either *"wo"* or *"co"* and must be provided only when the parameter *Error="yes"* is set. When the surface type is defined as *"uo"*, this parameter defines derivative of the surface profile of the secondary mirror. Note that for *"co"* and *"wo"*, this parameter adds as a surface figure error derivative to the base surface derivative, whereas in the case of *"uo"* this parameter defines the derivative of the base surface itself. *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 darsakx.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 Usage: Post Processing

This section explains how DarsakX utilizes the geometrical 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 of 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",Plot="yes")
```

4.1.1 darsakx.rtrace.psf() 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:

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

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

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

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

- **Plot:** This is an optional parameter with the default value set to "yes". By default, a plot of the output result will be generated unless it is set to "no".

4.1.2 darsakx.rtrace.psf() Output

Executing Code Example 4.1, it will, by default, generate a PSF plot unless *Plot*="no" is specified. 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, $EEF_{\mu\%}$, defines the size of the central region of the PSF that contains $\mu\%$ of the total energy of the entire PSF. The $EEF_{\mu\%}$ can be obtained as follows:

Code Example 4.3

```
1. EEF_μ% =raytrace_data.eef(Percentage=μ,Theta_Reflectivity=theta_rf,
   Reflectivity_p=rfp,Reflectivity_h=rfh,IsReflectivityCon="yes",Plot="yes")
```

4.2.1 darsakx.rtrace.eef() Parameters

- **Percentage:** In $EEF_{\mu\%}$, the value of μ 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.
- **Plot:** This is an optional parameter with the default value set to "yes". By default, a plot of the output result will be generated unless it is set to "no".

4.2.2 darsakx.rtrace.eef() Output

Executing Code Example 4.3, by default, generates a plot for $EEF_{\mu\%}$ vs. off-axis angle θ unless *Plot*="no" is specified. Users can extract the $EEF_{\mu\%}$ data in the form of *EEF* and *theta*, as shown below:

Code Example 4.4

```
1. EEF, theta= EEF_μ%
```

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",Plot="yes")
```

4.3.1 darsakx.rtrace.effa()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.
- **Plot:** This is an optional parameter with the default value set to "yes". By default, a plot of the output result will be generated unless it is set to "no".

4.3.2 darsakx.rtrace.effa() Output

Executing Code Example 4.5, by default, generates a plot for effective area vs. off-axis angle θ unless *Plot*="no" is specified. 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",Plot="yes")
```

4.4.1 darsakx.rtrace.vf() 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.
- **Plot:** This is an optional parameter with the default value set to "yes". By default, a plot of the output result will be generated unless it is set to "no".

4.4.2 darsakx.rtrace.vf() Output

Executing Code Example 4.7, by default, generates a plot for Vignetting Factor vs. off-axis angle θ unless *Plot*="no" is specified. 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",Plot="yes")
```

4.5.1 darsakx.rtrace.det_shap() 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.
- **Plot:** This is an optional parameter with the default value set to "yes". By default, a plot of the output result will be generated unless it is set to "no".

4.5.2 darsakx.rtrace.det_shap() Output

Executing Code Example 4.9 by default generates a plot for the detector shape in the XY plane and a comparison of the variation of EEF with θ for flat and curved detectors unless *Plot*="no" is specified. 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 darsakx.rtrace.gui() 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 per shell to be visualized.

Note: Users can press 'x', 'y', and 'z' on the keyboard to obtain the projected view of the 'YZ', 'ZX', and 'XY' planes, respectively.

4.6.2 darsakx.rtrace.gui() Output

Executing Code Example 5.1, by default, generates a 3D plot for ray-traced data.

5 Examples

Few examples are provided in this section to demonstrate DarsakX functionality.

5.1 Example-1: Performance Parameters

In this example, a multi-shell telescope is defined with 15 shells, spaced uniformly between radii of 150 mm and 290 mm with a gap of 10 mm between each shell. All shells have an identical thickness of 1 mm. The length of the primary and secondary mirrors is equal and has a value of 400 mm for each shell. The focal length for each shell is 10 m. The optical configuration of each shell is *'wo'* type without any figure error. The *'xi' or* ξ value is 1 for each shell by default.

First, the geometrical rays have been traced using the `darsakx.rtrace()` module, where the ray has been traced for 10 values of θ ranging from 0 to 0.5 deg uniformly. Ray density is considered as 2000 $rays/cm^2$. The ray tracing job for 10 values of θ is distributed over 5 cores using parallel processing.

After the geometrical ray trace, mirror reflectivity property for both mirrors in each shell is defined as a single layer of *Iridium* on a substrate of SiO_2 using the `darpanx` Python package. The X-ray reflectivity for these mirrors is calculated for the X-ray energy of 4 keV and at various incident angles of 0 to 5 deg with a step size of 0.01 deg. This reflectivity information is provided in various post-processing modules of DarsakX to estimate various performance parameters such as PSF, EEF, VF, and det_shape.

The PSF is estimated at $\theta = 0.2$ deg with a detector pixel size of 20 microns. The reflectivity is considered the same for both mirrors in each shell. The EEF is calculated with an energy percentage of 50%. The effective area and vignetting factor are also estimated. The optimum focal surface is also estimated for an energy percentage of 60%, and 5 random rays per shell are demonstrated for 3D visualization for a $\theta = 0.2$ deg. The terminal output of geometrical ray tracing in the form of a table is shown in Figure 2. The plots obtained for these functions are shown below from Figure 3 to 8.

