Calculation of the Beamline intensity based on WavePropaGator interface

Nuria Olivares i Royo, Universitat Politècnica de Catalunya, Spain

Supervisor: Ruiz Lopez, Mabel

September 4, 2019

Abstract

Usually the optical design of beamlines for the Free Electron Laser is realized with specific codes and interfaces i.e., Wave PropaGator. We developed an add-up code based on Python structures for calculating the final photon intensity at the end of beamlines. For this aim the reflectivity of mirrors, the dimensions of the optical elements and the source properties are considered. Although diffraction effects were not integrated in the code, a preliminary analysis shows that the difference with ray-tracing calculations is not significant.

Contents

1. Motivation

FLASH, the Free Electron Laser in Hamburg has extraordinary properties: high brilliance [more than 10^{28} ph s^{-1} mm^{-2} mrad^{-2}/0,1% Bandwidth], ultrashort pulse duration (10-200 fs) and high repetition rate (up to 8000 pulses/s in burst mode) [1, 2]. FLASH operates with wavelengths in the range of the Extreme Ultraviolet (EUV) from 4.1 nm to up to 90 nm.
The facility includes since 2014 a new experimental Hall named "Kai Siegbahn" like the physics Nobel Prize winner. The new Hall will double the number of user stations and nowadays FLASH operates 6 beamlines. The photon beam parameters can be chosen almost independently from FLASH1 thanks to an additional variable-gap undulator.

1.1. WPG - WaveProperGator

The beamlines and the optical equipment are usually designed using specific codes. In this sense WavePropaGator (WPG) interface is an appropriate tool to understand the behaviour of diffraction effects through the optics along the beamline. WPG is a framework for Synchrotron Radiation Workshop (SRW) developed in the European XFEl for beamline scientist [3]. The framework allows defining optics along the beamlines and propagate a Gaussian source through them. Three main parts can be differentiated in the WPG Scripts:

- Source: A Gaussian beam is defined in the framework using the inputs parameters source size and divergence.

- Optical elements: A sequence of the optical elements is saved in an external file called "beamline.py". The file contains the information about incident angles, distances between optical elements and dimensions among other parameters.

- Detector: The projection of the photon beam at the end of the beamline is calculated using the appropriate scaling parameters to observe the final intensity and phase distribution.

The simulations are based on the principle of physical optics using Fast Fourier Transform (FFT) and asymptotic expansion based on a propagator. The interface provides accurate results respect to the shape and dimensions of the photon beam indeed. It includes information about the size of the optics, angle of incidence and roughness of the optics. WPG provides information about the phase and the intensity. However, the latter is unfortunately given in arbitrary units and is independent on the intensity in the input since data needed to calculate the real final photon intensity is missed, i.e., optical coating. An add-up code to supply with such an information the code is certainly needed. In this sense, the ICS (Intensity Calculation Support) code development is a very helpful tool to calculate the intensity at the end or step-by-step along the beamline. The ICS code was tested is the beamline FL24 located at FLASH2.

1.2. Beamline FL24

The beamline FL24 is located at FLASH2. Figure 1 shows the different optical elements in the beamline:

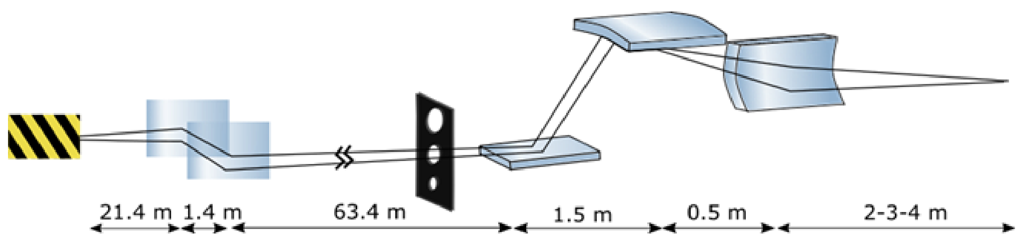


Figure 1: Sketch of Beamline FL24.

This beamline is made up of six optical elements, the first two mirrors are located in the tunnel and the following optical elements are found in the experimental hall:

- Two offset mirrors: whose function is to eliminate the Bremsstrahlung Effect produced by the deceleration of the charged electrons. They can also be used to correct the direction of the beam.

- An aperture: used to clip the beam when necessary.

- A planar mirror: also called pre-mirror, whose main function is to send the beam in the direction of the Kirk-Patrick Baez (KB) optics.

- Kirk-Patrick Baez optical focusing system: which consists in two elliptical mirrors that are disposed in a horizontal and a vertical position respectively. Their purpose is to create the smallest focus with the maximum efficiency. In order to do so, small grazing angles are used along all the optics of the beamline.

The reflectivity of these mirrors needs be taken in consideration in the calculation of the final intensity as the output, to do so, three main coating materials were considered: Platinum, Nickel and Carbon since they showed high reflectivity in the range of soft X-rays. Figure 2 shows a comparison between the different materials and their reflectivity.

As it can be observed, each of the coatings have the highest efficiency for a specific wavelength range. Platinum is the best option if the wavelength selected is very small (up to 2 nm), followed by Nickel that becomes the best option in the range from 2 to 5 nm. Finally for larger wavelengths (>5nm) Carbon shows a high and stable value.

