DarpanX: A Python Package for Modeling X-ray Reflectivity of Multilayer Mirrors

User Guide
Version 0.3

B. Mondal, S.V. Vadawale, N. P. S. Mithun, C.S. Vaishnava

July, 2020

Physical Research Laboratory
Ahmedabad, India.

Contents

1 Introduction

DarpanX is a Python package that provides functionality to compute reflectivity and other optical functions of multilayer/single-layer mirror for different incident energy and angles. It also allows users to fit the experimentally measured X-ray Reflectivity (XRR) of mirrors to derive the multilayer parameters.

Multilayer mirrors consist of a coating of alternate layers of high Z and low Z (Z indicates the atomic number)materials. Such coatings play an important role in enhancing the reflectivity of X-ray mirrors by allowing reflections at angles much larger than the critical angle of X-ray reflection for the given materials. Coating with an equal thickness of each bilayer (constant period multilayers) enhances the reflectivity at discrete energies, satisfying Bragg condition for the given thickness. However, by systematically varying the bilayer thickness in the multilayer stack (depth-graded multilayers), it is possible to design X-ray mirrors having enhanced reflectivity over a broad energy range. One of the most important applications of such a depth-graded multilayer mirror is to realize the hard X-ray telescopes for astronomical purposes. There are several applications of multilayer mirrors such as constant-period multi-layers for astronomical spectroscopy and multilayer mirrors to measure the polarization.

Laws of reflection/refraction at the boundary between two mediums for an electromagnetic wave are governed by the Fresnel's equations[1][2]. These equations relate the amplitudes of reflected and refracted waves with that of the incident wave. Calculation of optical functions in DarpanX is based on the Fresnel equations, modified for the finite surface roughness. The process for X-ray reflection from a multilayer mirror is governed by Bragg's law, the overall reflectivity of the mirror prominently depends on the surface micro-roughness, inter-layer roughness, interdiffusion at the interfaces, layer density, thickness, and uniformity of all layers. Precise knowledge of these parameters is essential for accurate modeling of X-ray reflectivity (XRR).

DarpanX includes a database of optical constants of 92 elements and some compounds, spanning the energy range from 0.001 keV to 433 keV. It can also compute the optical constants in the same energy range for any compound by using the atomic scattering factors of 92 elements that have been compiled from the database made available by NIST (National Institute of Standards and Technology)[3]. DarpanX can calculate the refractvie index for any given material by using corresponding atomic scattering factor data sets.

DarpanX can be used as a stand-alone package for the design of multilayer mirrors consisting of an arbitrary structure (i.e., any number of layers, materials, etc.) for various applications. In addition to this, it can also be used to measure the multilayer parameters by fitting the experimental XRR data along with the PyXspec (python version of XSPEC, An X-Ray Spectral Fitting Package). DarpanX is distributed under the GNU General Public License.

2 Installing DarpanX

DarpanX is designed in python object-oriented way and provided as a Python 3 module. It will run on any platform supported by python3.x. However, for the XRR data fitting, the platform should also support PyXspec.

Pre-requisites

It requires PyXspec (python version of XSPEC) installation with the same version of python 3; Other python modules required for DarpanX are:

- setuptools
- numpy
- scipy
- multiprocessing
- matplotlib
- tabulate

The DarpanX package can be downloaded from the Github link;
https : *//github.com/biswajitmb/DarpanX.git*
After downloading it, go to the *DarpanX* − *master* directory (where the file, '*setup.py*' located) under the downloaded directory:

```
cd DarpanX-master
```

To install DarpanX package system-wide:

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

Now the module will be installed in your system. Note that the optical constant database and some other functionality given within the DarpanX package will be accessible from the DarpanX directory, where you run the setup file, *setup.py*. Hence, do not remove that directory. You can check the installation by directly importing it in your python shell as:

```
import darpanx as drp
```

Note that, all the functionality of DarpanX has been checked with Python version 3.6 on Linux and Mac os.

3 Quick Start Guide

DarpanX offers two distinct functionalities: (i) computing X-ray reflectivity and other optical functions for different energies or angles to aid design of multi-layer systems and (ii) fit the experimental XRR data with a reflectivity model using PyXspec to obtain the parameters of a multi-layer system. Examples for both these are given here. Complete usage for these purposes are given in sections 4 and 5.

3.1 Computing optical function of single layer

As an example, here we use DarpanX to calculate the reflectivity, transmitivity, and absorbtance of a Pt single layer on SiO_2 substrate for incident energy of 8 keV over 0-5 degree angle range.

- For this, we first import the DarpanX module in the python shell/script and create an instance of $darpanx.Multilayer()$ class which defines the parameters of the multilayer structure:

```
import darpanx as drp
import numpy as np
m=drp.Multilayer(MultilayerType="SingleLayer",SubstrateMaterial="SiO2",LayerMaterial=["Pt
                                                    "],Period=80)
```

- Then, define the angle and energy values for which the optical functions are to be computed:

```
Energy=[8.0] # Incident beam energy in KeV
Theta=np.arange(start=0,stop=5,step=0.01) # Define Grazing angles in degree .
```

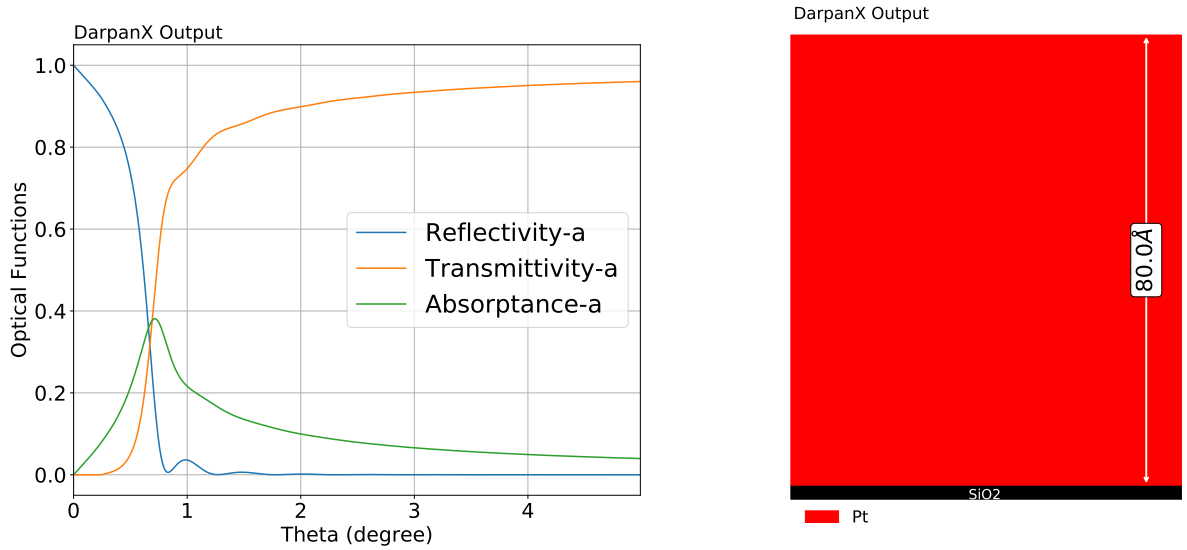
- To compute the optical functions, use the method *get_optical_func* of Multilayer class:

```
m.get_optical_func(Theta=Theta,Energy=Energy,AllOpticalFun ="yes")
```

- Now, the optical functions are computed and to plot the results and save the plots as pdf, use the plot method of Multilayer class:

```
m.plot(ylog="no", Comp=["Ra","Ta","Aa"], AllComp="oplot",OutFile="Example1_Single_Pt_80",
                                                    Scale="yes",Struc="yes")
```

Figure 1 shows the outputs where the the left panel shows the reflectivity, transmitivity, absorbtance and the right panel is the multilayer structure. These will appear as two windows. It will also be saved in the file *Single_Pt_80_Theta_RaTaAa.pdf* and *Single_Pt_80_structure.pdf*.



(a) (b)

Figure 1: (a) Optical functions. (b) Single Pt layer, shown in red and substrate is in black.

All the steps of this example are given in the script *Single_Pt_Layer.py*, available in the directory *DarpanX/examples/Example*1. Note that the output class attributes (Rs, Rp, Ra, Ts, Tp, Ts, As, Ap, Aa) are available as:

```
m.Rs # s-polarized component of reflectivity.
m.Rp # p-polarized component of reflectivity.
m.Ra # average reflectivity
m.Ta # average transmitivity
m.Aa # average absorbtance. etc.
```

3.2 Fitting XRR measurement of single layer sample

As a simple example , here we use the DarpanX to fit the XRR data of single Si layer on the top of SiO_2 substrate. This data is available within the DarpanX package, inside the directory: $DarpanX/examples/Example2\_XRR$.