Code Example 5.1

```
from darsakx import rtrace
import darpanx as drp
import matplotlib.pyplot as plt
import numpy as np

## Telescope parameters
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=2000, 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=0.2,NumRays=5)
    plt.show()
```

```
Parallel processing: no.cores is 5
```

Telescope Parameters	
Surface type	Wolter-1
Ray Density[Rays/cm2]	2000
Figure-Error	no
Approximation-Method	no
Theta[deg]	0.0-0.5
Detector-Position[mm]	0

Telescope Geometrical Parameters

Sr.No	Radius[mm]	Lp[mm]	Lh[mm]	f[mm]	ξ	t[mm]
1	150	400	400	10000	1	1
2	160	400	400	10000	1	1
3	170	400	400	10000	1	1
4	180	400	400	10000	1	1
5	190	400	400	10000	1	1
6	200	400	400	10000	1	1
7	210	400	400	10000	1	1
8	220	400	400	10000	1	1
9	230	400	400	10000	1	1
10	240	400	400	10000	1	1
11	250	400	400	10000	1	1
12	260	400	400	10000	1	1
13	270	400	400	10000	1	1
14	280	400	400	10000	1	1
15	290	400	400	10000	1	1

```
Run Time[s] =  24.238
Ray Trace Completed!
```

Figure 2

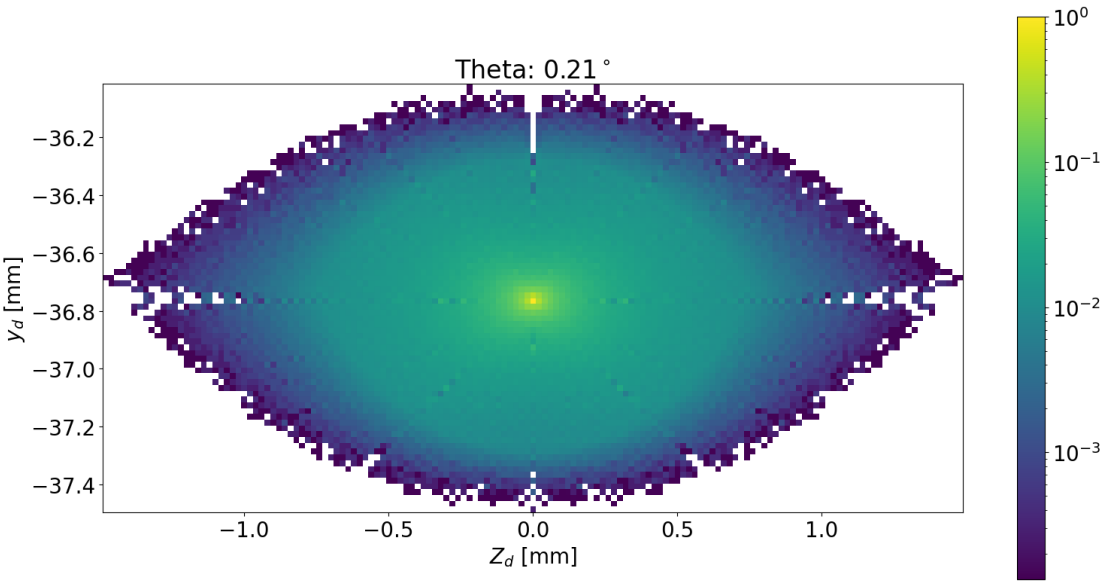


Figure 3

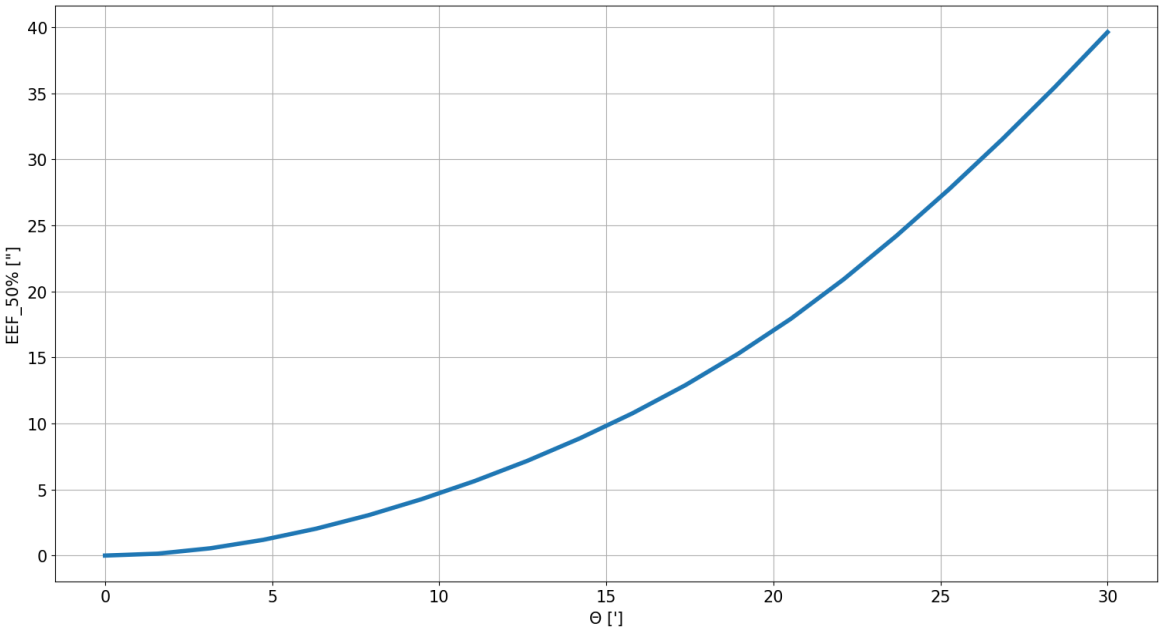


Figure 4

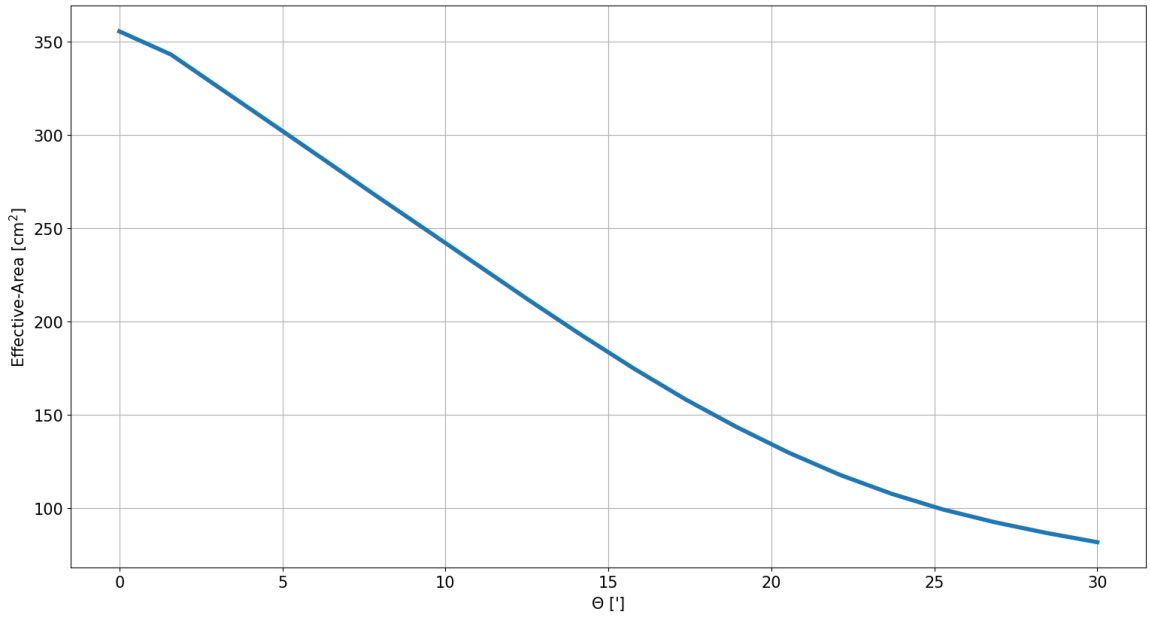


Figure 5