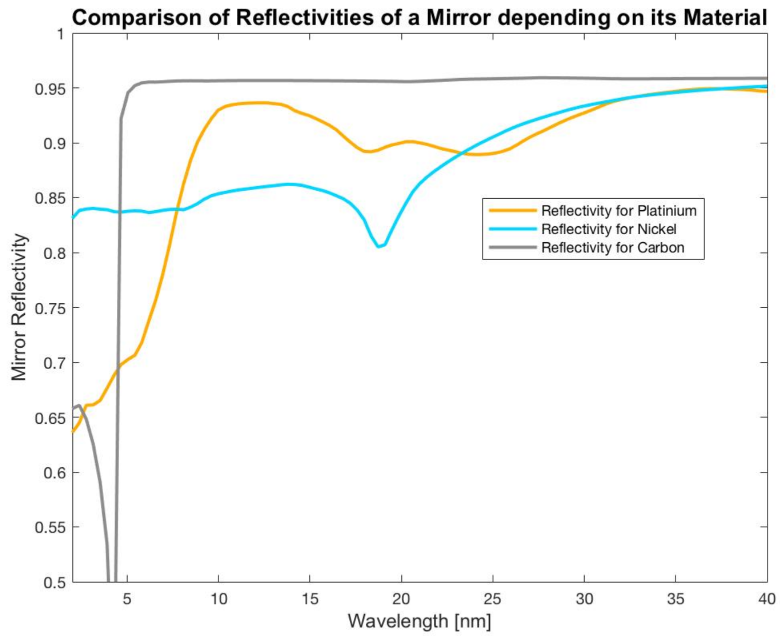


Figure 2: Reflectivity at incident angle θ=2 degrees for Nickel (Ni), Carbon (C) and Platinum (Pt)

2. ICS: Intensity Calculation Support

The code is based on ray tracing technique. By using a very simple formula that considers the reflectivity of each mirror and the fraction of the beam at the mirror clipped by the surface of the mirror.

$$I = \prod_{i=0}^{O.E.} \rho_i \dot{} I_i \tag{1}$$

where I is the final intensity, ρ_i is the reflectivity of the coatings and I_i is the partial intensity of the optical element.

If the diffraction effects are considered, the formula becomes as follows:

$$I = \prod_{i=0}^{O.E.} \rho_i \dot{} I_i \dot{} D.E._i \tag{2}$$

where the diffraction effects are also included (D.E.$_i$)

As an schematic way to describe it, the code obtains information from three main sources:

- CXRO Webpage: This web page contains information about the reflectivity of the mirrors for a large range of wavelengths based on different parameters: the material, density, roughness, polarization and the incidence angle.

- Beamline.py: This file is created in WPG and saved after running a beamline simulation. It contains information about the sizes of the optical elements, the incidence angle of the beam and also the distance between the optical elements.
- Graphical User Interface: Obtains the missing data needed in order to proceed: the material of the optical elements, the path to the beamline file and the divergence and wavelength used.

The code can be subdivided in three different parts: the GUI where the user can write the input parameters, the main function, that analyzes the data obtained from the GUI and also activates the Mirror Ref function, whose purpose is to obtain the refractivities from the CXRO page.

2.1. GUI - Graphical User Interface

Figure 3 shows the graphical user interface.
This window firstly allows the user selecting the number of optical elements from the beamline. After that the user can select the type of optical element and the coating for them. In case of apertures, the coating is not used. The selection needs to be done in the correct sequence of elements from the source to the detector.
Once the optical elements and their coatings are selected, one has to select the path to the file *Beamline.py*. Finally, the user need to introduce the desired wavelength as well as the divergence. Both parameters can be also found in the dedicated script of WPG. After all the parameters are selected, the final *RUN* button starts the MAIN function that leads to the corresponding final intensity at the end of the beamline