As discussed later in Section 5, a user-defined configuration file is required in a prescribed format for the DarpanX model to fit the XRR data. After Initialising DarpanX and Pyxspec in your python shell, load this configuration file into DarpanX. After making the configuration file, users have to initialize the DarpanX and PyXspec in the python shell. After loading both DarpanX and PyXspec user have to load the configuration file in DarpanX, and connect DarpanX with PyXspec. Once it is connected, all the input parameters will show in the python shell. Then, the user has to to make a DarpanX model accessible from PyXspec. After that, the PyXspec fitting commands, as described in PyXspec manual (available in the the page-pyxspec) can be directly accessible from the python shell for the DarpanX model.

In a quick start, here we will load the interactive session for this sample by compiling the file $DarpanX/examples/Example2\_XRR/Example2\_XRR\_SingleSiModel.py$ (where all the required steps as mentioned above are done and saved) in the python shell and try to fit with the DarpanX model. It requires two additional files; (1) '*.drpnx*' file, from where DarpanX will get the information about the model (i.e., user input configuration file) and (2) the standard XSPEC '*.xcm*' file, which mainly contains the information of free parameters guess values as well as their lower and upper bounds for fitting. For the details step by step procedure of XRR data fitting, see the Section 5.

- Go to the directory $DarpanX/examples/Example2\_XRR/$ and initialized HEASoft package for XSPEC (e.g, heainit) in your environment and after that running python shell like ipython;

```
cd DarpanX/examples/Example2_XRR/ #Changing the current directory to Example2_XRR
heainit #Initializing HEASoft package for XSPEC
ipython #Opening the ipython
```

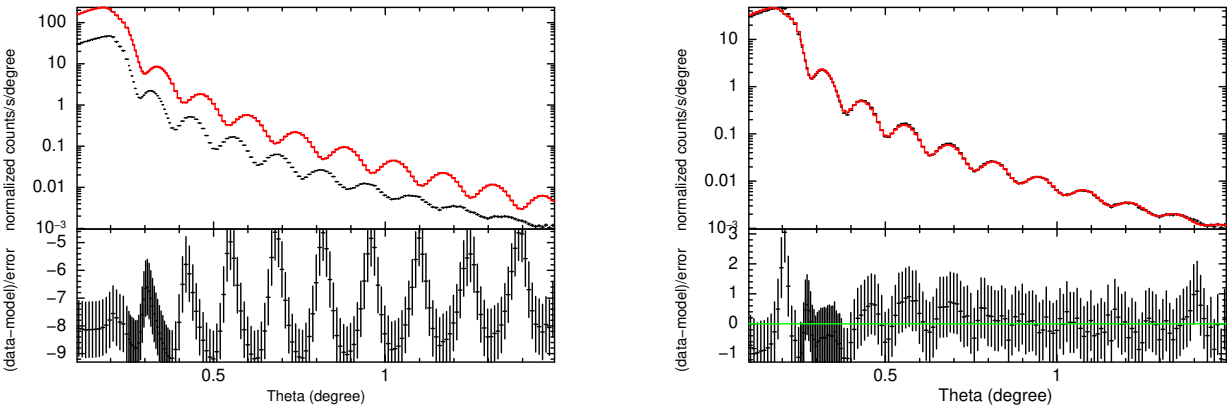
- Run the file $Example2\_XRR\_SingleSiModel.py$ inside ipython:

```
%run Example2_XRR_SingleSiModel.py
```

- To fit the XRR data with model and plot the final fitted data:

```
Fit.perform()
Plot("lda delc")
```

Figure-2 shows the fitted data with DarpanX model before and after fitting. In this example we considered the thickness,density, surface roughness (two interface, ambient/Si and Si/substrate) of the Si layer and the normalization factor of the model as a free variables.



(a) (b)

Figure 2: Data (black) with model (red) (a) Before fitting. (b) After fitting.

4 DarpanX usage: Multi-layer optical functions

For the design of multi-layer, the structure of the multi-layer system (i.e., define the number of layers, thickness, etc.) and their properties(like materials, their density, etc.) needs to be defined. After that, users have to choose which optical function/functions (reflectivity/transmitivity/Absorbtance); they want to compute for a given range of angle/energy.

The design of multilayer can be possible by using the class '$Multilayer$'. Users can use it in an interactive python shell or inside scripts. It is very useful to write a script designing the optical system (like X-ray optics) to satisfy a given condition by varying the different multilayer parameters/structures. the class, '$Multilayer$' has several methods, out of them, '$get\_optical\_func$' and '$plot$' is for the calculation of optical function for the given angle/energy and to plot the outputs.

4.1 Multilayer class

The '$Multilayer$' class can be initiated as $drp.Multilayer(parameter = value)$ and the parameters are explained in the subsections 4.1.1-4.1.4.

4.1.1 Define Multilayer structure

DarpanX algorithms are optimized for different types of multilayers. The types of multilayers can be set by using the keyword, "$MultilayerType$", and after defining the type of multilayer, users have to define the associated required parameters for that type of multilayer. $MultilayerType$ has several options, as described below:

1. $MultilayerType = 'UserDefinedML'$: Users can define their own mutilayer structure by giving the input parameters as follows:

(a) *Repetition:* Number of repeated structures in the system.

(b) *EachRepetitionLayerNum:* Number of layers present in each repetition.

(c) *EachRepetitionLayerThick:* Array of thickness values of the layer/layers present in each repetition.

(d) *TopLayerNum:* If there is an additional layer present on the top of the structure, then the number of additional layers present.

(e) *TopLayerThick:* Array of thickness values of added layer/layers on the top.

2. *MultilayerType = 'SingleLayer'* : To define a single layer system.

   (a) *Period:* Single layer thickness value.

3. *MultilayerType = 'BiLayer'* : To define a bi-layer system.

   (a) *Period:* Period (total thickness of high-Z and low-Z materials) of the bilayer system.

   (b) *Gamma:* Gamma (ratio of the thickness of high-Z to Period) of the bilayer system.

   (c) *Repetition:* Number of repetitions of the same bilayer combination.

4. *MultilayerType = 'ClusterGraded'* : To define a cluster-graded bi-layer system.

   (a) *NumStack:* Number of stacks to form the cluster-graded system.

   (b) *Period:* Array of period values of all the stacks. The length of the array should be equal to '$NumStack$'.

   (c) *Gamma:* Array of gamma values of all the stacks. Length of the array should be equal to '$NumStack$'.

   (d) *Repetition:* Number of repetitions of the same bilayer combination.

5. *MultilayerType = 'DepthGraded'* : To define a depth-graded system.

   (a) *D_max:* Maximum thickness value of the structure.

   (b) *D_min:* Minimum thickness value of the structure.

   (c) *GammaTop:* Gamma value of the top bilayer.

   (d) *Gamma:* Gamma value of rest(except top) of the system.

   (e) *C:* Power law index controlling the slope of the thickness profile from top to bottom.

6. *MultilayerType = 'UserDefined':* To define an arbitrary system.
   In this case, users have to provide an array of thickness profiles of their system.

   (a) *LayerNum:* Total number of layers.

   (b) *Z_Array:* Thickness profile array. It should be an 1D array consisting with number of elements equal to $LayerNum$.

4.1.2 Specify Material properties

The material properties of the layers, as well as the ambient and substrate, is defined by the following keywords:

1. *LayerMaterial*: Array of material/materials from top to bottom, used to make the layer structure, such as for a bilayer of W/Si with an additional top WO_3 layer, it should be "WO3","W","Si"].
2. *AmbientMaterial*: (default='Vacuum') The material of the ambient medium.
3. *SubstrateMaterial*: (default='Vacuum') The material of the substrate.
4. *AmbientDensity*: The density of the ambient medium. If not provided it will take from the header of the SF-NK data file (for details see the section-6) of AmbientMaterial.
5. *SubstrateDensity*: The density of the substrate. If not provided it will take from the header of the SF-NK (for details see the section-6) data file of SubstrateMaterial.
6. *DensityCorrection*: (default='no') If 'yes', then the refractive index of the layer materials will be calculated for given density values, and the optical functions will calculate by using new refractive index values. Note that if *DensityCorrection* = 'no', then refractive indices will be calculate for the density values given in the header of SF-NK (section-6) data of associated material. If the density is not available for a compound material, then the refractive index of that material will calculate for unit density (one message will show in the interface for the same).
7. *LayerDensity*: It is an array of density values (usable for *DensityCorrection* = 'yes') of layer-materials to calculate the refractive index. Its length should be equal to the length of the LayerMaterial.

4.1.3 Interface surface roughness

The properties of layer interfaces (interlayer profile, interlayer roughness(σ) etc.) are defined by the following keywords:

1. *SigmaValues* = [0.0] (default):
   The default value is [0.0], which means all the interfaces are considered an ideal surface.
   (a) If it consists of one element, then all the interface of the system will set for the same given RMS surface roughness value.
   (b) If it consists of an array with a number of elements more than one but equal to that of the number of different types of interface, then all the similar interfaces will be set to the given roughness values. Note that, it is not applicable for *MultilayerType* = *UserDefined*.
   (c) If it is an array of length equals the number of interfaces, then the roughness values of each interface will be assigned for the given values from top to bottom of the system.