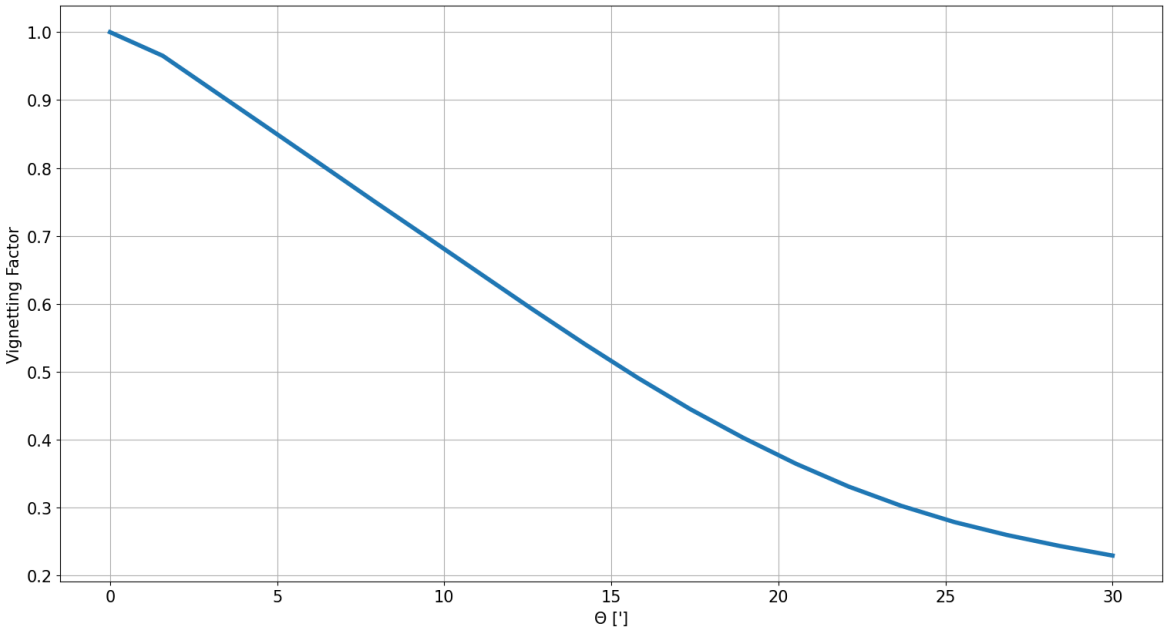


Figure 6

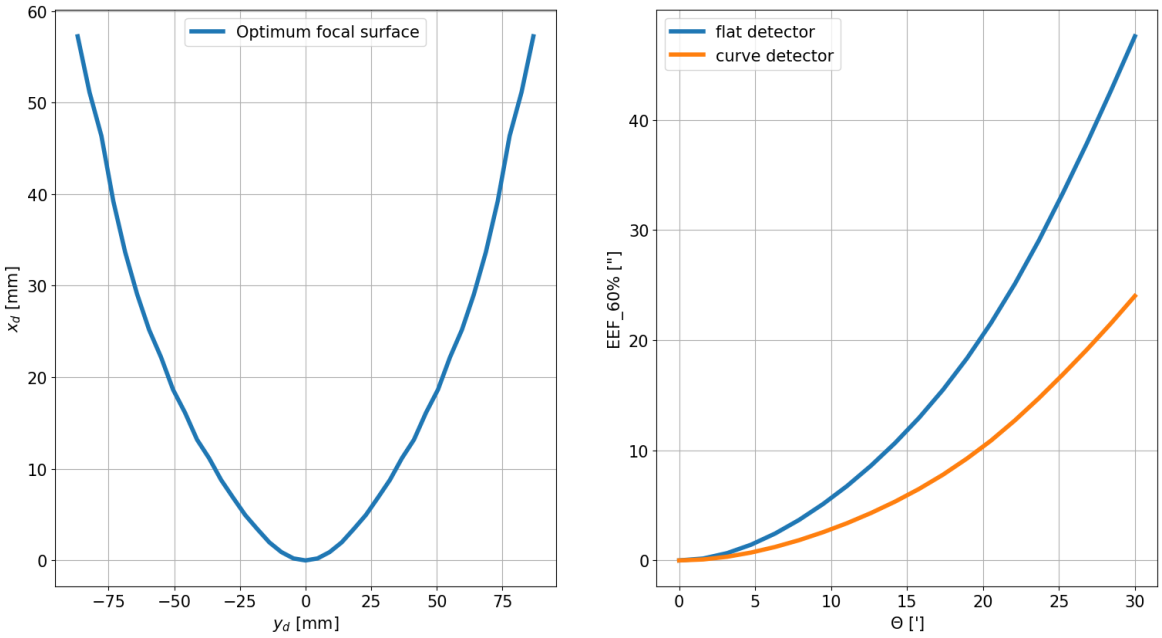


Figure 7

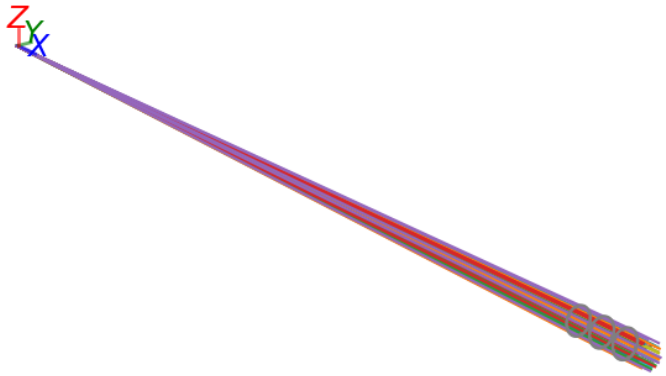


Figure 8

The effective area variation with energy can be obtained by running the post-processing function darsakx.rtrace.effa() multiple times in a loop for various energy values. However, to calculate the mirror reflectivity variation with θ for multiple energies, a module from DarpanX, darpanx.Multilayer.get_optical_func(), also needs to be run multiple times in a loop. In code example 5.2, the effective area variation with energy is obtained for the same telescope represented by code example 5.1. In this example, rays have been traced only for four values of θ between 0 and 0.3 degrees without using any parallel processing. Ray density has been set to 200 *rays*/cm^2. Effective area is calculated for 500 energy values uniformly ranging from 1 to 20 keV.

Code Example 5.2

```
from darsakx import rtrace
import darpanx as drp
import matplotlib.pyplot as plt
import numpy as np

## Telescope parameters
theta=np.array([0,0.1,0.2,0.3]) # Source location theta in deg
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