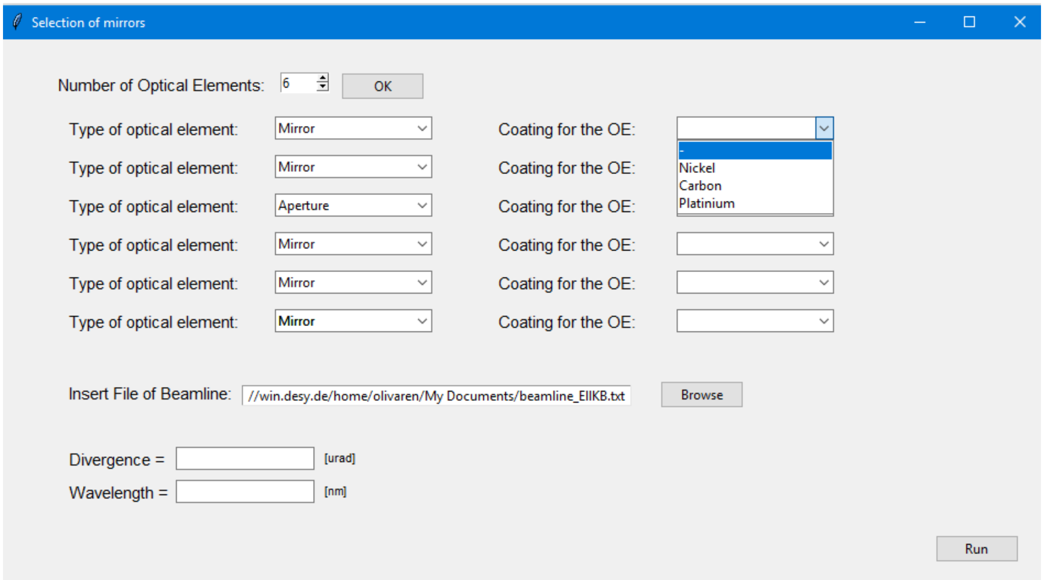


Figure 3: Print-out of the GUI for the ICS (Intensity Calculation Support) code

2.2. Main Function

The Main Function code consists in a function that processes and analyzes all the input information to calculate the final intensity. As a first step it collects all the relevant information (incident angle, dimensions of the optics and distances between optical elements) from the "Beamline.py" file. To do so, the code searches for specific notations inside the "Beamline.py" file. For example, in order to find the distances between the optical elements, the code looks for the words "Drifts" and store the parameter written near to it. For the angle of incidence, the information must be written as: "theta". And the dimensions of the optical elements must be disposed as: "Dx" and "Dy".
After going through different filtering methods, the main information of the file is stored in a DataFrame as the one shown in Figure 4.

Index	Distances	Incangles	Material	Shape	X Dimension	Y Dimension
Mirror 1	21.40	0.00060912	N	c	0.8	0.8
Mirror 2	22.8	0.00060912	N	c	0.8	0.8
Aperture	84.8	1.5708	-	c	0.014	0.014
Mirror 3	86.2	0.00060912	N	c	0.5	0.5
Mirror 4	87.7	0.00060912	N	e	0.36	0.02
Mirror 5	88.25	0.00060912	N	e	0.36	0.02

Figure 4: DataFrame containing information about the optical elements in beamline FL24: Distances, Incidence angles, Material and Dimensions

The second step is calculating the difference between the beam size and the optical dimensions for each element. The output of the partial intensity of each mirror is given as a vector that represents the percentage of area of the beam inside the area of the element. For low divergence the beam would be smaller than the dimensions of the optics and the so called divergence efficiency is 1 (100%).

2.3. MirrorRef Function

In order to calculate the reflectivity of the mirror coatings a new function called "MirrorRef" is used. For this case the Data Frame contains information about the reflectivity of the optical elements for the specified coating material, obtained from the web page of the Center of X-Ray optics (CXRO[1]).
The interaction between the code MirrorRef and the web of CXRO is done using the module Selenium [4]. This module allows the user pre-selecting the actions that are after done in that web page, so the input is automatic.

[1] CXRO main page

The function accesses to the "thick mirror" option on the web and writes in it the given parameters [2]. Afterwards, it returns a data file that contains the information about reflectivity for such a mirror. This file is used in a new *DataFrame* in the python main function.

3. Results

In order to study the behaviour of the code, we analyze the total intensity at the end of the beamline FL24. Values are given normalized in decimals. Different cases, where different divergences and different apertures were chosen are used.
In Table 1 we compare two cases: one where the parameters used as the same as in [5] and one where the divergence is θ=200 μrad.

Optical Element	Intensity
Mirror 1	0.999947
Mirror 2	0.999947
Aperture	1
Mirror 3	0.999947
Mirror 4	0.999947
Mirror 5	0.999947
Total Intensity	0.999735
Divergence= 125 μrad	

Optical Element	Intensity
Mirror 1	0.999947
Mirror 2	0.999947
Aperture	0.681404
Mirror 3	0.999947
Mirror 4	0.999947
Mirror 5	0.999947
Total Intensity	0.681223
Divergence= 200 μrad	

Table 1: Total intensity calculated for, on the left, the case 1 with divergence θ=125μrad and on the right the case 2 with divergence θ=200μrad. As observed the case 2 only provides approx. 70% of the initial intensity.

The intensity at the end of the beamline varies dramatically from the Case 1 to Case 2. As observed the case 2 only provides 68% of the initial intensity. Both calculations use the same coatings (Nickel), optical dimensions (diameter of aperture: $14 * 10^{-3}$) and the wavelength (8 nm).

The studied case 3 and case 4 compare the final intensity for different aperture diameters. As observed the case 3 provides approx. 30% of the initial intensity. Both calculations use the same coatings (Nickel), divergence (125 μrad) and the wavelength (8 nm):

[2]CXRO - Thick mirror datapage

Optical Element	Intensity
Mirror 1	0.999947
Mirror 2	0.999947
Aperture	1
Mirror 3	0.999947
Mirror 4	0.999947
Mirror 5	0.999947
Total Intensity	0.999735
Diameter= 14 mm	

Optical Element	Intensity
Mirror 1	0.999947
Mirror 2	0.999947
Aperture	0.302846
Mirror 3	0.999947
Mirror 4	0.999947
Mirror 5	0.999947
Total Intensity	0.287959
Diameter= 7 mm	

Table 2: Results of output intensity depending on aperture size for a wavelength of 8 nm and Divergence=150 μrad

A 14 mm diameter is sufficient for a divergence smaller than 125 μrad, however, the final number of photons decreases 1/3 when the aperture is 7 mm instead.

Figure 5 shows the output result for the Total Intensity calculated with ray tracing for a range of aperture diameter from 2 to 15 mm. We observe that the intensity is scale down fast for aperture smaller than 10 mm at normal conditions, i.e., Divergence= 150 μrad and wavelength= 8 nm.