   Otherwise, the program will through an error message with the correct possible combination of all three above.

2. *InterfaceProf* = '*ErrorFunction*'(default):
   The default value is the '*ErrorFunction*', means the interface profiles are considered as an ErrorFunction for non-ideal case. The other options are−

   (a) '*ErrorFunction*'

   (b) '*ExponentialFunction*'

   (c) '*LinearFunction*'

   (d) '*SinusoidalFunction*'

   (e) '*StepFunction*'

3. *SigmaCorrMethod* = '*NevotCroce*'(default):
   This indicates which methods will be applicable for surface roughness corrections. The default is the Nevot-Croce method (Suggested for X-ray reflectivity). Another option is the Debye-Waller method.

   (a) '*NevotCroce*'

   (b) '*DebyeWaller*'

4.1.4 Correction for experimental effects

In the X-ray reflectivity experiment (XRR), at a very small angle (below critical angle) of the incident beam on the sample, the reflectivity is effected due to the finite size (small) of the sample and incident beam diameter, called projection effect. Also the instrumental finite angular/spatial resolution, polarization state of the incident beam and the Polarization-Analyzer (PA) sensitivity of the detector will effect the calculated optical functions of the mirror sample. DarpanX can include all these effect in the theoretical calculation by using the following keywords:

1. *ProjEffCorr* : (Default='no') If 'yes', then the reflectivity will be corrected for the experimental projection effect.

2. *SampleSize* : (Default=100) Sample sized (in mm along the beam propagation) required for the correction due to the experimental projection effect on reflectivity. This will be use only when $ProjEffCorr$ = '*yes*'.

3. *IncidentBeamDia* : (Default=0.1) Incident beam diameter (in mm) required for the correction due to experimental projection effect on reflectivity. This will be used only when $ProjEffCorr$ = '*yes*'.

4. *BeamPolarization*: (Default=0) Incident beam polarization state. (a) +1 for pure p-Polarization, (b)−1 for pure s-Polarization and (c)0 for unpolarized.

5. *DetectorPA*: (Default=1) Polarization-Analyzer sensitivity (q) of the detector, q=(sensitivity to s-pol / sensitivity p-pol)

6. *InsRes*: (Default=0) Instrumental angular/spatial resolution. Only available for XRR data fitting. If > 0, then the output reflectivity will be convoluted by a Gaussian function of slandered-deviation = InsRes.

4.1.5 Parallel Processing

DarpanX has a parallel processing method, by which it can calculate the optical functions by using more than one (predefined) number of cores of the running system. The keyword '$NumCore = i$' can enable parallel processing. Here, 'i' is an integer number representing the number of cores used in the calculation.

4.1.6 Show all parameters

The keyword '$ShowPar$' controls whether to display the input parameters and values used in the calculation. If $ShowPar$='yes', then all the parameters will be shown in the interface as shown in figure-3.

MultilayerType	UserDefinedML
Repetition	40
TopLayerNum	1
EachRepetitionLayerNum	2
EachRepetitionLayerThick	[20, 40]
LayerMaterial	['PtO2', 'Pt', 'SiC']
DensityCorrection	no
AmbientMaterial	Vacuum
SubstrateMaterial	SiO2
SigmaValues	[2.0]
InterfaceProf	ErrorFunction
SigmaCorrMethod	NevotCroce
ProjEffCorr	no
BeamPolarization	0
DetectorPA	1

(a)

Independent Parameters	
Xscan	Theta
Energy (keV)	[8.0]

	Density Used in Calculation
AmbientDensity	1.0
SubstrateDensity	2.65
LayerDensity	[10.2 21.41 3.21]

(b)

Figure 3: Parameters used in the calculation. Note that, here AmbientMaterial='Vacuum', because of that it is showing AmbientDensity=1.0, it means that the refractive index of ambient medium is consider as (1+0j).

By default '$ShowPar$' is sets to 'yes', if user do not want to show the parameters, then set it to 'no'.

4.2 Method: get_optical_func

After the construction of the multilayer structure and the other experimental setup by initiating the '$Multilayer$' class as discussed in Section 4.1, users can calculate the optical functions by the method, called '$get\_optical\_func$'. For the calculation, users have to define the independent variables, the incident grazing angle ('$Theta$') of the beam in degree and its energy ('$Energy$') in keV unit.

This method can be called as: $drp.Multilayer.get\_optical\_func(parameters = value)$. The parameters are: '$Theta$', '$Energy$' and '$AllOpticalFun$' and described as follows:

1. $Theta$: Grazing incident angles in degree. The length of '$Theta$' should be more than one and the length of '$Energy$' should be equals to one for the calculation of optical properties as a function of angles and vice-versa for the calculation as a function of incident beam energy.

2. *Energy*: Incident beam energy in keV. The length of '*Energy*' should more than one and the length of '*Theta*' should be equals to one for the calculation of optical properties as a function of energy and vice-versa for the calculation as a function of incident grazing angle.

3. *AllOpticalFun*: Users have to define which optical function they want to get as an output. By default, only the reflectivity will be calculated. If *AllOpticalFun* ='*yes*' (only available for theoretical calculation) is selected then only all the optical functions, Reflectivity, Transmittivity, and Absorptance will be calculated.

Note that, for XRR data fitting, the independent array ('*Theta*' or '*Energy*') of more than one element will be taken from the data itself.

4.3 Method: plot

Once the optical function is calculated by the method '*get_optical_func*', users can plot the outputs by using the method '*plot*'. This method can be called as: $drp.Multilayer.plot(parameters = value)$ and it has the following parameters;

1. *OpFun* : (default='yes') If 'yes', it will plot the optical functions.

2. *Struc* : (default='no') If 'yes', it will show the layer structure.

3. *Comp* : (default=None) Optical functions (string array) which user wants to plot. Such as ["Ra","Rp","Rs","Ts","Tp","Ta","As","Ap","Aa"]

4. *AllComp* : (default=None) If not given, it will plot all the optical functions in different windows. If select "oplot", it will over plot them in a single window.

5. *OutFile* : Name (string) of output save file. If given the output plots will be saved in a file.

6. *OutFileFormat* : (default='pdf') Format (pdf, ps, eps, png, jpg) of output save file as given in the previous keyword.

7. *ylog* : If 'yes', y-axis will be in the logarithmic scale.

8. *xlog* : If 'yes', x-axis will be in the logarithmic scale.

9. *xlim* : Array [x1,x2]. X-axis limit in the plot

10. *ylim* : Array [y1,y2]. Y-axis limit in the plot

11. *title* : (default='DarpanX Output') Title of the plot.

12. *Scale* : (default='no') If 'yes' then scaling will be shown in the layer structure plot (as shown in fig-1b).

4.4 More examples

One example for single layer was given in the subsection-3.1. Three more examples with different types of multilayer structure are given here.

4.4.1 Example-3

Consider a bilayer of Pt/SiC with a $Repetition$=40 and an extra top layer of PtO_2 with a thickness 10Å on the top. Let all the interface has a σ value of 1 Å($SigmaValues = [1.0]$). Here, we calculate the average reflectivity in between the grazing incident angles 0-10 degree.

- Import the DarpanX module in the python shell/script and create an instance of darpanx.Multilayer class which defines the parameters of the multilayer structure:

```
import darpanx as drp
import numpy as np
m=drp.Multilayer(MultilayerType="UserDefinedML",SubstrateMaterial="SiO2",TopLayerNum=1,
                                                EachRepetitionLayerNum=2,LayerMaterial=["
                                                PtO2","Pt","SiC"],TopLayerThick=[10],
                                                EachRepetitionLayerThick=[20,40],
                                                Repetition=40,SigmaValues=[1.0])
```

- Then, define the angle and energy values for which the optical functions are to be computed:

```
Energy=[8.0] # Incident beam energy in KeV
Theta=np.arange(start=0,stop=10,step=0.01) # Define theta range in degree .
```

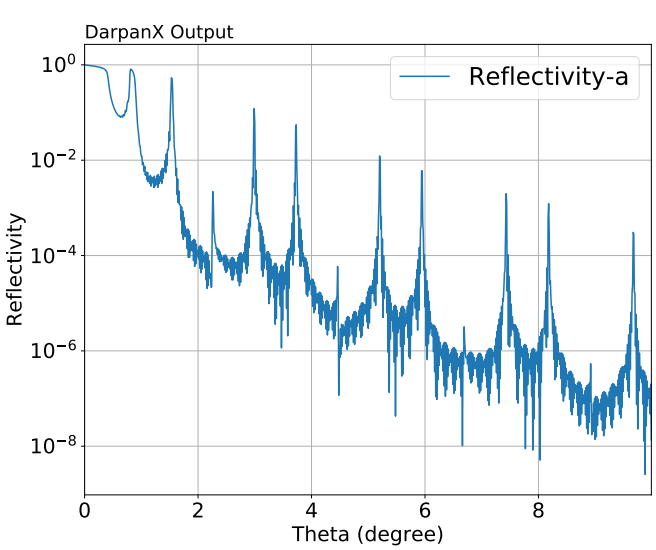
- To compute reflectivity, use the method $get\_optical\_func$ of Multilayer class:

```
m.get_optical_func(Theta=Theta,Energy=Energy)
```