## Ray-Trace
raytrace_data=rtrace(Radius=r0,Focallength=x0,Lengthpar=lp,
                     Lengthhyp=lh,ShellThickness=st,Theta=theta,
                     Raydensity=200, SurfaceType="wo")

## Effective-Area vs Energy for multiple theta
```

```
Energy=np.linspace(1,20,500) # in keV
effa=np.zeros((len(theta),len(Energy)))
m=drp.Multilayer(MultilayerType="SingleLayer",SubstrateMaterial
                 ="SiO2", LayerMaterial=["Ir"],Period=300,
                 ShowPar='no')
for i in range(len(Energy)):
   theta_rf=np.arange(start=0,stop=5,step=0.01)
   m.get_optical_func(Theta=theta_rf,Energy=[Energy[i]],
                      AllOpticalFun ="yes")
   rf=m.Ra
   effa[:,i],_=raytrace_data.effa(Theta_Reflectivity=theta_rf,
                                  Reflectivity_p=rf,
                                  Reflectivity_h=rf,Plot="no")

## Plot for Effective-Area vs Energy
plt.rcParams.update({'font.size':20})
for i in range(len(theta)): plt.plot(Energy,effa[i,:],
               linewidth=4,label="$\Theta=$"+str(theta[i])+
               "$^\circ$")
plt.xlabel('Energy[keV]')
plt.ylabel('Effective-Area[$cm^2$]')
plt.legend(loc='upper right')
plt.grid(True)
plt.show()
```