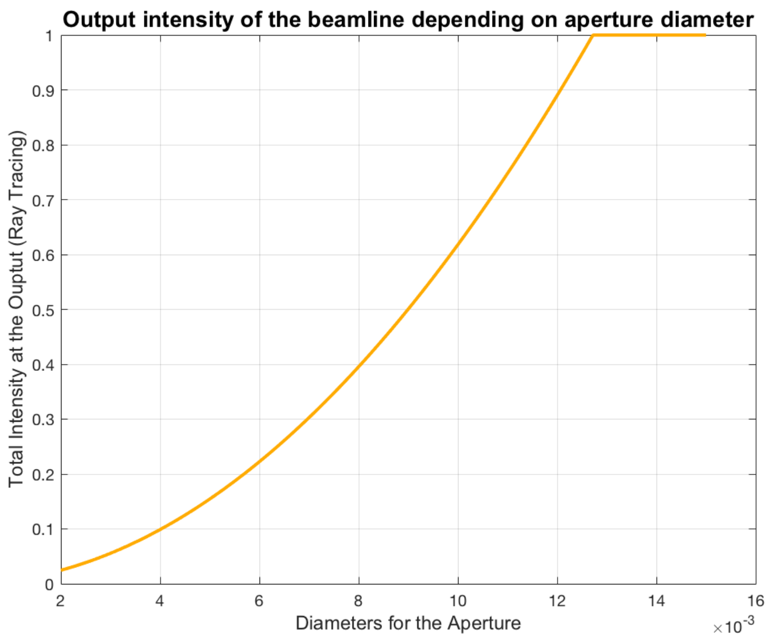


Figure 5: Calculation of the Total Intensity of the Beamline in the common conditions

4. Future Work: Diffraction Effects

The diffraction effects are not included in the ICS code, however since the framework WPG is based on wave-propagation, we believe that is important to understand how the can affect the final intensity.
To calculate the difference between the intensity given by ray tracing and wavefront propagation, we observed the diffraction effects produced after an aperture illuminated with a source at 21,40 m. In the near field, an aperture illuminated with coherent radiation project an Airy disc. In other words the diffraction pattern will correspond to the beam surrounded by several circles around it, whose size depends on the level of diffraction. The program simulates this in a pretty accurate way as it can be seen in the following picture:

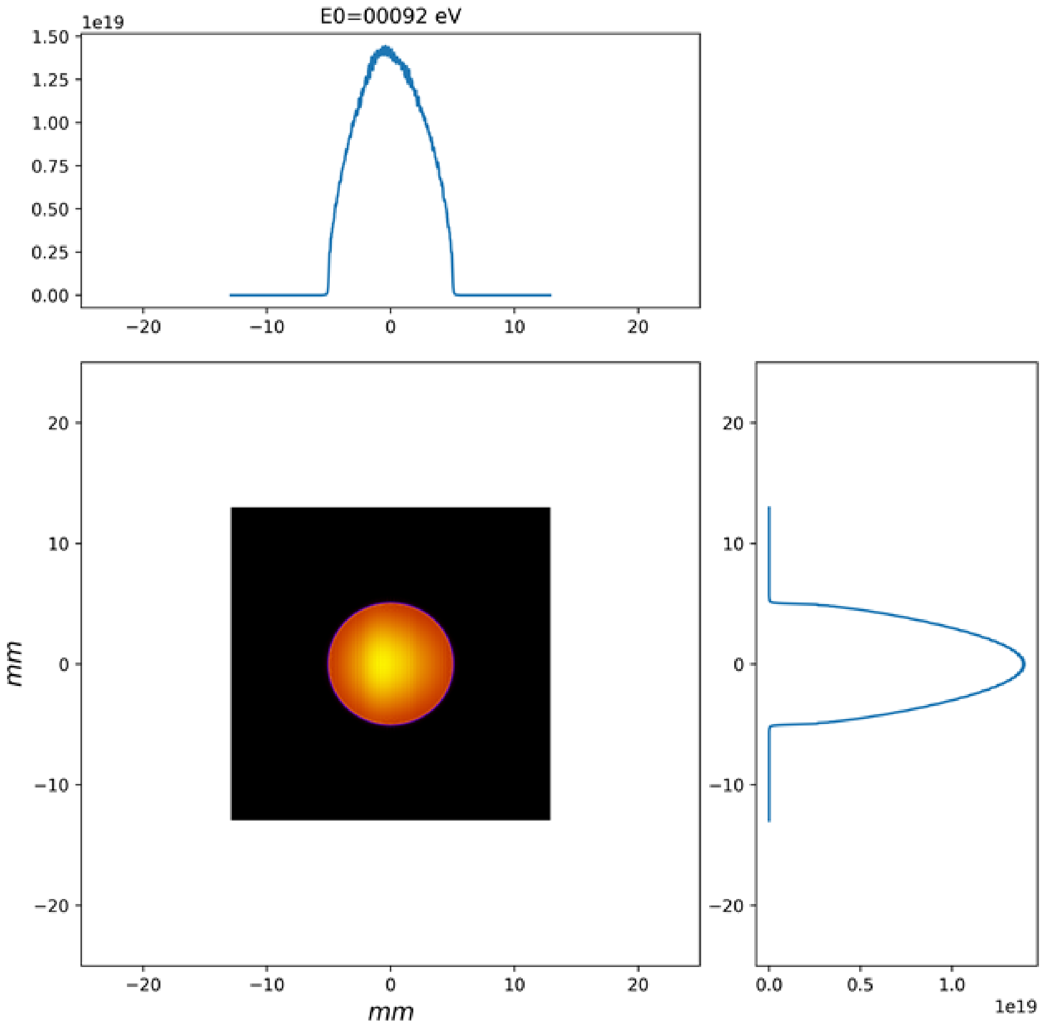
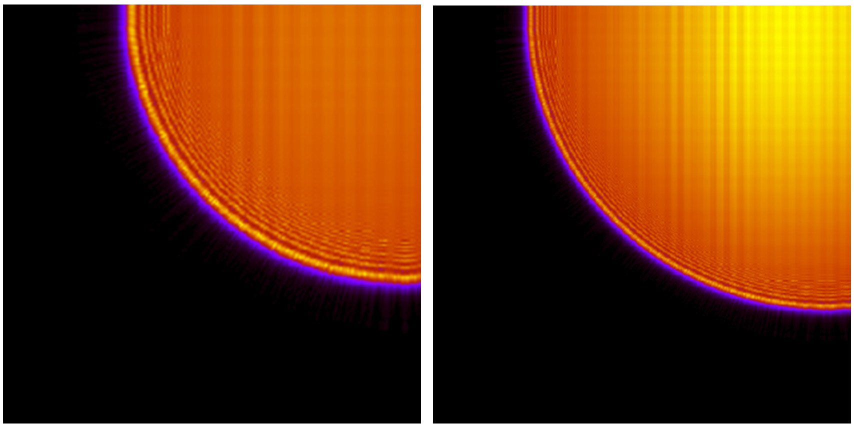