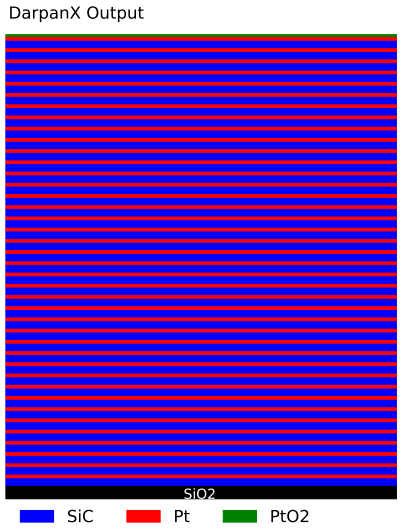
- Plot the outputs and save into .pdf file:

```
m.plot(ylog="yes",Comp=["Ra"],OutFile="Example3_Pt_SiC",Struc="yes")
```

Figure-4 shows the reflectivity as a function of grazing angles and the multilayer structure. Note that this example is available in the script $Example3.py$ inside the directory $DarpanX/examples/Example3$.



(a)



(b)

Figure 4: (a) Average Reflectivity. (b) Pt/SiC Bi-layer, shown in red/blue. Top PtO_2 is shown in green and substrate is in black.

4.4.2 Example-4

We can make any arbitrary layer profile by choosing $MultilayerType$=$UserDefined$. In this case we have to give the layer thickness profile (Z_Array) and material names ($LayerMaterial$) for each layer as a input. Consider a multilayer consisting with layer of ['W', 'Si', 'W, 'C', 'Pt', 'C', 'Ni'] with an exponential thickness profile and then we want to calculate the optical functions for this structure.

- Import the DarpanX module in the python shell/script and create an instance of darpanx.Multilayer class which defines the parameters of the multilayer structure:

```
import darpanx as drp
import numpy as np

#Define some random thickness profile:
expo=np.arange(7)-3
Z_Array=np.exp(-expo)+5.0
#Make multilayer structure
m=drp.Multilayer(MultilayerType="UserDefined",SubstrateMaterial="SiO2",LayerNum=7,
                                                    LayerMaterial=["W","Si","W","C","Pt","C","
                                                    Ni"],Z_Array=Z_Array)
```

- Define the angle and energy values for which the optical functions are to be computed:

```
Energy=[8.0] # Incident beam energy in KeV
Theta=np.arange(start=0,stop=5,step=0.01) # Define theta range in degree .
```

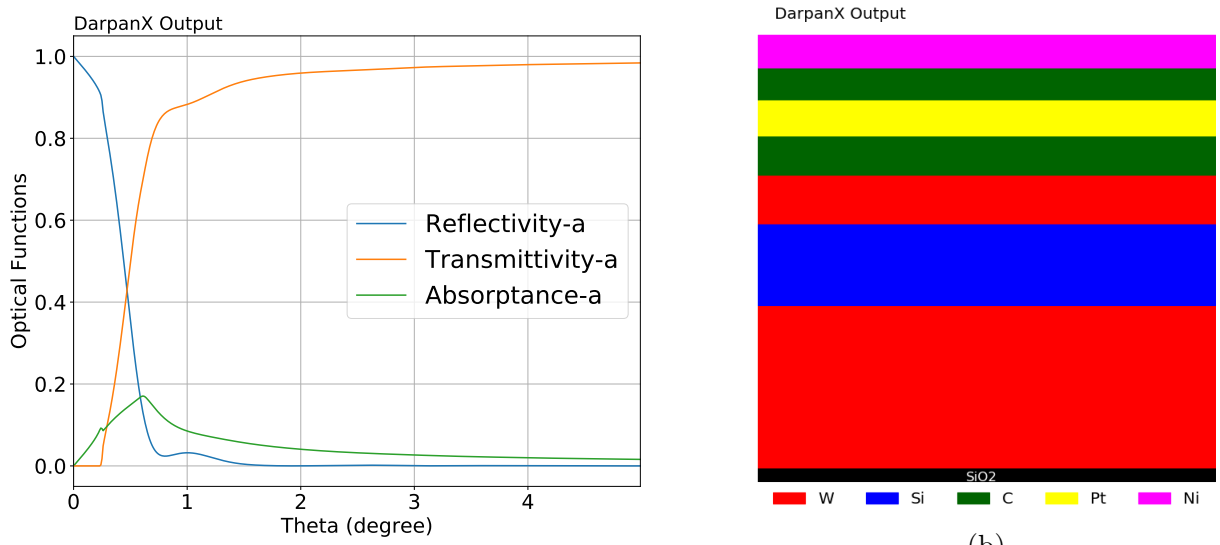
- To compute optical functions, use the method $get\_optical\_func$ of Multilayer class:

```
m.get_optical_func(Theta=Theta,Energy=Energy,AllOpticalFun ="yes")
```

- Plot and save the outputs:

```
m.plot(ylog="no",Comp=["Ra","Ta","Aa"],AllComp="oplot",OutFile="Example4_userdefined",
                                                    Struc="yes")
```

Figure-5 shows the optical functions and the multilayer structure. This example is available in the script *Example4.py* inside the directory DarpanX/examples/Example4.



(a) (b)

Figure 5: (a) Average Optical functions. (b) Multilayer structure.

4.4.3 Example-5

In this example, a depth-graded multilayer structure is considered. The thickness profile of the structure is taken from Brejnholt (2012)[4]. Here the thickness of the bilayers are varying from top to bottom of the multi-layer system according to the following formula:

$$d_i = \frac{a}{(b+i)^c} \qquad where, i = 1, 2...N \ \ is \ \ the \ \ bilayer \ \ numbers(Repetation). \tag{1}$$

where, $a = D_{min}(b+N)^C$, $b = \frac{1-N.k}{k-1}$ and $k = (\frac{D_{min}}{D_{max}})^{\frac{1}{C}}$, here, D_{min}, D_{max} is the bottom most and top most d-spacing (or period) and C controls the slop between these extreme value over the N-bilayers.

Consider a depth-graded Pt/SiC system of 150 bilayers with the specification given in the following table:

$Material$	D_{max} ($\mathring{A}$)	D_{min} ($\mathring{A}$)	N	Γtop	Γ	c	σ ($\mathring{A}$)
Pt/C	128.10	31.70	150	0.7	0.45	0.245	4.5

Table 1: Multilayer depth-graded recipe details. D_{max} and D_{min} is the maximum and minimum period corresponding to the top and bottom most bi-layer. N and Γ are the number of bi-layers and gamma factor respectively. Γtop is the gamma factor corresponding to the top most layer. C controls the slope between these extreme values of D_{max} and D_{min} over the N layers.

We want to calculate the reflectivity of this depth-graded system as a function of incident X-ray beam energy from 0.1-80.0 keV for a incident grazing angle of 0.08^0.

- Import the DarpanX module in the python shell/script and create an instance of darpanx.Multilayer class which defines the parameters of the multilayer structure:

```
import darpanx as drp
import numpy as np
m=drp.Multilayer(MultilayerType="DepthGraded",SubstrateMaterial="SiO2",LayerMaterial=["Pt
                                                    ","C"],Repetition=150,D_max=128.1,D_min=31
                                                    .7,C=0.245,GammaTop=0.7,Gamma=0.45,
                                                    SigmaValues=[4.5])
```

- Define the angle and energy values for which the optical functions are to be computed:

```
Theta=[0.08] # Incident grazing angle in degree.
Energy=10**(np.arange(start=np.log10(0.1),stop=np.log10(80.0),step=0.01)) # 0.1-80.0 keV
                                                    Energy in linear grid.
```