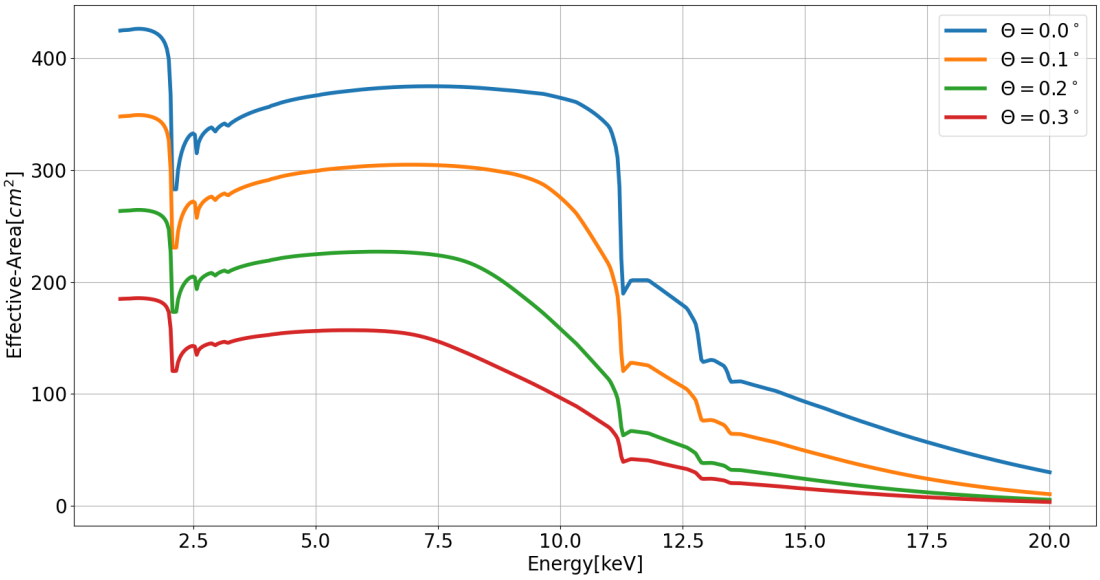


Figure 9

For this telescope, the variation of *EEF* with off-axis angle θ, *PSF* and *det_shape* can be produced for the optical design when all the shells are considered as *'co'* type instead of *'wo'* by executing code example 5.3. These plots are shown in Figure 10 to 12.

Code Example 5.3

```
from darsakx import rtrace
import darpanx as drp
import matplotlib.pyplot as plt
import numpy as np

## Telescope parameters
theta=np.linspace(0,0.5,20) # 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=2000, NumCore=5,
                         SurfaceType="co")

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

    plt.show()
```