Figure 6: Beam Wavefront after an aperture of D=10 mm

When the image of the wavefront is zoomed in, the circular shapes around it can be easily appreciated. In Fig. 7 we shows the diffraction rings of an aperture of 6 and 10 mm of diameter.:



(a) Aperture size = 6 mm (b) Aperture size = 6 mm

Figure 7: Diffraction Effects on Different aperture sizes

With this patterns of diffraction, a quantification of this effect is possible by processing the images obtained. The process of obtaining a result for this effect can be summarized in the following steps:

- Simulating two wavefronts: one which propagates through the aperture, one Gaussian wavefront at the position of the aperture, but without any structure producing in it a diffraction pattern.
- Process those images by integrating, vertically and horizontally.
- Calculating the ratio between the diffracted beam and the non-diffracted one.

By analyzing the results in the following apertures, it was observed that the diffraction effects do make a change in the calculation of the Total Intensity, smaller than 30%:

Aperture Size	6 mm	7,5 mm	10 mm
Total Intensity (Ray Tracing)	0.22244	0.34756	0.61789
Diffraction Effects	0.8174	0.8346	0.8714
Total Intensity	**0.1818**	**0.2901**	**0.5384**

Table 3: Results of output intensity considering diffraction effects for different aperture sizes.

5. Conclusions

The ICS code based on Python structures has been developed to calculate the total intensity at the end of the beamlines. For this aim, the code uses information created previously in the WPG framework. Likewise, the code collects information about the coating efficiency from the web page of the CXRO.

We have tested ICS for different situations at the beamline FL24. However the code can easily applied to other beamlines with a different disposal of elements. To do that some parameters are nonspecific to the beamline FL24.

Although the efficiency is calculated using the ray tracing method, a diffraction case was calculated separately and compared with the results obtained in our code. We observed that the total number of photons is reduced at least 20% respect to the ray tracing calculation. In conclusion ICS code is a very useful tool to have a preliminary idea of the total intensity at the end of the beamlines.

References

[1] Wet al Ackermann, G Asova, V Ayvazyan, A Azima, N Baboi, J Bähr, V Balandin, B Beutner, A Brandt, A Bolzmann, et al. Operation of a free-electron laser from the extreme ultraviolet to the water window. *Nature photonics*, 1(6):336, 2007.

[2] B Faatz, E Plönjes, S Ackermann, A Agababyan, V Asgekar, V Ayvazyan, S Baark, N Baboi, V Balandin, N von Bargen, et al. Simultaneous operation of two soft x-ray free-electron lasers driven by one linear accelerator. *New journal of physics*, 18(6):062002, 2016.

[3] Liubov Samoylova, Alexey Buzmakov, Oleg Chubar, and Harald Sinn. Wavepropagator: interactive framework for x-ray free-electron laser optics design and simulations. *Journal of applied crystallography*, 49(4):1347–1355, 2016.

[4] Richard Lawson. *Web scraping with Python*. Packt Publishing Ltd, 2015.

[5] Mabel Ruiz-Lopez, Liubov Samoylova, Günter Brenner, Masoud Mehrjoo, Bart Faatz, Marion Kuhlmann, Luca Poletto, and Elke Plönjes. Wavefront-propagation simulations supporting the design of a time-delay compensating monochromator beamline at flash2. *Journal of synchrotron radiation*, 26(3), 2019.