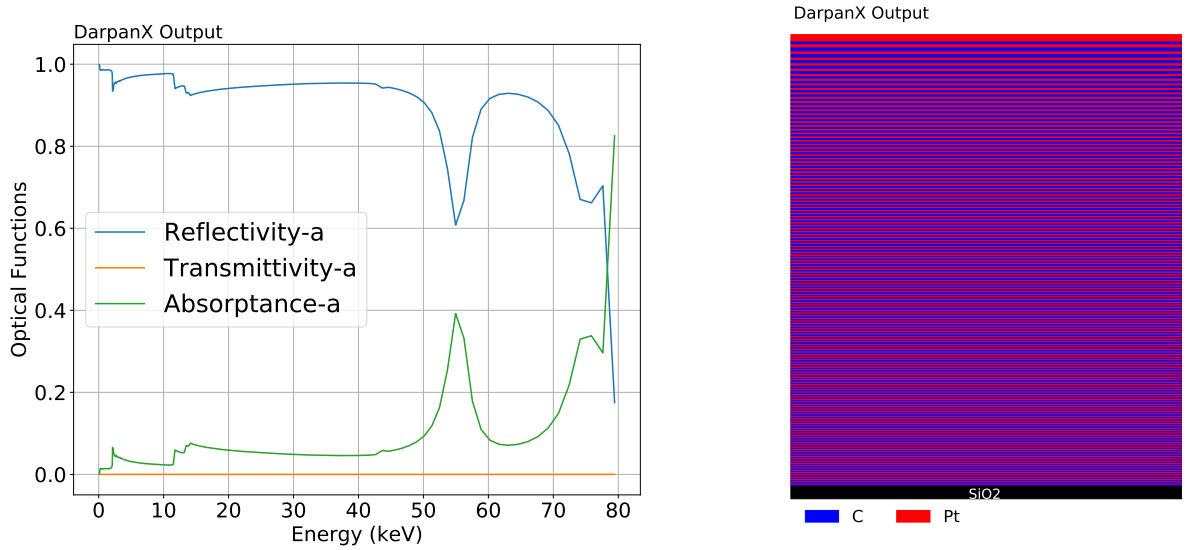
- To compute optical functions, use the method get optical f unc of Multilayer class:

```
m.get_optical_func(Theta=Theta,Energy=Energy,AllOpticalFun ="yes")
```

- Plot and save the outputs:

```
m.plot(ylog="no",Comp=["Ra","Ta","Aa"],AllComp="oplot",OutFile="Example5_Pt_C_DepthGraded
                                                    ",Struc="yes")
```

Figure-6 shows the computed optical functions and the multi-layer structure profile. This example code is available in the directory: DarpanX/examples/Example5.



(a) (b)

Figure 6: (a) Average Optical functions. (b) Multilayer structure.

Two more options (MultilayerType=BiLayer and MultilayerType=ClusterGraded) are given for the multilayer profile, which are not shown in the examples. Details of these type of multilayers are given in section-4.1.1.

5 DarpanX usage: XRR data fitting

DarpanX can be used as a tool for the diagnostic of experimental XRR data by using it as a local model of PyXspec. In this section we will describe how to use DarpanX as a model function in PyXspec to fit the XRR data and extract the properties of the layer structure.

For the DarpanX model to fit the XRR data, a user input configuration file is required in the prescribed format as given in the Section-5.1. Inside the configuration file, the required keywords/parameters, as described in Sections 4.1 to form a layer structure, should be defined in the prescribed format. However to make the model for the XRR data fitting, an extra keyword $Xscan$ (='Theta' or 'Energy') is required. For example, if the experimental data is the reflectivity as a function of grazing angle, then the 'Xscan' should be equal to 'Theta' (means the x-axis is theta of the XRR data), then the grazing angles to compute the optical functions will take from the XRR data itself and users have to provide the value of incident beam energy.

5.1 Input parameter file definition

All the required input keywords/parameters as discussed in Section 4.1 to form the model of layer structure, should be given inside an ASCII file, each line containing one keyword/parameter. Note that for all the keywords, the value/values should be given inside a square bracket ([ ]) irrespective of any type (like string/array/float, etc.). For a particular $MultilayerType$, the program will only consider the associated keywords, and others will be ignored. If some values are given wrong (or not given in the file), then it will consider the default value of it. Note that, If $Xscan =$ '$Theta$', a PyXspec model function will be available as a name '$darpanx\_Th$' and if $Xscan =$ '$Energy$',

then a PyXspec model named, '*darpanx_En*' will be available.

The format for the input configuration file (where, we have to define the keywords/parameters as discussed in Section 4.1 including the parameter '$Xscan$') is as follows:

$NK\_dir = [/home/DarpanX/nist/]$
$Xscan = [Theta]$
$Energy = [8.075]$
$Theta = [0.2]$
$MultilayerType = [SingleLayer]$
$AmbientMaterial = [Vacuum]$
$SubstrateMaterial = [SiO2]$
$LayerMaterial = [Si]$
$Repetition = [1]$
$TopLayerNum = [0]$
$TopLayerThick = [3.0]$
$EachRepetitionLayerNum = [2]$
$EachRepetitionLayerThick = [20, 100]$
$Period = [300.0]$
$Gamma = [0.4]$
$D\_max = [95.0]$
$D\_min = [25.0]$
$C = [0.2]$
$GammaTop = [0.8]$
$NumStuck = [4]$
$LayerNum = [None]$
$SigmaValues = [4.0, 4.0]$
$DensityCorrection = [yes]$
$InterfaceProf = [ErrorFunction]$
$SigmaCorrMethod = [NevotCroce]$
$BeamPolarization = [0]$
$DetectorPA = [1]$
$ProjEffCorr = [yes]$
$SampleSize = [20]$
$IncidentBeamDia = [0.2]$
$LayerDensity = [2.32]$
$InsRes = [0]$
$Z\_Array = [None]$
$ShowPar = [yes]$
$NumCore = [4]$

5.2 Loading the model and fitting the data

After making the configuration file, users have to initialized the DarpanX and PyXspec in the python shell as follows:

```
heainit # Initializing HEASoft package for PyXspec.
ipython # Opening the ipython.
import darpanx as drp # Load DarpanX
from xspec import* # Load PyXspec
```

Then user can use the DarpanX method, '*call_wrap*' to load the configuration file (such as '*InputConfig.dat*') in DarpanX and connect DarpanX with PyXspec, as follows:

```
drp.call_wrap("InputConfig.dat")
```

Once it is connected, all the required input parameters will show in the python shell as shown in the figure-7.

```
=====================================================

%% DarpanX_message: Configuraton file (UserInFile) used:  inputs.dat
%% DarpanX_Settings: Undefined < NK_dir >. Set default direcrory (nk_data/nk)
%% DarpanX_Settings: Incorrect or Undefined < BeamPolarization >. Set to 0
%% DarpanX_Settings: Incorrect or Undefined < DetectorPA >. Set to 1
%% DarpanX_message: Parallel processing is using with no.cores = 8

+-------------------+---------------+
| MultilayerType    | SingleLayer   |
|-------------------+---------------|
| Period            | [300.0]       |
| LayerMaterial     | ['Si']        |
| DensityCorrection | yes           |
| LayerDensity      | [2.32]        |
| AmbientMaterial   | Vacuum        |
| SubstrateMaterial | SiO2          |
| AmbientDensity    | Default       |
| SubstrateDensity  | Default       |
| SigmaValues       | [4.0, 4.0]    |
| InterfaceProf     | ErrorFunction |
| SigmaCorrMethod   | NevotCroce    |
| ProjEffCorr       | yes           |
| SampleSize        | 20.0          |
| IncidentBeamDia   | 0.2           |
| BeamPolarization  | 0             |
| DetectorPA        | 1             |
+-------------------+---------------+

+--------------------------+-------+
| Independent Parameters   |       |
|--------------------------+-------|
| Xscan                    | Theta |
| Energy (keV)             | 8.075 |
+--------------------------+-------+


=====================================================
```

Figure 7: Parameter values used in the configuration file.

Once the DarpanX and PyXspec are connected with each other, users have to call the DarpaX method, *make_model*, to make a reflectivity model accessible from PyXspec as follows:

```
drp.make_model()
```

Then that model is used as a PyXspec local model, and all the PyXspec commands as described in the PyXspec manual (available in the page-pyxspec) are applicable for the model and for the fitting user can use the class $xspec.FitManager$. All the information (like, model name, parameter values, which parameters are considered as a free variable for fitting, etc.) of the model will show in the interface, looks like in figure-8 (for a single Si-layer model with finite instrumental resolution and corrected for projection effect.).

```
Parameters defined:
========================================================================
Model gsmooth<1>*darpanx_Th<2> Source No.: 1   Active/On
Model Model Component  Parameter  Unit     Value
 par  comp
   1    1   gsmooth    Sig_6keV   keV      2.00000E-02  frozen
   2    1   gsmooth    Index               0.0          frozen
   3    2   darpanx_Th d_Si       Angs     300.000      +/-  0.0
   4    2   darpanx_Th rho_Si     g/cm3    2.00000      +/-  0.0
   5    2   darpanx_Th BeamDia    mm       0.200000     frozen
   6    2   darpanx_Th sigma0     Angs     4.00000      +/-  0.0
   7    2   darpanx_Th sigma1     Angs     4.00000      +/-  0.0
   8    2   darpanx_Th norm                1.00000      +/-  0.0
------------------------------------------------------------------------
```

Figure 8: Model information: Column-1 is the parameter number; Column-2 is the model components, here two models are present, gsmooth and darpanx_Th as discussed in the text; Column-3 shows the parameters name of the models, here sig_6keV and $Index$ are the parameters associated with gsmooth model and d_Si, rho_Si, $BeamDia$, $sigma0$, $sigma1$ are the parameters associated with the model darpanx_Th as mentioned in the text. The overall normalization factor is shown by the $norm$ parameter.; Column-5 shows the unit of the parameters; Column-6 shows the present values of the associated parameters and Column-7 shows the status of the parameters for fitting, if it shows $frozen$, then that parameter will not vary to fit the XRR data.