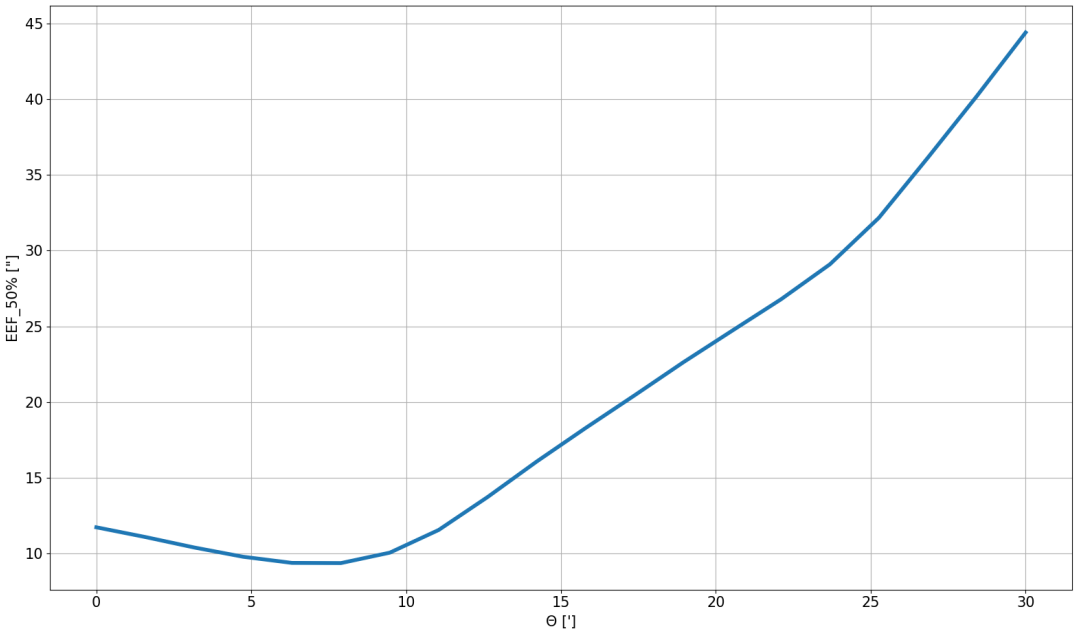


Figure 10

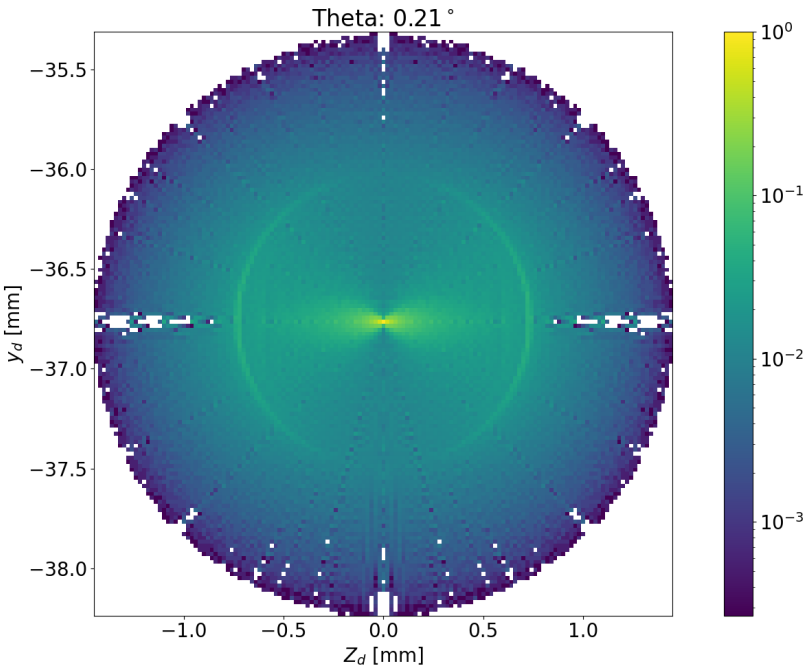


Figure 11

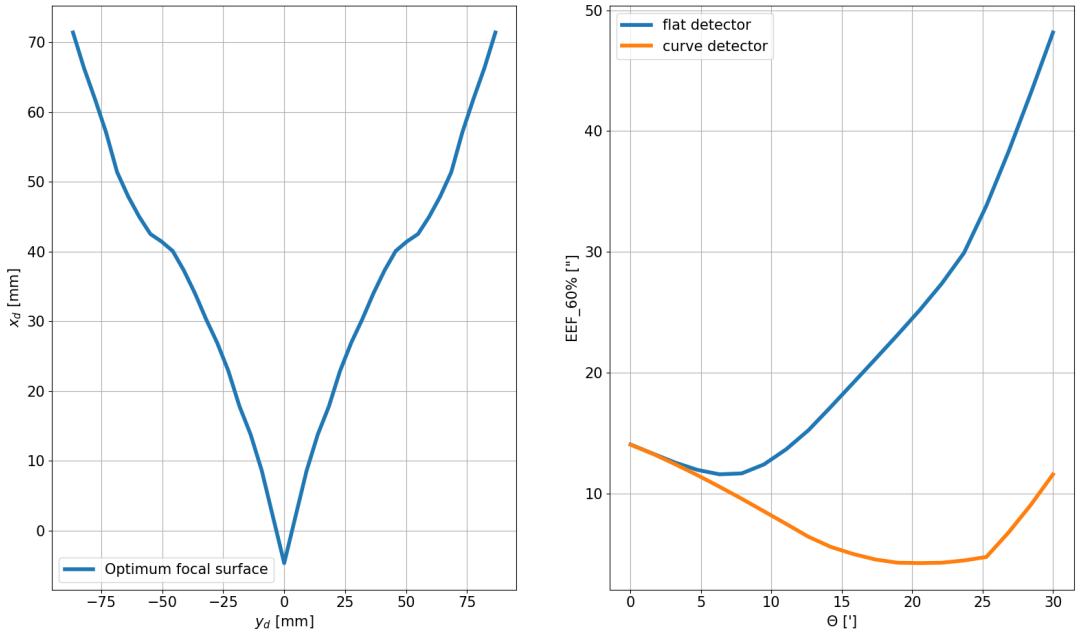


Figure 12

5.2 Example-2: Effect of Figure Error

In this example, we consider a telescope consisting of 4 shells with radii of 30 cm, 40 cm, 50 cm, and 60 cm, each with a focal length of 10 m. The length of the primary and secondary mirrors is identical in each shell, with a value of 50 cm. The thickness of each shell is considered to be 5 mm.

The X-ray mirrors are coated with a single layer of Au on SiO_2 substrate. The geometrical shape of the mirrors is a combination of a base surface as WO_1 with identical figure error in each shell. The shape of the figure is a parabola in both the primary and secondary mirrors of each shell. The figure has a peak value of 4 μm in the primary mirror and 8 μm in the secondary mirror, with zero values at the edges of the mirrors.

In the following code example (see 5.4), we define the geometrical properties of the telescope from lines 8 to 14. Next, from lines 19 to 27, we calculate the reflectivity of the mirror as a function of the incident angle at an energy of 5 keV. Lines 31 to 32 are dedicated to plotting the figure errors and their derivatives, which are considered in this telescope. The Python file ErrorFunctions.py, containing the figure error and their derivative functions, is referred to in code example 5.5.

Moving forward to lines 46 through 63, we trace the rays for three different cases. The first case involves no figure error in any shell of the telescope. In the second case, figure errors are considered, but rays are traced using an approximate analytical method. In the third case, figure errors are considered, but rays are traced using the exact method without approximation. In the first two cases, rays are traced for 20 sets of θ values with a ray density of 1000 rays/cm^2. However, in the third case, rays are traced for only 10 sets of θ values with a ray density of 200 rays/cm^2.

Continuing from lines 68 to 90, we calculate the EEF variation with θ and the PSF at $\theta = 3$ arcmin for all three cases. The results obtained in the form of plots are shown in Figures 13 to 15. Figure 13 displays the figure errors and their derivatives used in this

telescope. Figure 14 illustrates the EEF variation with θ for all three cases. Figure 15a depicts the PSF produced without figure error. Figures 15b and 15c show the PSF produced with figure error, using the approximate and exact methods, respectively.