A. ICS Codes

A.1. GUI

```
# -*- coding: utf-8 -*-
"""
Created on Wed Aug 21 11:33:39 2019

@author: olivaren
"""
import tkinter
from tkinter import ttk
from tkinter.filedialog import askopenfilename
import numpy as np

#import os
from MAINDEFFUN import maindeffun

matr=np.empty([2, 6], dtype=str)

window = tkinter.Tk()
window.title("GUI")

def closewindow ():
    global matr
    global Divergence
    global wavelen
    global Efficiency
    global Effi
    for ii in range (numirr):
        matr[0,ii]=globals()['cb{}'.format(ii)].get()
        matr[1,ii]=globals()['cbm{}'.format(ii)].get()
    Divergence=e1.get()
    wavelen=e2.get()
    window.destroy()
    Efficiency,Effi=maindeffun(name, matr, Divergence, numirr, wavelen)
    print(Efficiency)

def OpenFile():
    global name
    name = askopenfilename(initialdir="C:/Users/Batman/Documents/",
   filetypes =(("Text File", "*.txt"),("All Files","*.*")), title = "
   Choose a file.")
    try:
        with open(name,'r') as UseFile:
            UseFile.read()
        file.configure(text=name)
    except:
        print("No File Selected")

def okay ():
    global numirr
```

```python
    numirr=int(spin.get())
    inih=70
    for i in range (numirr):
        element=tkinter.Label(window, text="Type of optical element:", fg='black', font=("Helvetica", 11))
        element.place(x=60, y=inih+35*i)
        typo=tkinter.StringVar()
        globals()['cb{}'.format(i)]=ttk.Combobox(window, textvariable=typo)
        globals()['cb{}'.format(i)]['values']=("Mirror", "Aperture", "VLS grating")
        globals()['cb{}'.format(i)].place(x=250, y=inih+35*i)

        element=tkinter.Label(window, text="Coating for the OE:", fg='black', font=("Helvetica", 11))
        element.place(x=450, y=inih+35*i)
        material=tkinter.StringVar()
        globals()['cbm{}'.format(i)]=ttk.Combobox(window, textvariable=material)
        globals()['cbm{}'.format(i)]['values']=("-","Nickel", "Carbon", "Platinium")
        globals()['cbm{}'.format(i)].place(x=615, y=inih+35*i)

label=tkinter.Label(window, text='Number of Optical Elements:', fg='black', font=("Helvetica", 11))
label.place(x=50, y=30)
spin = tkinter.Spinbox(window, from_=0, to=100, width=5)
spin.place(x=255, y=30)
button = ttk.Button(text = "OK", command=okay)
button.place(x=310, y=30)

a=370
label=tkinter.Label(window, text='Insert File of Beamline:', fg='black', font=("Helvetica", 11))
label.place(x=60, y=a-60)
buttn=ttk.Button(window, text="Browse", command=OpenFile)
buttn.place(x=600, y=a-60)
file = tkinter.Label(window, text='No File', borderwidth=1, background='white', relief="groove", height=1, width=50)
file.place(x=220, y=a-56)

label=tkinter.Label(window, text='Divergence =', fg='black', font=("Helvetica", 11))
label.place(x=60, y=a)
e1 = ttk.Entry(window)
e1.place(x=160, y=a)
label=tkinter.Label(window, text='[urad]', fg='black', font=("Helvetica", 8))
label.place(x=292, y=a)

label=tkinter.Label(window, text='Wavelength =', fg='black', font=("Helvetica", 11))
```

```
label.place(x=60, y=a+30)
e2 = ttk.Entry(window)
e2.place(x=160, y=a+30)
label=tkinter.Label(window, text='[nm]', fg='black', font=("Helvetica", 8))
label.place(x=292, y=a+30)

button = ttk.Button(text = "Run", command=closewindow)
button.place(x=850, y=450)

window.title('Selection of mirrors')
window.geometry("950x500+10+10")
window.mainloop()
```