In figure-8, the thickness, density, and roughness of two interfaces (top to bottom) of the single Si layer is shown by the parameters d_Si, rho_Si, $sigma0$, and $sigma1$, respectively. Note that if we choose a finite instrumental resolution ($InsRes > 0$) in the configuration file, an extra Gaussian convolution model is required to include the effect of instrumental resolution with actual calculation (as discussed in subsection-4.2). Then the $make\_model$ method will use the XSPEC convolution model $gsmooth$ for the same purpose, where the $darpanx\_Th/darpanx\_En$ model is convolved with $gsmooth$. The $gsmooth$ model is a simply Gaussian smoothing model with a variable standard deviation (Sig_6keV), but $make\_model$ use it with a constant standard deviation by setting the $Index$ parameter to 0.0 and make it frozen for the fitting. Details about gsmooth are given in this link-gsmooth. Hence the instrumental resolution is represented by the parameter Sig_6keV in figure-8.

Inside configuration file if we choose, $ProjEffCorr = [yes]$, then as discussed in subsection-4.1.4, the model reflectivity will be corrected for the experimental projection effect, which is very prominent at a lower grazing angle (less than the critical angle) for the small size of the sample and large incident beam diameter of the XRR setup. This effect is required to consider modeling the XRR data below the critical angle. Sometimes the actual beam diameter (at the sample position)

of the XRR setup is not known, because of that *make_model* will consider another parameter, *BeamDia* (incident beam diameter) as a fitting variable. If the incident beam diameter is well known, then the user can freeze this parameter with that value.

5.3 Plotting the fit results

To plot the fitted data along with the model, users can use the PyXspec Plot method. Alternatively DarpanX has two methods, *plot*() and *iplot*() (can be called as: $drp.plot(parameter = value)$ and $drp.iplot(parameter = value)$) to plot the XRR data along with the model. The *iplot* method uses the PyXspec PlotManager and combines most of the PyXspec commands as a function. All the parameters of these methods are given below:

- method *plot()*:
    1. *xlim* : Array [x1,x2]. X-axis limit in the plot
    2. *ylim* : Array [y1,y2]. Y-axis limit in the plot
    3. *xlog* : If 'yes', y-axis will be in the logarithmic scale.
    4. *ylog* : If 'yes', y-axis will be in the logarithmic scale.
    5. *title* : (default='DarpanX Output') Title of the plot.
    6. *Struc* : (default='no') If 'yes', it will show the layer structure.
    7. *Scale* : (default='no') If 'yes' then scaling will show in the layer structure plot (as shown in fig-1b)
    8. *OutFile* : Name (string) of output save file. If given the output plots will be saved in a file.
    9. *OutFileFormat* : (default='pdf') Format (pdf, ps, eps, png, jpg) of output save file as given in the previous keyword.
- method *iplot()*:
    1. *device* : (default=$'/xw'$) Which X-window you want to open for plotting.
    2. *xlog* : (default=False) If True then the x-axis will in log scale.
    3. *ylog* : (default=True) If True then the y-axis will in log scale.
    4. *xaxis* : (default=$'kev'$) In XSPEC there are two option to plot in x-axis, either channel(i.e, $'ch'$) or energy (i.e $'kev'$).
    5. *comp* : (default=' ') Along with the data with fitted model one can plot the ratio, residual, delta-chi etc in a subplot. To plots these in subplot chose $comp = 'ratio'$ or $comp = 'ratiodelc'$ etc.
    6. *xlabel* : (default='Theta (degree)') X-axis label.
    7. *ylabel* : (default='counts/s/degree') Y-axis label.
    8. *OutFile* : Output file name to save the plot.

5.4 Saving and restoring the fit results

Users can save the interactive session of XRR data fitting along with the outputs, by using the 'save()' method (can be called as: $drp.save(parameters = value)$). It has the following parameters:

1. *OutFile* : Name of output file. If OutFile='DarpanxOut', then it will save three files-
   (a) DarpanxOut.drpnx - Contains all the information, including configuration of model component, output model data, etc
   (b) DarpanxOut.xcm - Contains the output fitted results from XSPEC.
   (c) DarpanxOut.py - Loading file for future. To load the output, users have to run this file in python interface. Note that, to run this, the other two files(DarpanxOut.drpnxand DarpanxOut.xcm) should be in the same directory.
2. *SaveAll* : If SaveAll='yes', then all the model parameters along with the output optical functions values will be saved in .drpnx file. Otherwise only the model parameters will save.

5.5 More examples

One example for single layer XRR data fitting was given in the subsection-3.2. One more example of the XRR data fitting for a multilayer structure is given here.

5.5.1 Example-6:

Consider a W/B_4C bi-layer sample consisting of 50 numbers of repetitions formed on a Si substrate. We want to model this sample and fit the XRR data with DarpanX multilayer layer model. Give the minimum required parameters inside the configuration file, $InputConfig.dat$ as follows:

$Xscan = [Theta]$
$Energy = [8.075]$
$MultilayerType = [UserDefinedML]$
$AmbientMaterial = [Vacuum]$
$SubstrateMaterial = [Si]$
$LayerMaterial = [W, B4C]$
$Repetition = [50]$
$TopLayerNum = [0]$
$TopLayerThick = [0]$
$EachRepetitionLayerNum = [2]$
$EachRepetitionLayerThick = [20, 20]$
$SigmaValues = [4.0, 4.0, 4.0, 4.0]$
$DensityCorrection = [yes]$
$ProjEffCorr = [yes]$
$SampleSize = [30]$
$IncidentBeamDia = [0.1]$
$LayerDensity = [19.25, 2.52]$
$InsRes = [0.001]$

$NumCore = [12]$

Note that we used $NumCore = [12]$, i.e., 12 number of cores of the running system will use for parallel computing. The XRR data of this sample is located inside the directory- $DarpanX/examples/Exampl$ Here we will describe the fitting procedure of this sample step by step as follows:

Initialize HEASoft package for XSPEC (e.g, heainit) and open ipython/python (v +3.x) interface.

```
cd DarpanX/examples/Example6_XRR/BiLayerModel#Changing the current directory to Example6_XRR/
                                              BiLayerModel
heainit #Initializing HEASoft package for XSPEC
ipython #Opening the ipython
```

Import the DarpanX and pyXspec modules in the ipython shell:

```
import darpanx as drp # Load DarpanX
from xspec import* # Load PyXspec
```

Define configuration file and call DarapanX-PyXspec wrapper method:

```
UserInFile="InputConfig.dat"
drp.call_wrap(UserInFile)
```

After this all the input parameters and settings will show in the interface, which will look similar to figure-8.

Call *make_model*() method to load the DarpanX model as a local model in PyXspec.

```
drp.make_model()
```

Now the DapanX model is available as a PyXspec local model, and all the PyXspec commands are applicable for the model. All the parameters will be display in the terminal, as shown in figure-9.

```
Parameters defined:
========================================================================
Model gsmooth<1>*darpanx_Th<2> Source No.: 1   Active/On
Model Model Component  Parameter  Unit     Value
 par  comp
   1    1   gsmooth    Sig_6keV   keV      1.00000E-03  frozen
   2    1   gsmooth    Index                0.0          frozen
   3    2   darpanx_Th d_W        Angs     17.0000      +/-  0.0
   4    2   darpanx_Th d_B4C      Angs     27.0000      +/-  0.0
   5    2   darpanx_Th rho_W      g/cm3    19.2500      +/-  0.0
   6    2   darpanx_Th rho_B4C    g/cm3    2.52000      +/-  0.0
   7    2   darpanx_Th BeamDia    mm       0.100000     frozen
   8    2   darpanx_Th sigma0     Angs     4.00000      +/-  0.0
   9    2   darpanx_Th sigma1     Angs     4.00000      +/-  0.0
  10    2   darpanx_Th sigma2     Angs     4.00000      +/-  0.0
  11    2   darpanx_Th sigma3     Angs     4.00000      +/-  0.0
  12    2   darpanx_Th norm                1.00000      +/-  0.0
------------------------------------------------------------------------
```

Figure 9: Model information: Column-1 is the parameter number; Column-2 is the model components, here two models are present, gsmooth and darpanx_Th as discussed in subsection-5.2; Column-3 shows the parameters name of the models, here sig_6keV represent the instrumental resolution and d_W, d_B4C, rho_W, rho_B4C are the thickness and density of W and B_4C layer. $sigma0$, $sigma1$, $sigma2$, $sigma3$ are the surface roughness values for the interfaces $ambient/W$, W/B_4C, B_4C/W and $W/substrate$, respectively. $BeamDia$ represents the incident beam diameter in mm. The $norm$ parameter shows the overall normalization factor.; Column-5 shows the unit of the parameters; Column-6 shows the present values of the associated parameters and Column-7 shows the status of the parameters for fitting, if it shows $frozen$, then that parameter will not vary to fit the XRR data.