Code Example 5.4

```
from darsakx import rtrace
import darpanx as drp
import matplotlib.pyplot as plt
from ErrorFunctions import*
import numpy as np


## Telescope parameters
theta=np.linspace(0, 0.4, 20) # Source location theta in degrees
r0=np.array([300, 400, 500, 600]) # Radius of each shell
x0=10000 * np.ones_like(r0) # Focal length of each shell in mm
lp=500 * np.ones_like(r0) # Length of parabola of each shell in mm
lh=500 * np.ones_like(r0) # Length of hyperbola of each shell in mm
st = np.zeros_like(r0) + 5 # Thickness of each shell in mm

if __name__ == '__main__':

    ## Reflectivity Calculation using DarpanX.................
    Energy = [5]  # Energy of X-rays
    m = drp.Multilayer(MultilayerType="SingleLayer",
        SubstrateMaterial="SiO2",LayerMaterial=["Au"],
        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
    ##.....................................................

    ## Plot Figure error
    xh=np.linspace(x0[0]-lh[0],x0[0],100)
    xp=np.linspace(x0[0],x0[0]+lp[0],100)
    plt.rcParams.update({'font.size': 20})
    fig, (ax1, ax2) = plt.subplots(2,sharex=True,)
    ax1.plot(xh,Gh_fun(xh),xp,Gp_fun(xp), linewidth=4)
    ax2.plot(xh,dGh_fun(xh),xp,dGp_fun(xp),linewidth=4)
    ax1.grid(True); ax2.grid(True)
    ax1.set_ylabel('$G_p(x), G_h(x)$')
    ax2.set_ylabel('$dG_p(x), dG_h(x)$')
    ax1.legend(['$G_h(x)$', '$G_p(x)$'])
    ax2.legend(['$dG_h(x)$', '$dG_p(x)$'])
    ax2.set_xlabel('X')
```

```
43      ##.....................................................
44
45      ## Ray-Trace
46      rf = [rf] * len(r0); theta_rf = [theta_rf] * len(r0)
47      Gp = [Gp_fun] * len(r0); dGp = [dGp_fun] * len(r0)
48      Gh = [Gh_fun] * len(r0); dGh = [dGh_fun] * len(r0)
49
50      raytrace_data = rtrace(Radius=r0, Focallength=x0, Lengthpar
51                      =lp, Lengthhyp=lh, ShellThickness=st,
52                      Theta=theta, Raydensity=1000, NumCore=5)
53
54      raytrace_data_approx= rtrace(Radius=r0, Focallength=x0,
55              Lengthpar=lp, Lengthhyp=lh, ShellThickness=st,
56              Theta=theta, Raydensity=1000, NumCore=5, Error
57              ='yes', Approx='yes',Gp=Gp ,dGp=dGp,Gh=Gh,dGh=dGh)
58
59      theta = np.linspace(0, 0.4, 10)
60      raytrace_data_exact = rtrace(Radius=r0, Focallength=x0,
61              Lengthpar=lp, Lengthhyp=lh, ShellThickness=st,
62              Theta=theta, Raydensity=200, NumCore=5, Error
63              ='yes', Approx='no',Gp=Gp ,dGp=dGp,Gh=Gh,dGh=dGh)
64      ##.....................................................
65
66
67      ## Post Processing
68      raytrace_data.psf(Thetaforpsf=0.05, Pixel_size=20,
69            Theta_Reflectivity=theta_rf, Reflectivity_p
70            =rf, Reflectivity_h=rf)
71
72      EEF_withoutfigure, theta_withoutfigure=raytrace_data.eef(
73          Percentage=50, Theta_Reflectivity=theta_rf, Reflectivity_p
74          =rf, Reflectivity_h=rf,Plot="no")
75
76      raytrace_data_approx.psf(Thetaforpsf=0.05, Pixel_size=20,
77          Theta_Reflectivity=theta_rf, Reflectivity_p=rf,
78          Reflectivity_h=rf)
79
80      EEF_approx, theta_approx=raytrace_data_approx.eef(Percentage=50,
81          Theta_Reflectivity=theta_rf, Reflectivity_p=rf,
82          Reflectivity_h=rf,Plot="no")
83
84      raytrace_data_exact.psf(Thetaforpsf=0.05, Pixel_size=20,
85          Theta_Reflectivity=theta_rf, Reflectivity_p=rf,
86          Reflectivity_h=rf)
87
88      EEF_exact, theta_exact=raytrace_data_exact.eef(Percentage=50,
89          Theta_Reflectivity=theta_rf, Reflectivity_p=rf,
```

```
90          Reflectivity_h=rf,Plot="no")
91
92      plt.figure()
93      plt.plot( theta_withoutfigure,EEF_withoutfigure,linewidth=4,
94         label='Without Figure Error')
95      plt.plot( theta_approx,EEF_approx,linewidth=4, label=
96         'With Figure Error, approx')
97      plt.scatter( theta_exact,EEF_exact,s=150,label=
98         'With Figure Error, exact', marker="D", color='red')
99      plt.xlabel("$\Theta$ ['] ")
100     plt.ylabel('EEF_50% ["]')
101     plt.legend(loc='upper left', fontsize=20)
102     plt.show()
```