A.2. Main Function

```
# -*- coding: utf-8 -*-
"""
Created on Fri Aug 23 17:27:02 2019

@author: olivaren
"""
#from mirrorRef2 import thickmirror2
#import numpy as np
#import pandas as pd
#import math
def maindeffun(name, matr, Divergence, numirr, wavelen):
    from mirrorRef2 import thickmirror2
    import numpy as np
    import pandas as pd
    import math
    # FIXED PARAMETERS:
    lambdamin='2.0'
    lambdamax='40.0'
    lambdanum='100'
    density=-1
    roughness='0.3'
    # VARIABLES (ASKED IN THE GUI):
    pol=1

    # MATR, OBTAINED IN GUI: WE WILL SUPOSE NO VLS GRATINGS
    f=open(name, "r")
    mylines = []                          # Declare an empty list named mylines.
    with open ('beamline_EllKB.txt', 'rt') as myfile: # Open lorem.txt for reading text data.
        for myline in myfile:             # For each line, stored as myline,
            mylines.append(myline)        # add its contents to mylines.
            f.close()
            #........................................................................
```

```
    # OBTAIN GENERAL PARAMETERS: (obtains general parameters of the beamline.text)
    distances=[]
    angles=[]
    incangles=[]
    leng=len(mylines)
    dimx=[]
    dimy=[]

    for linea in range(leng):
        mylines[linea]=mylines[linea].strip()
        if mylines[linea]=='#--1.1. Distances-----------':
            distances.append(mylines[linea+1].split()[2])
        if mylines[linea].find("Drift(")!=-1:
            a=mylines[linea][mylines[linea].find('Drift(')+len('Drift('):].split()[0].strip(',').strip(')')
            distances.append(a)
        if mylines[linea]=='#--1.2. Angles--------------':    #Obtain different angles that can be a parameter
            for n in range(5):
                angles.append(mylines[linea+n+2])
                angles[n]=angles[n].split()[1]
                angles[n]=angles[n].strip("=")
                angles[n]=str(float(angles[n]))
        if mylines[linea].find('theta=') != -1:              #This line obtains the name of the angle of incidence in each mirror
                incangles.append(mylines[linea])
        if mylines[linea].find("Dx")!=-1:
            try:
                dimx.append(str(float(mylines[linea].split()[4].strip(',').strip('Dx='))))
                dimy.append(str(float(mylines[linea].split()[5].strip(')').strip('Dy='))))
            except:
                dimy.append(str(float(mylines[linea].split()[6].strip(')').strip('Dy='))))
    lastmirrortoscreen=distances[len(distances)-1]
    distances=distances[0:len(distances)-1]
    # This filter finds if the angle is the one selected and obtains non-repeated parameters:
    values=[]
    for ii in range(len(incangles)):
        incangles[ii]=incangles[ii].strip(",")
        incangles[ii]=incangles[ii].split('=')[1]
        for linea in range(leng):
            if mylines[linea].find(incangles[ii]+'=') != -1:
                values.append(mylines[linea])
    values=list(set(values))

    valueangles=np.zeros((len(values),3), dtype=object)
    for k in range(len(values)):
```

```
            values[k]=values[k].split()
            values[k][0]=values[k][0].strip("=")
            values[k][2]=values[k][2].strip("#")
            valueangles[k,0]=str(values[k][0])
            valueangles[k,1]=values[k][1]
            valueangles[k,2]=values[k][2]
            if valueangles[k][2]=='[rad]':
                valueangles[k,1]=math.radians(float(valueangles[k,1]))
                valueangles[k,2]='[deg]'
    # Assigns a value of theta for each mirror:
    for j in range(len(incangles)):
        for u in range(len(valueangles)):
            if incangles[j]==valueangles[u][0]:
                incangles[j]=valueangles[u][1]
    # ...........................................................................

    # ANALIZE TYPE OF OPTICAL ELEMENTS:
    Material=[' ']*numirr          #Material of each mirror
    Shapes=['c']*numirr            #Shape circular or elliptical
    typo=[]                        #Indexing of the mirrors
    for ii in range(numirr):
        if matr[0,ii]=='M':
            typo.append('Mirror'+str(ii))
            for linea in range(40,leng):
                if mylines[linea].find(' mirror')!= -1:
                    Material[ii]=matr[1,ii]
        if matr[0,ii]=='A':      # Angle of inc of Apertures is 90(normal inc.) and with circular shape. Considers only one aperture for typo
            typo.append('Aperture')
            for linea in range(0,leng):
                if mylines[linea]== '#Aperture ----------------------------------------------------------------------------------------':
                    Material[ii]=matr[1,ii]
                    if mylines[linea+1].find("shape='c'")!= -1:
                        if mylines[linea+1].find("Dx")!= -1:
                            Shapes[ii]='c'
                            incangles.insert(ii,math.pi/2)
    for j in range(numirr):
        if dimx[j]!=dimy[j]:
            Shapes[j]='e'
    dimy=dimy[0:numirr]
    dimx=dimx[0:numirr]
    distances=distances[0:numirr]
    incangles=incangles[0:numirr]
    # CREATE DATAFRAME WITH INFO
    d={'Material': Material, 'Incangles': incangles, 'X Dimention': dimx, 'Y Dimention':dimy , 'Distances':distances, 'Shape':Shapes}
    Information=pd.DataFrame(data=d, index=typo)

    #
```

```
# DIFFRACTION EFFICIENCY
# Sum distances so that there is the total value from source to mirror:
for l in range(1,Information.shape[0]):
    Information['Distances'][l]=float(Information['Distances'][l])+float(Information['Distances'][l-1])
    #Create vector with the diffraction efficiencies:
DiffEff=[]
for k in range(Information.shape[0]):
    Beam=float(Information['Distances'][k])*math.tan(float(Divergence)/2*1e-6)
    Proj=abs(math.sin(float(Information['Incangles'][k])))*Beam
    if Information['Shape'][k]=='c':
        if Proj/float(Information['X Dimention'][k])*2>1: #mirror smaller than beam
            DiffEff.append((float(Information['X Dimention'][k])/2/Proj)**2)
        elif Proj/float(Information['X Dimention'][k])/2<=1:
            DiffEff.append(1)
    elif Information['Shape'][k]=='e':
        if Proj/float(Information['Y Dimention'][k])*2>1:
            DiffEff.append(float(Information['X Dimention'][k])*float(Information['Y Dimention'][k])/2/Proj**2)
        elif Proj/float(Information['X Dimention'][k])/2<=1:
            DiffEff.append(1)
# Supposing normal incidence in the apperture: area of ap=0.0001539 and aera of beam=0.01306
# Obtains from the web the different Reflectivities:
Ref=pd.DataFrame()
for w in range(Information.shape[0]):
    if  Information.index[w]=='Aperture':
        ref=[1]*(int(lambdanum)+1) #list de 1
    elif Information.index[w]!='Aperture':
        if Information['Material'][k]=='N':
            material='Ni'
            ref,wave=thickmirror2(lambdamin, lambdamax, lambdanum, pol, density, material, roughness, angle=str(Information['Incangles'][w]))
        if Information['Material'][k]=='P':
            material='Pt'
            ref,wave=thickmirror2(lambdamin, lambdamax, lambdanum, pol, density, material, roughness, angle=str(Information['Incangles'][w]))
        if Information['Material'][k]=='C':
            material='C'
            ref,wave=thickmirror2(lambdamin, lambdamax, lambdanum, pol, density, material, roughness, angle=str(Information['Incangles'][w]))
        if Information['Material'][k]=='-':
            print('Error: Lacking material of mirror '+str(k))
    Ref['Mirror '+str(w+1)] = pd.Series(ref)

#Only in order to create the titles for the columns:
```

```
    vec=list(Ref.columns)
    for i in range(Ref.shape[1]-1):
        if Information.index[i]=='Aperture':
            vec[i]='Aperture'
            for a in range(i,len(vec)-1):
                vec[a+1]='Mirror '+str(a+1)
    Ref.columns=vec
    Ref.index=wave
    vec.append('Total Efficiency')

    # Create Dataframe that accumulates the diff. eff. for each lambda:
    Effofeachmirror=pd.DataFrame()
    for l in range(Ref.shape[1]):
        di=[]
        for o in range(Ref.shape[0]):
            di.append(float(Ref.iloc[o,l])*float(DiffEff[l]))
        Effofeachmirror['Mirror '+str(l+1)] = pd.Series(di)
    total=[]
    for q in range(Ref.shape[0]):
        total.append(Effofeachmirror.iloc[q,:].prod())
    Effofeachmirror['Total Eff'] = pd.Series(total)
    Effofeachmirror.columns=vec
    Effofeachmirror.index=wave

    # Select the lambda:
    for p in range(Effofeachmirror.shape[0]):
        if float(wavelen)==float(Effofeachmirror.index[p]):
            Reflectivity=list(Effofeachmirror.iloc[p])
            break
        elif float(wavelen)>float(Effofeachmirror.index[p]) and float(wavelen)<float(Effofeachmirror.index[p+1]):
            Reflectivity=(Effofeachmirror.iloc[p]+Effofeachmirror.iloc[p])/2
            break

    return Reflectivity,Effofeachmirror
```