Now the experimental XRR data and associated response file), which we want to fit, has to be loaded. This can done by using PyXspec *Specrum* method as follows:

```
s=Spectrum("W_B4C_XRR.pha")
s.response="W_B4C_XRR.rsp"
```

Ignoring the data, which is not required for fitting (here < 0.26 deg and > 6.0 deg):

```
AllData(1).ignore("**-0.26 6.0-**")
```

This W/B_4C sample was made with a target period (total thickness of W and B_4C) of 44Å and gamma factor(ratio of W thickness to the period) of 0.4, which means the target thickness of W and B_4C were 17.5Å and 27.0Å. Because of that we choose the guess values for thicknesses of W (d_W) and B_4C (d_B4C) is 17.0Å and 27.0Å and varying them with a range of 10.0-30.0Å and 20.0-40.0Å. Setting the guess values and lower/upper bounds for the thicknesses as follows:

```
AllModels(1)(3).values=[17.0, 1.0, 10.0, 10.0, 30.0, 30.0]
AllModels(1)(4).values=[27.0, 1.0, 20.0, 20.0, 40.0, 40.0]
```

Note that, if users have no prior information about the sample, they can use the PyXspec *steppar* method to find out a rough estimation of the parameters by choosing the guess value, where the chi-squared is converging.

we consider the density (rho_W and rho_B4C) of both the W and $B4C$ as free variables and choosing the guess values of $19.0 g/cm^3$ and $2.5 g/cm^3$ with a variation of $10.0 - 19.6 g/cm^3$ and $1.0 - 2.6 g/cm^3$, respectively.

```
AllModels(1)(6).values=[19.0, 1.0, 10.0, 10.0, 19.0, 19.6]
AllModels(1)(6).values=[2.5, 1.0, 1.0, 1.5, 2.5, 2.6]
```

Similarly, we considered the surface roughness of different type of interface ($ambient/W$ as $sigma0$, W/B_4C as $sigma1$, B_4C/W as $sigma2$ and $B_4C/substrate$ as $sigma3$) as a free parameters for fitting with an initial guess value 2.0Å for each of them and varying them in the region of 1Å to 14Å. These steps are done by changing the values of each parameter as follows:

```
AllModels(1)(8).values=[2.0, 0.1, 1.0, 1.0, 9.0, 14.0]
AllModels(1)(9).values=[2.0, 0.1, 1.0, 1.0, 9.0, 14.0]
AllModels(1)(10).values=[2.0, 0.1, 1.0, 1.0, 9.0, 14.0]
AllModels(1)(11).values=[2.0, 1.0, 1.0, 1.0, 9.0, 14.0]
```

The XRR measurement of this sample is done for the setup of incident beam diameter 0.3 mm and the instrumental resolution of setup was nearly 0.015 degrees. Because of that, we choose these parameters as a fixed variable for fitting with their associated values:

```
AllModels(1)(1).values=[0.015, -0.001, 0.0, 0.0, 0.1, 0.1]
AllModels(1)(7).values=[0.3, -0.001, 0.0001, 0.0001, 0.30, 0.4]
```

If required, set some systematic error as follows (here 10% is used. Without giving systematic it is possible to fit if the error associated with the data is well known):

```
AllModels.systematic =0.1
```

Plot the data with model before fitting:

```
drp.iplot()
```

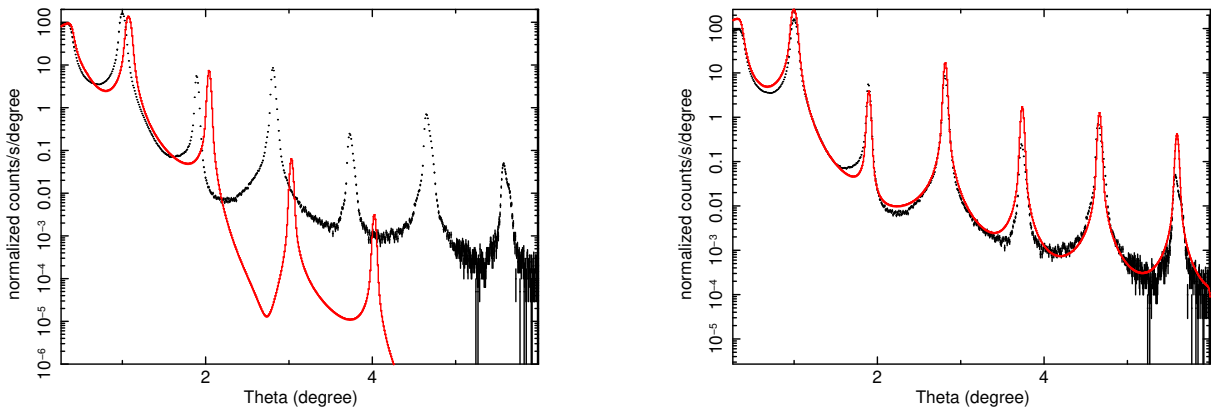
Fitting the data with model by using the PyXspec FitManager class as follows:

```
Fit.nIterations=100 # Set maximum number of fit iterations to perform.
Fit.perform() #run for fitting
```

Re-plot the data with model after fitting:

```
drp.reiplot()
```

Figure-10a shows the fitted XRR data with the bi-layer model before fitting and figure-10b shows after fitting.



(a) (b)

Figure 10: Data (black) with model (red) (a) Before fitting. (b) After fitting.

Users can use various functionality available in PyXspec (e.g., steppar, error, etc. as described in PyXspec manual, available in the page-pyxspec) for the analysis. To define some finite offset between the model and XRR data, users can use PyXspec response parameters $gain.offset$. In DarpanX, it can be alternatively available through a method, $gainfit()$. In this method, it will set the offset value to 0.0 and make it as a free variable for fitting within the range -0.1 to +0.1. It can be callable as follows:

```
drp.gainfit(status="on") # If status="off" then it will freeze the offset value for fitting.
```

Save the outputs (results, configuration setup etc)

```
drp.save(OutFile="Example6_XRR_W_B4C_BiLayer_ModelFit")
```

It will save three file:

1. $Example6\_XRR\_W\_B4C\_BiLayer\_ModelFit.drpnx$ - Contains all the information, including configuration of model component, output model data, etc
2. $Example6\_XRR\_W\_B4C\_BiLayer\_ModelFit.xcm$ - Contains the output fitted results from XSPEC.
3. $Example6\_XRR\_W\_B4C\_BiLayer\_ModelFit.py$ - Loading file for future. To load the output, users have to run this file in the python sell (here in ipython) as:

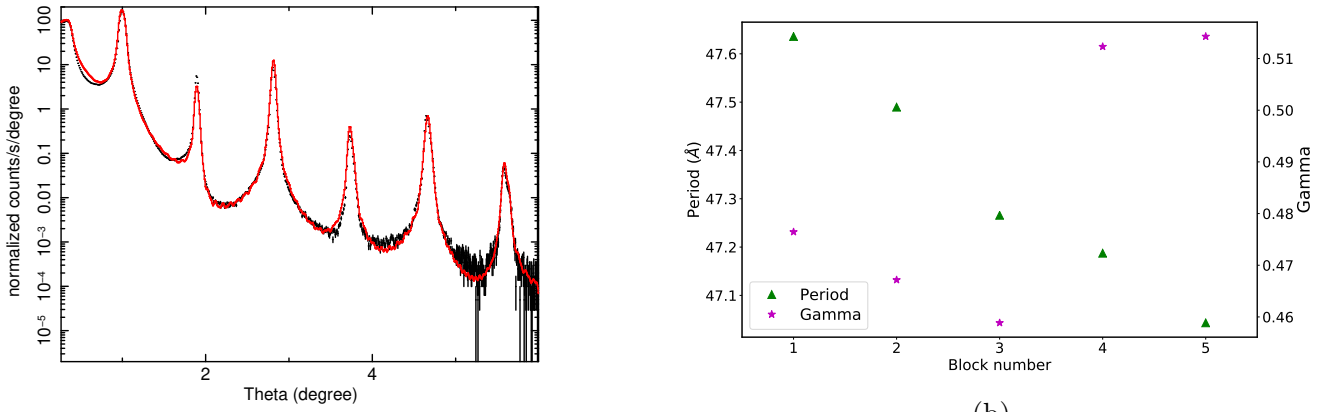
   ```
   %run Example6_XRR_W_B4C_BiLayer_ModelFit.py
   ```

   Then, the previous fitting will be loaded into the current session.