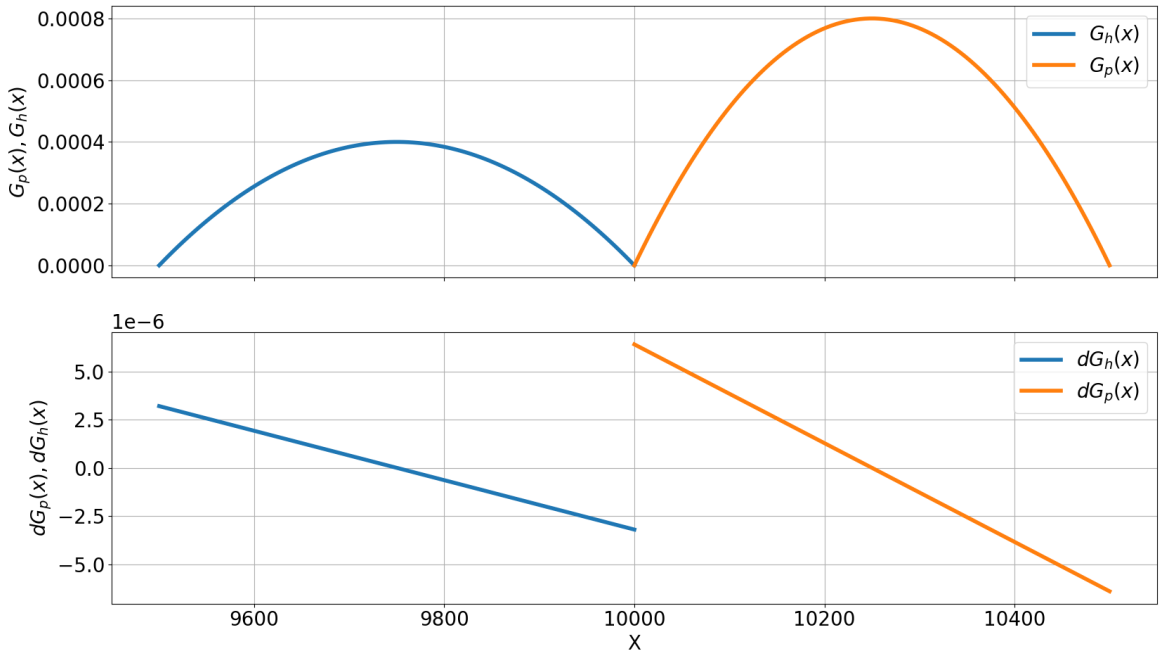


Figure 13

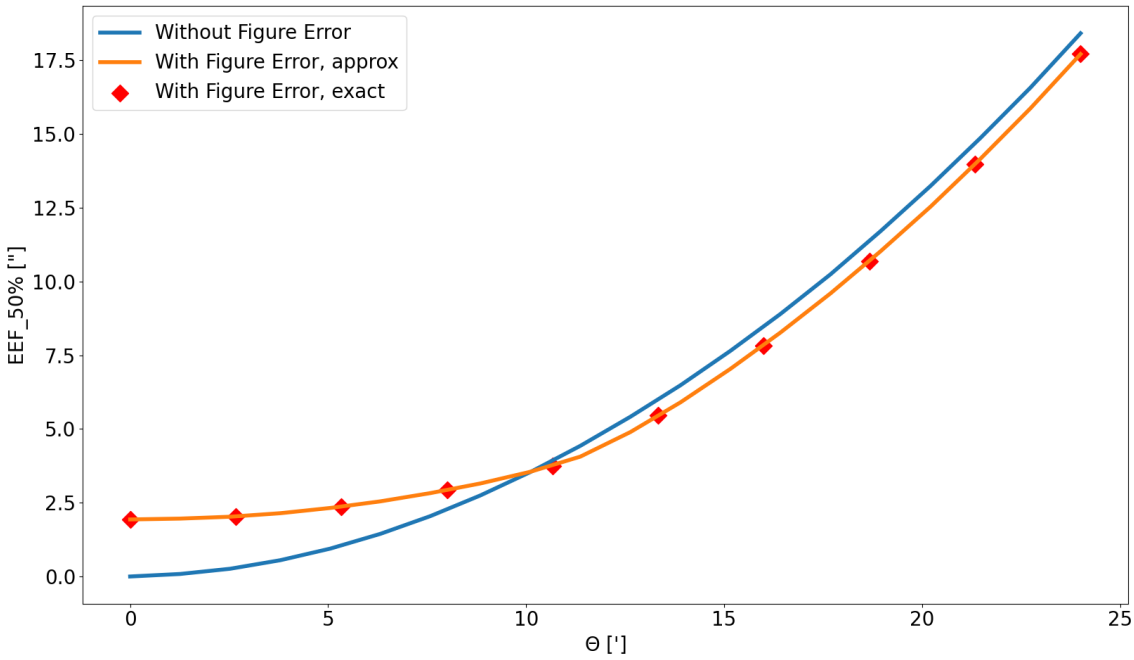
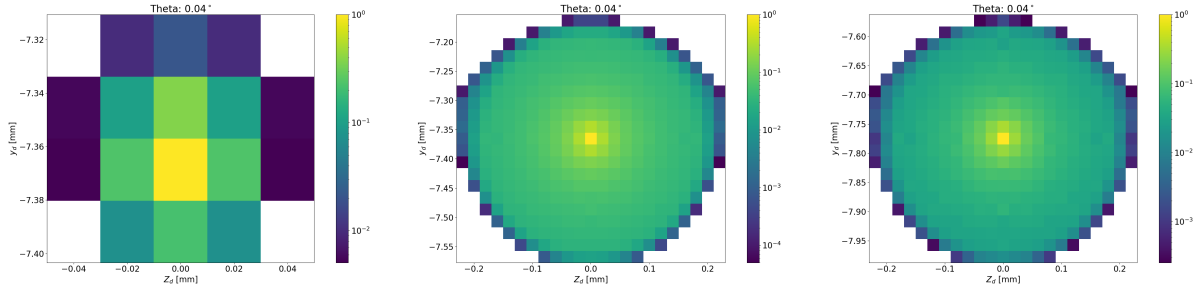


Figure 14



(a) (b) (c)

Figure 15

Code Example 5.5

```
lp=500
x0=10000

def Gp_fun(x): # Error in parabola
    Gp=-(1e-3)*(x-x0-lp)*(x-x0)*0.8*4/lp**2
    return Gp

def Gh_fun(x): # Error in hyperbola
    Gh=-(1e-3)*(x-x0+lp)*(x-x0)*0.4*4/lp**2
    return Gh

def dGp_fun(x): # Parabola error derivative
```

```
13      d_Gp=-(1e-3)*(2*x-2*x0-lp)*0.8*4/lp**2
14      return d_Gp
15
16  def dGh_fun(x): # Hyperbola error derivative
17      d_Gh=-(1e-3)*(2*x-2*x0+lp)*0.4*4/lp**2
18      return d_Gh
```

5.3 Example-3: Optimizing Wide-Field Telescope

References

[1] Neeraj K. Tiwari. Darsakx paper. http://ref-to-darsakx-paper, 2024.

[2] Neeraj K. Tiwari. Darsakx package. https://github.com/xastprl/darsakx.git, 2024.