A.3. MirrorRef

```
# -*- coding: utf-8 -*-
"""
Created on Fri Jul 26 08:44:36 2019
@author: olivaren
"""
import pandas as pd
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.support.ui import Select
from selenium.webdriver.common.keys import Keys
submission_dir = 'completed_assignments'

def thickmirror2(lambdamin, lambdamax, lambdanum, pol, density, material, roughness, angle):
```

```
    webthickmirror='http://henke.lbl.gov/optical_constants/mirror2.html'
#    driver = webdriver.Chrome(ChromeDriverManager().install())
    driver = webdriver.Chrome()
    driver.get(webthickmirror)
    # Selecting Variable Options
    formula_box = driver.find_element_by_name('Formula')
    formula_box.clear()
    formula_box.send_keys(material)
    density_box = driver.find_element_by_name('Density')
    density_box.clear()
    density_box.send_keys(density)
    roughness_box = driver.find_element_by_name('Sigma')
    roughness_box.clear()
    roughness_box.send_keys(roughness)
    pol_box = driver.find_element_by_name('Pol')
    pol_box.clear()
    pol_box.send_keys(pol)
    select = Select(driver.find_element_by_name('Scan'))
    select.select_by_value('Wave')
    lambdamin_box = driver.find_element_by_name('Min')
    lambdamax_box = driver.find_element_by_name('Max')
    step_box = driver.find_element_by_name('Npts')
    lambdamin_box.send_keys(Keys.BACKSPACE*2)
    lambdamin_box.send_keys(lambdamin)
    lambdamax_box.send_keys(Keys.BACKSPACE*4)
    lambdamax_box.send_keys(lambdamax)
    step_box.send_keys(Keys.BACKSPACE*3)
    step_box.send_keys(lambdanum)
    angle_box = driver.find_element_by_name('Fixed')
    angle_box.clear()
    angle_box.send_keys(angle)
    enter_button = driver.find_element_by_xpath("/html/body/form/input")
    enter_button.click()
    # in case of pop ups: alert = driver.switch_to_alert()
    driver.switch_to_window(driver.window_handles[1])
    link = driver.find_element_by_link_text('data file here')
    link.click()
    url = driver.current_url
    test = pd.read_csv(url, skiprows=2, names=['Wl'])
    test.dropna(inplace = True)
    new = test["Wl"].str.split(" ", n = 4, expand = True)
    new.drop(new.columns[[0,1,2]], axis=1, inplace=True)
    new.columns=['Wavelength', 'Refl']
    new['Wavelength'] = pd.to_numeric(new['Wavelength'])
    new['Refl'] = pd.to_numeric(new['Refl'])

    Reflectivity=new.Refl.tolist()
    Wave=new.Wavelength.tolist()
    return Reflectivity,Wave
    driver.close()
```