Figure-10b shows that the data is not fitted well with the bi-layer model. This is because we consider a bi-layer constant period model, where all the bi-layer has a constant value of period and gamma values. However, because of the experimental uncertainty, there are variation of period/gamma values throughout the multilayer formation. To include this effect, we segregated the bilayer systems as a succession of 5 blocks formed by ten bilayers each. This type of multilayer structure is called cluster graded. As a result, any period variation of the layers will be approximately modeled. We assume the density of B_4C, W , and the surface roughnesses of B_4C/W

, W/B_4C interfaces remain the same over the multilayer stack and keep them as free variables for fitting. In this cluster-graded model, we fit the data, and all the outputs are saved inside the directory- DarpanX/examples/Example6_XRR/ClusterGradedModel.

Figure-11a shows the fitted data with the cluster-graded model, and figure-11b shows the period and gamma values at each block of the cluster-graded model.



(a) (b)

Figure 11: (a) Fitted data with model. (b) Variation of average period and gamma of W/B_4C bilayers at each block of cluster-graded model.

We can see from figure-11b; though the variation of period and gamma values at each block of the cluster-graded model are not varying much, the model describes the XRR data well (fitting is good in figure-11a). Thus, in this case, the multilayer sample is not a pure constant period. However, if we consider the surface roughness values of each block as a free variables along with the period and gamma values, the fitting will improve a bit.

6 SF-NK Database

The calculations in DarpanX require an SF-NK database, where either X-ray scattering factors (SF) or refractive index (NK) values as a function of energy is given for different layer materials. NIST (National Institute of Standards and Technology) provides the SF for 92 elements from Z=1 to Z=92 in the energy range of $0.001 - 433$ keV [3]. DarpanX uses this SF data from NIST to calculate the refractive index (For details, see the DarpanX paper). Inside DarpanX package one script ($get\_NISTData.py$) is given to download the NIST data sets.

6.1 Database structure

All the NIST SF data are downloaded inside the directory $nk\_data/nist/sf$. To calculate the refractive index from SF data a class, $darpanx\_nkcal.nkcal()$ is given within the package. Note that for the refractive index calculation from SF data, the density and Atomic weight of the material is required. Inside SF data sets the density and atomic weight values are stored as a header for 92 elements. But if a layer consists of a compound material such as $SiO_2 (= Si + O + O)$, then

users have to provide the density values to calculate the refractive index (NK) values. A simple script (*get_NkFromSF.py* inside the directory *darpanx*) is given for the calculation of NK from SF by using the class *darpanx_nkcal.nkcal*(). The NK values for all 92 elements, along with some compounds are calculated by using this script and saved inside the directory *nk_data/nist/nk*.

During calculation of optical functions within DarpanX whenever the NK values are required, initially, it will search the nk directory for the NK data set. If the NK data set is available DarpanX will use that for the further calculation; otherwise it will search the sf directory for the SF data set and then it will calculate NK from SF values and then proceed for further calculations. Note that for a compound, if the NK data is not available then for the calculation of NK from SF the density values has to be provide by the keyword *DensityCorrection* and *LayerDensity*. Otherwise, the program will through an error message.

Users can use their own SF or NK data sets by setting $'NK\_data'$ keyword to your directory (i.e, NK_dir =To your directory) and making two sub-directory 'sf' and 'nk' within that directory and saved your SF and NK data within the sub-directories 'sf' and 'nk'. Note that for the NK and SF data sets, a predefined format is required. These are as follows:

- SF data format:

$\#$
$\#$
$\#Element = Ca$
$\#Z = 20$
$\#A(g/mol) = 40.08$
$\#Rho(g/cm3) = 1.55$
$\#Energy\ range = 0.001 - 433\ keV$
$\#$
$\#\ E(keV) \quad f1(e/atom) \quad f2(e/atom)$
$4.45740e-02 \quad 7.88161e+00 \quad 8.64609e-01$
$4.63967e-02 \quad 7.85859e+00 \quad 8.85830e-01$
. . . . .
. . . . .

- NK data format:

$\#$
$\#$
$\#Element = Ca$
$\#Z = 20$
$\#A(g/mol) = 40.08$
$\#Rho(g/cm3) = 1.55$
$\#Energy\ range = 0.001 - 433\ keV$
$\#$
$\#\ Wavelength(A) \quad n \quad k$
$4.78555e+02 \quad 8.88091e-01 \quad 3.53709e-01$
$4.55722e+02 \quad 7.57062e-01 \quad 1.80675e-01$

. . . . .
. . . . .

6.2 Computing refractive indices from scattering factors

Users can use the *darpanx_nkcal.nkcal*() class for the NK data calculation from their SF data sets. In this sub-section, we will describe how to use it.

Consider we have SF data for elements Si and O, named as Si.dat and O.dat (in the directory $test/sf$) in the same format as described above. We want to calculate NK data of SiO_2. In first step we have to initialized DarpanX and then define compound element name and density for which we want to calculate NK data. We also have to provide the directory where sf and nk sub-directories are present. After that we can calculate the NK by using the method *darpanx_nkcal.nkcal.get_nk* and finally we can plot SF and NK by using *darpanx_nkcal.nkcal.plot_sfnk* method.

- Initialize DarpanX:

```
import darpanx as drp
```

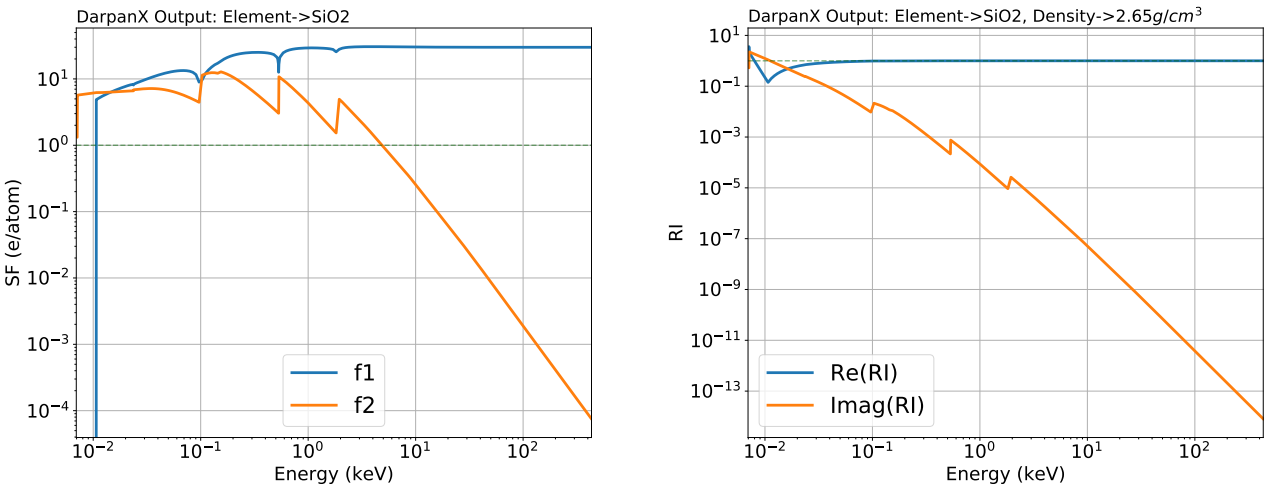
- Define compound name, density and the directory where sf and nk sub-directories are exist.

```
dir=test
CompoundElements="SiO2"
Density=2.65
```

- Calculate NK and plot outputs:

```
RI=drp.nkcal(NK_dir=dir,Formula=CompoundElements) # Call darpanx_nkcal class
RI.get_nk(OutDir=dir+"/nk",Density=Density) # Calculate NK
RI.plot_sfnk(Comp=["SF","NK"]) # Plot SF and NK
```

- Plots:


DarpanX Output: Element->SiO2
SF (e/atom)
Energy (keV)
f1
f2
DarpanX Output: Element->SiO2, Density->2.65g/cm^3
RI
Energy (keV)
Re(RI)
Imag(RI)

(a) (b)

Figure 12

References

[1] M. Born, E. Wolf, A. B. Bhatia, *et al.*, *Principles of Optics: Electromagnetic Theory of Propagation, Interference and Diffraction of Light*, Cambridge University Press, 7 ed. (1999).

[2] J. D. Jackson, *Classical electrodynamics*, Wiley, New York, NY, 3rd ed. ed. (1999).

[3] "X-Ray Form Factor, Attenuation, and Scattering Tables." Available at https://physics.nist.gov/PhysRefData/FFast/html/form.html (2020/01/01).

[4] N. Brejnholt, *NuSTAR calibration facility and multilayer reference database: Optic response model comparison to NuSTAR on-ground calibration data: Optic response model comparison to NuSTAR on-ground calibration data*. PhD thesis, DTU Space — National Space Institute — Technical University of Denmark (2012).